



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Ph.D. DISSERTATION

Autonomous Navigation Based on
Imitation Learning with Look-ahead Point
for Semi-structured Environment

준정형화된 환경에서 Look-ahead Point를 이용한
모방학습 기반 자율 내비게이션 방법

BY

JOONWOO AHN

FEBRUARY 2023

DEPARTMENT OF TRANSDISCIPLINARY STUDIES
THE GRADUATE SCHOOL OF CONVERGENCE
SCIENCE AND TECHNOLOGY
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Autonomous Navigation Based on
Imitation Learning with Look-ahead Point
for Semi-structured Environment

준정형화된 환경에서 Look-ahead Point를 이용한
모방학습 기반 자율 내비게이션 방법

BY

JOONWOO AHN

FEBRUARY 2023

DEPARTMENT OF TRANSDISCIPLINARY STUDIES
THE GRADUATE SCHOOL OF CONVERGENCE
SCIENCE AND TECHNOLOGY
SEOUL NATIONAL UNIVERSITY

Autonomous Navigation Based on Imitation Learning with Look-ahead Point for Semi-structured Environment

준정형화된 환경에서 Look-ahead Point를 이용한
모방학습 기반 자율 내비게이션 방법

지도교수 박 재 흥
이 논문을 공학박사 학위논문으로 제출함

2023년 1월

서울대학교 대학원

융합과학부

안 준 우

안준우의 공학박사 학위 논문을 인준함

2022년 11월

위원장	곽 노 준	(인)
부위원장	박 재 흥	(인)
위원	이 원 중	(인)
위원	송 봉 섭	(인)
위원	신 민 용	(인)

Abstract

This thesis proposes methods for performing autonomous navigation with a topological map and a vision sensor in a parking lot. These methods are necessary to complete fully autonomous driving and can be conveniently used by humans. To implement them, a method of generating a path and tracking it with localization data is commonly studied. However, in such environments, the localization data is inaccurate because the distance between roads is narrow, and obstacles are distributed complexly, which increases the possibility of collisions between the vehicle and obstacles. Therefore, instead of tracking the path with the localization data, a method is proposed in which the vehicle drives toward a drivable area obtained by vision having a low-cost.

In the parking lot, there are complicated various static/dynamic obstacles and no lanes, so it is necessary to obtain an occupancy grid map by segmenting the drivable/non-drivable areas. To navigating intersections, one branch road according to a global plan is configured as the drivable area. The branch road is detected in a shape of a rotated bounding box and is obtained through a multi-task network that simultaneously recognizes the drivable area. For driving, imitation learning is used, which can handle various and complex environments without parameter tuning and is more robust to handling an inaccurate perception result than model-based motion-planning algorithms. In addition, unlike existing imitation learning methods that obtain control commands from an image, a new imitation learning method is proposed that learns a look-ahead point that a vehicle will reach on an occupancy grid map. By using this point, the data aggregation (DAgger) algorithm that improves the performance of imitation learning can be applied to autonomous navigating without a separate joystick, and the expert can select the optimal action well even in the human-in-

loop DAgger training process. Additionally, DAgger variant algorithms improve DAgger’s performance by sampling data for unsafe or near-collision situations. However, if the data ratio for these situations in the entire training dataset is small, additional DAgger iteration and human effort are required. To deal with this problem, a new DAgger training method using a weighted loss function (WeightDAgger) is proposed, which can more accurately imitate the expert action in the aforementioned situations with fewer DAgger iterations. To extend DAgger to dynamic situations, an adversarial agent policy competing with the agent is proposed, and a training framework to apply this policy to DAgger is suggested. The agent can be trained for a variety of situations not trained in previous DAgger training steps, as well as progressively trained from easy to difficult situations.

Through vehicle navigation experiments in real indoor and outdoor parking lots, limitations of the model-based motion-planning algorithms and the effectiveness of the proposed method to deal with them are analyzed. Besides, it is shown that the proposed WeightDAgger requires less DAgger performance and human effort than the existing DAgger algorithms, and the vehicle can safely avoid dynamic obstacles with the DAgger training framework using the adversarial agent policy. Additionally, the appendix introduces a vision-based autonomous parking system and a method to quickly generate the parking path, completing the vision-based autonomous valet parking system that performs driving as well as parking.

keywords: Vision-based Navigation, Look-ahead Point, Multi-task Perception Network, Imitation Learning, Data Aggregation Algorithm, WeightDAgger, Adversarial Agent Policy

student number: 2016-26039

Contents

Abstract	i
Contents	iii
List of Tables	viii
List of Figures	ix
1 INTRODUCTION	1
1.1 Autonomous Driving System and Environments	1
1.2 Motivation	4
1.3 Contributions of Thesis	6
1.4 Overview of Thesis	8
2 MULTI-TASK PERCEPTION NETWORK FOR VISION-BASED NAVIGATION	9
2.1 Introduction	9
2.1.1 Related Works	10
2.2 Proposed Method	13
2.2.1 Bird’s-Eye-View Image Transform	14
2.2.2 Multi-Task Perception Network	15
2.2.2.1 Drivable Area Segmentation (Occupancy Grid Map (<i>OGM</i>))	16

2.2.2.2	Rotated Road Bounding Box Detection	18
2.2.3	Intersection Decision	21
2.2.3.1	Road Occupancy Grid Map (OGM_{road})	22
2.2.4	Merged Occupancy Grid Map (OGM_{mer})	23
2.3	Experiment	25
2.3.1	Experimental Setup	25
2.3.1.1	Autonomous Vehicle	25
2.3.1.2	Multi-task Network Setup	27
2.3.1.3	Model-based Branch Road Detection Method	29
2.3.2	Experimental Results	30
2.3.2.1	Quantitative Analysis of Multi-Task Network	30
2.3.2.2	Comparison of Branch Road Detection Method	31
2.4	Conclusion	34

3 DATA AGGREGATION (DAGGER) ALGORITHM WITH LOOK-AHEAD POINT FOR AUTONOMOUS DRIVING IN SEMI-STRUCTURED ENVIRONMENT 35

3.1	Introduction	35
3.2	Related Works & Background	41
3.2.1	Dagger Algorithms for Autonomous Driving	41
3.2.2	Behavior Cloning	42
3.2.3	Dagger Algorithm	43
3.3	Proposed Method	45
3.3.1	Dagger with Look-ahead Point Composition (State & Action)	45
3.3.2	Loss Function	49
3.3.3	<i>Data-sampling Function</i> in <i>Dagger</i>	50
3.3.4	Reasons to Use Look-ahead Point As Action	52
3.4	Experimental Setup	54

3.4.1	Driving Policy Network Training	54
3.4.2	Model-based Motion-Planning Algorithms	56
3.5	Experimental Result	57
3.5.1	Quantitative Analysis of Driving Policy	58
3.5.1.1	Collision Rate	58
3.5.1.2	Safe Distance Range Ratio	59
3.5.2	Qualitative Analysis of Driving Policy	60
3.5.2.1	Limitations of <i>Tentacle</i> Algorithm	60
3.5.2.2	Limitations of <i>VVF</i> Algorithm	61
3.5.2.3	Limitations of Both <i>Tentacle</i> and <i>VVF</i>	62
3.5.2.4	Driving Results on Noisy Occupancy Grid Map	63
3.5.2.5	Intersection Navigation	65
3.6	Conclusion	68

4 WEIGHT DAGGER ALGORITHM FOR REDUCING IMITATION LEARNING ITERATIONS 70

4.1	Introduction	70
4.2	Related Works & Background	71
4.3	Proposed Method	74
4.3.1	Weighted Loss Function in <i>WeightDagger</i>	75
4.3.2	Weight Update Process in Entire Training Dataset	78
4.4	Experiments	80
4.4.1	Experimental Setup	80
4.4.2	Experimental Results	82
4.4.2.1	Ablation Study According to τ	82
4.4.2.2	Ablation Study According to ε	83
4.4.2.3	Ablation Study According to α	84
4.4.2.4	Driving Test Results	85
4.4.3	Walking Robot Experiments	86

4.5	Conclusion	87
5	DAGGER USING ADVERSARIAL AGENT POLICY FOR DYNAMIC SITUATIONS	89
5.1	Introduction	89
5.2	Related Works & Background	91
5.2.1	Motion-planning Algorithms for Dynamic Situations	91
5.2.2	Dagger Algorithm for Dynamic Situation	93
5.3	Proposed Method	95
5.3.1	Dagger Training Framework Using Adversarial Agent Policy	95
5.3.2	Applying to Oncoming Dynamic Obstacle Avoidance Task	97
5.3.2.1	Ego Agent Policy	98
5.3.2.2	Adversarial Agent Policy	100
5.4	Experiments	101
5.4.1	Experimental Setup	101
5.4.1.1	Ego Agent Policy Training	102
5.4.1.2	Adversarial Agent Policy Training	103
5.4.2	Experimental Result	103
5.4.2.1	Performance of Adversarial Agent Policy	103
5.4.2.2	Ego Agent Policy Performance Comparisons Trained with / without Adversarial Agent Policy	104
5.5	Conclusion	106
6	CONCLUSIONS	107
	Appendix A	110
A.1	Vision-based Re-plannable Autonomous Parking System	110
A.1.1	Parking Spot Detection	112
A.1.2	Re-planning Method	113

A.2 Biased Target-tree* with RRT* Algorithm for Fast Parking Path	
Planning	115
A.2.1 Introduction	115
A.2.2 Proposed Method	117
A.2.3 Experiments	119
Abstract (In Korean)	143
Acknowledgement	145

List of Tables

1.1	Autonomous Driving Environments	3
2.1	Results of the Perception Network	31
2.2	Intersection Detection Results	32
3.1	<i>Dagger</i> Results in Actual Autonomous Vehicle	56
3.2	Collision Rate	58
3.3	Safe Distance Range Ratio	60
4.1	Accuracy According to τ	83
4.2	Accuracy Comparison According State Similarity Threshold . . .	83
4.3	Driving Test Results for <i>VanillaDagger</i> , <i>SafeDagger</i> , <i>HG-Dagger</i> , <i>EnsembleDagger</i> and with Weighted loss function	86
A.1	Comparison of Parking Spot Detection Methods	113
A.2	Result for Figure A.6(a)	122
A.3	Result for Figure A.6(b)	122
A.4	Result for Figure A.6(c)	123
A.5	Result for Figure A.6(d)	123

List of Figures

1.1	Levels of autonomous driving (NHTSA)	1
1.2	Complex and narrow parking lot and necessity for valet parking	4
1.3	Inaccuracy in obtaining localization data with DGPS, 3D LiDAR-based SLAM, and vision-based SLAM in semi-structured environments.	5
1.4	Architecture of proposed autonomous navigating system.	6
1.5	Data for vision-based autonomous navigating in semi-structured environment: (a) and (b) for navigating	7
2.1	System architecture of vision-based drivable area segmentation and branch road detection for navigation, including intersections.	13
2.2	Process of transforming front to bird’s eye view image.	14
2.3	Multi-task perception network for intersection navigation.	15
2.4	Process of drivable area segmentation to obtain occupancy grid map.	16
2.5	Rotated road bounding box detection results.	18
2.6	Non-maximum suppression algorithm.	20
2.7	Intersection decision process.	21
2.8	Processes of obtaining merged occupancy grid map (OGM_{mer}).	23
2.9	Driving Policy Network.	24
2.10	Autonomous vehicle.	25

2.11	Parking lot environments for collecting perception data.	28
2.12	Results for <i>intersection scan model (ISM)</i> method.	29
2.13	Branch roads detection results at Intersection.	32
3.1	Motion-planning algorithms and comparison between them. . . .	36
3.2	Behavior cloning and DAgger algorithms for autonomous driving.	41
3.3	System architecture of the proposed driving method.	45
3.4	Dataset collection process of imitation learning (behavior cloning).	46
3.5	Illustration of the data-sampling function of <i>DAgger</i>	47
3.6	Human labeling is effective.	52
3.7	Labeling look-ahead point on the state is more clear than using steer-acc/brake.	53
3.8	Parking lots used in the real autonomous driving experiment. . .	55
3.9	Results of safe distance range ratio.	59
3.10	Problems of the <i>tentacle</i> algorithm.	60
3.11	Problems with the <i>VVF</i> algorithm.	61
3.12	Problems for driving in narrow drivable area with large curvature changes: (a) <i>VVF</i> , (b) <i>Tentacle</i>	62
3.13	Driving results with <i>DAgger</i> and noisy occupancy grid map. . . .	64
3.14	Intersection navigating results using motion-planning algorithms.	66
3.15	Problems of model-based motion-planning algorithm at intersec- tion.	67
4.1	<i>WeightDAgger</i> application to autonomous driving.	76
4.2	Applying the weight update process (step 2) of <i>WeightDAgger</i> .	78
4.3	Semi-structured environments used for autonomous driving test created using UNREAL ENGINE4 (a) Trained environment (data collection and driving test) and (b) Un trained environment (only driving test).	80

4.4	Weight update processes in <i>WeightDagger</i> (Step 2)	81
4.5	Test results of the average accuracy according to parameter α	84
4.6	Driving test results of <i>EnsembleDagger</i> with/without weight.	85
4.7	Half-Cheetah, Ant, and Hopper walking robots in MuJoCo.	86
5.1	Original DAgger for dynamic situations	94
5.2	Proposed DAgger for dynamic situations	95
5.3	Detail proposed training framework and its effect	97
5.4	Proposed training framework is applied to dynamic situation of autonomous driving. The ego agent (white, left) avoids static and dynamic (adversarial agent) obstacles. The adversarial agent (red, right) blocks the ego agent’s driving.	99
5.5	Adversarial agent policy that gradually improves as training progresses	104
5.6	Comparison of trained ego agent policy performance with and without adversarial agent policy. The ego agent policy trained at each DAgger training step was played against $\pi_{adv_{i=10}}$ (a-b) or $\pi_{parking}$ (c), and the number of collisions was counted. (a): trained environment (parking lot), (b): un-trained environment (empty space), (c): un-trained environment (parking situation).	105
A.1	Vision-based autonomous parking system.	111
A.2	Results with (a) and without (b) re-planning method.	113
A.3	Process and limitations of target-tree* algorithm with RRT*.	116
A.4	Process of collecting label data of network, μ^* ; and process of biased target-tree* algorithm.	118
A.5	(a) Autonomous vehicle (provided by PHANTOM AI) and collision checking range and (b) Parking situations for data collection.	119

A.6	Path planning results in un-trained environments using biased target-tree* algorithm with RRT*	121
A.7	Probability of Planning Shortest Path for Planning Time	123

Chapter 1

INTRODUCTION

1.1 Autonomous Driving System and Environments




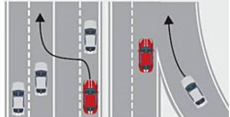


	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Level	No Automation	Driver Assistance	Partial Automation	Conditional Automation	High Automation	Fully Automation
Driver	Always Driving	Speed or Steering Support	Speed and Steering Support	Eyes Off in Some Situations	Driverless in Fair Conditions	Always Driverless
Service Time	Now	Now	Now	2020 ~ 2024	2024 ~	2030 ~
Technology	Warning 	LKAS (Lane Keeping Assist System) ACC (Adaptive Cruise Control) 	Highway Driving Assist, Autopilot 	FSD (Full Self Driving Capability) 	Automated Valet Parking 	Taxi, Shuttle 

Figure 1.1: Levels of autonomous driving (NHTSA)

National Highway Traffic Safety Administration (NHTSA) has classified autonomous vehicle technologies into six levels (see Figure 1.1) [1]. ‘Level 0, 1, and 3’ driving technologies are being applied to vehicles in structured environments such as highways or urban environments [2]. The lowest level, ‘Level 0’, is the

level without automation. An audio-visual alarm serves as an emergency notification and an auxiliary function for human driving. ‘Level 1’ assists driving concerning speed or steering on a road with lanes. A human controls the vehicle and is responsible for detecting variables in driving and for accidents. There is Adaptive Cruise Control (ACC), which maintains the vehicle speed and distance between vehicles, and Lane Keeping Assist System (LKAS), which keeps the lane while driving.

Partial automation is possible in ‘Level 2’. The steering, acceleration, and deceleration of the vehicle within a specific condition are controlled by the vehicle or human. The driver always needs to monitor the driving situation and must intervene immediately in a situation that the system is not aware of. Representative technologies at this stage include highway driving assist (HDA) of Hyundai and Autopilot of Tesla. In ‘Level 3’, the system performs driving control and situation detection, and conditional automation is possible. In most urban areas that are not complex and on highways except toll gates, the system takes charge of driving, and the driver only intervenes in case of danger. A representative technology is a full self-driving beta (FSD) in Tesla. It can automatically change the lane, join the inter or junction changes, and cope with lane changes or intervening vehicles.

High automation is possible in ‘Level 4’, and autonomous driving is possible on most roads. Technologies in semi-structured environments such as alleyways and parking lots are necessary to complete fully autonomous driving. A parking assistance system has been commercialized, and research on valet parking technology including parking lot driving is required. In ‘Level 5’, technologies up to ‘Level 4’ are integrated, and fully autonomous driving that does not require a driver is performed. The system takes care of driving in all conditions. At this level, vehicles without steering and acceleration/brake will be adapted.

Table 1.1: Autonomous Driving Environments

	Changes in Width and Curvature of Drivable Area	Lane	Topological Map Representation	Road Condition	Etc.
Un-structured Environment	Large and Varied	X	X	Un-paved	Dessert, Indoor Room, Large Space
Semi-structured Environment	Large and Varied	\triangle (center lane X)	O	Paved	Parking Lot, Wide Roundabout or Uncontrolled Intersection
Structured Environment	Small and Smooth	O	O	Paved	Highway, City Road

The environments to which autonomous driving technology can be applied are divided into the un-structured, semi-structured, and structured environments. Each environment is classified according to several criteria, as shown in Table 1.1. The un-structured environment is the most difficult environment to apply autonomous driving, and it is mainly an unpaved road environment such as a desert or an indoor room. The change in curvature and width of the drivable area is large, and it is difficult to represent with a topological-map.

Road environments where parking lots or lanes are ambiguous belong to semi-structured environment. This environment is similar to un-structured environment, but most of it is a paved road, and it is an environment that can be represented by a topological-map. Even if a topological-map level global plan can be executed, the width of the drivable area is wide and there is no center lane, so the location and direction of static and dynamic obstacles can be varied. In addition, there are cases where the shape of the drivable area is not divided into lanes, and even if it is divided, safe navigation is difficult because various obstacles such as curbs, pedestrian paths, vehicles, and pedestrians exist (see Figure 1.2). Unlike this, the structured environment is most of the roads that vehicles drive on, which is possible to cover it with researched and developed autonomous driving technology.

1.2 Motivation



Figure 1.2: Complex and narrow parking lot and necessity for valet parking

Autonomous navigating technology for semi-structured environments is at ‘Level 4’ of the aforementioned autonomous driving levels, which is important for fully autonomous driving. In particular, autonomous navigating in complex parking lots as shown in Figure 1.2 are hard for humans, which can be convenient for them. In the parking lot navigating, a vehicle drives from a point where passengers get off until it reaches a parking spot. After this step, the vehicle automatically parks through a similar process to automated parking.

Most studies on the semi-structured environment navigating including parking lot have suggested a system based on high-cost infrastructure and sensors [3–6]. In these methods, the vehicle receives a goal point using a vehicle-to-infrastructure (V2I) communication system [7–9]. To reach the goal, a global path consisting of way-points is generated using the high definition (HD) map constructed by 3D-LiDAR and depth cameras [10–12]. However, the cost of constructing and managing the infrastructure and the HD map is expensive.

The vehicle tracks the global path by acquiring localization data which is the position and direction of the vehicle related to the HD map or the global path. However, this data is inaccurate in the semi-structured environment. A

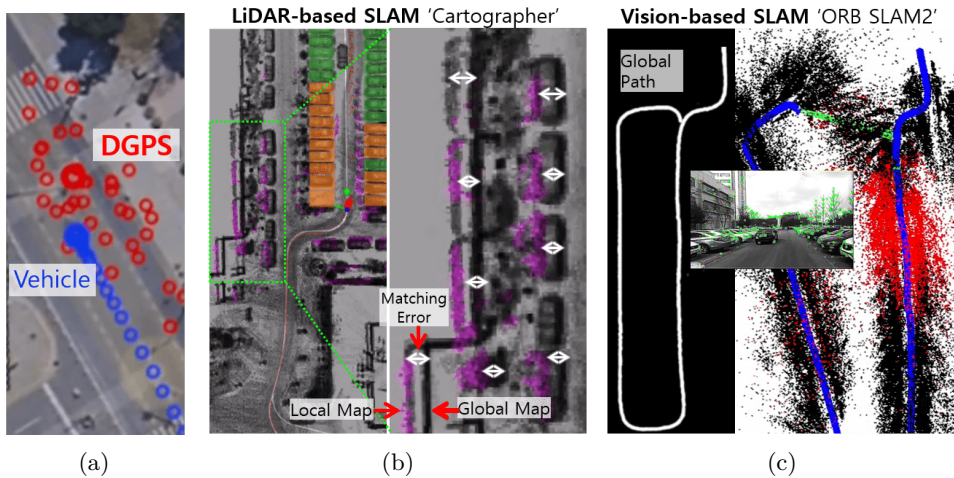


Figure 1.3: Inaccuracy in obtaining localization data with DGPS, 3D LiDAR-based SLAM, and vision-based SLAM in semi-structured environments.

differential global positioning system (DGPS) or LiDAR-based simultaneous localization and mapping (SLAM) is used to obtain it. Mostly, the global positioning system (GPS) is used and has an error of about 1 ~ 30 m. Differential GPS (DGPS) uses communication base stations with GPS and has an error of about 0.1 ~ 3 m (see Figure 1.3(a)). However, the roads in the parking lot are narrower and closer to each other than urban roads, and the error becomes high [5]. Besides, the error is larger in an area close to the building, and these cannot be used in the indoor parking lot.

The LiDAR and vision based SLAM using the state-of-the-art algorithms were tested, but these cannot get accurate localization data in the parking lot. Figure 1.3(b) is a result using the Cartographer algorithm [13] with Velodyne 64E 3D-LiDAR and IMU. A matching error occurred between the previously acquired global map and the local map acquired obtained while driving, occurring an error of about 0.1 m [6, 11]. Figure 1.3(c) shows a result of vision-based SLAM using the ORB SLAM2 [14] algorithm with *ZED* stereo camera. There

was an error on every road, especially at corners (before loop closing), and the error was more than 0.15 m [8, 15]. These algorithms also require high computation time and are difficult to ensure in real-time when the map is enlarged.

Due to the localization error, a matching error occurs between the planned path and the path that is tracked, which increases the possibility of colliding with obstacles. Therefore, there is a need for a parking lot navigation method without generating the global path and tracking it with the localization data. This eliminates the need for HD maps, infrastructure sensors, and localization data with additional sensors.

1.3 Contributions of Thesis

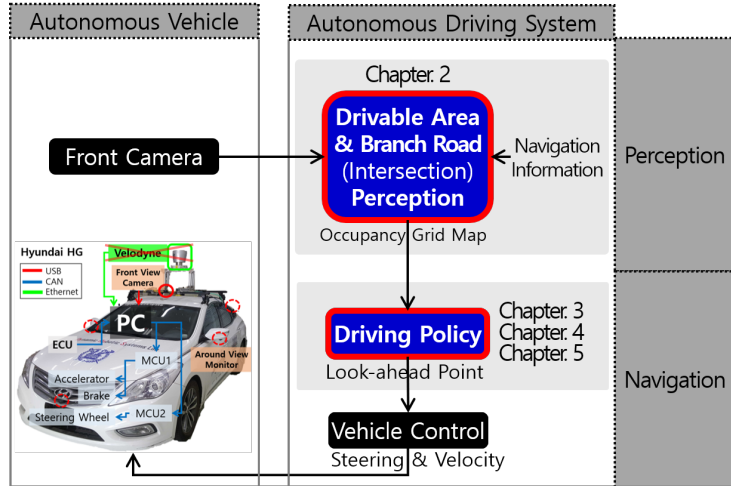


Figure 1.4: Architecture of proposed autonomous navigating system.

This thesis proposes a method to perform autonomous navigating in a semi-structured environment using only deep learning and vision sensor in order to use minimal global information. A proposed navigating method does not use global information such as the HD map, the global path, and the localization

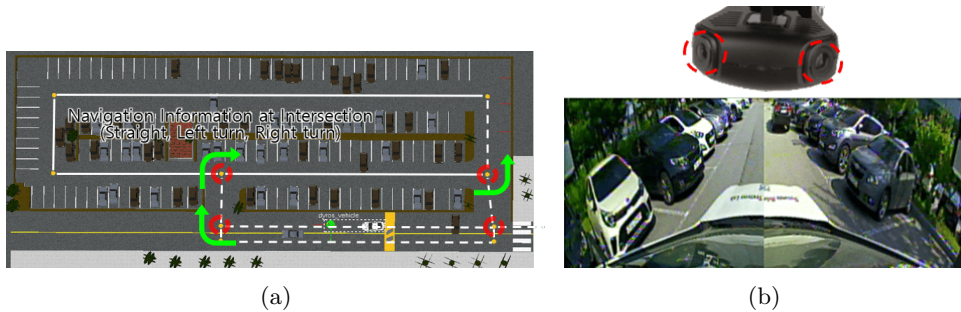


Figure 1.5: Data for vision-based autonomous navigating in semi-structured environment: (a) and (b) for navigating

data. As shown in Figure 1.5(a) and 1.5(b), only following data are used: the topological map consisting of intersections and roads, navigation information at intersections (straight/left turn/right turn), and vision.

To navigate at an intersection with a driving policy that does not consider intersections, a multi-task network is proposed. It recognizes not only the drivable area but also the branch road existing at the intersection in the form of a rotated bounding box. According to the navigation information at the intersection, one branch road is selected and combined with the drivable area to obtain the occupancy grid map consisting of one branch road.

Imitation learning is used to obtain a driving policy for avoiding obstacles and driving toward the drivable area obtained by the vision in the semi-structured environment. The proposed imitation learning method uses the look-ahead point, so the data aggregation (Dagger) algorithm, which improves the performance of imitation learning, can be applied to autonomous driving. Besides, even when the vehicle is controlled by the trained policy action in Dagger training (even in human-in-loop design), the expert can select the optimal action well. Real-world experiments show limitations of the model-based motion-planning algorithms and the effectiveness of the proposed method that is robust

to sensor noise and does not need to tune model parameters to handle various and complex environments.

A new DAgger training algorithm, DAgger with the weighted loss function (*WeightDAgger*), is proposed to more accurately imitate the look-ahead point in unsafe or near-collision situations than not using this function, requiring fewer DAgger iterations. The weight value is calculated through the discrepancy between policy and expert actions during DAgger execution. This is additionally paired to the entire training dataset with a high state similarity. Based on the weighted value, the policy is trained with a high learning rate for unsafe or near-collision situations among the entire training dataset.

In order to extend *WeightDAgger* on dynamic obstacle avoidance situations, a novel DAgger training framework is proposed, which adopts an adversarial policy having a competitive relationship with the agent policy. The proposed adversarial policy generates various data by training, which considers the performance of the agent policy that is gradually improved through DAgger. The performance of the agent and adversarial policies are updated together, and finally an agent policy with high performance is obtained.

1.4 Overview of Thesis

The remainder of this thesis is as follows. Chapter 2 shows the multi-task network for vision-based navigation. Chapter 3 explains the DAgger algorithm with the look-ahead point. The proposed DAgger training algorithm *WeightDAgger* is described in Chapter 4. For avoiding dynamic obstacles, Chapter 5 introduces the DAgger training framework using adversarial policy. The thesis is concluded in Chapter 6. Finally, Appendix A mentions a vision-based autonomous parking system.

Chapter 2

MULTI-TASK PERCEPTION NETWORK FOR VISION-BASED NAVIGATION

2.1 Introduction

This chapter introduces a detection method to apply the existing motion-planning methods to the navigation of semi-structured environments including intersections using only vision sensor. Localization data is primarily used for autonomous vehicles navigating at intersections by tracking the global path. However, this data is inaccurate in an semi-structured environment due to narrow roads and complex obstacles. Hence, instead of tracking the path, a method for detecting and driving using vision has been studied. To navigate at intersections, it is necessary to distinguish exiting roads at those intersections. Model-based detection methods recognize patterns of the branch roads; but are sensitive to sensor noise and have difficulty finding appropriate model parameters for various complex situations.

To address the limitations, his chapter proposes a vision-based branch road

detection method using deep learning that can learn various shapes of branch roads in semi-structured environments [16]. Branch roads consist of straight, left turn, and right turn roads, which are detected and are represented as a shape of rotated bounding boxes. Compared to using the un-rotated bounding box, the rotated version can detect the branch road more accurately even when the vehicle is turning at an intersection. At the intersection, one branch road is selected according to a global plan, and the inside of the selected road is regarded as the drivable area of an occupancy grid map. Furthermore, for safe navigating, obstacles inside the selected road are considered in this map by using the segmented drivable area image. This occupancy grid map is used as an input to existing motion-planning algorithms to enable intersection navigation. In addition, drivable area segmentation and branch road detection consist of a single multi-task deep neural network, which improves the learning performance of each task and reduces overall network memory usage. Experiments in real environments show that the proposed method detected the branch roads more accurately than the model-based detection method, and the vehicle drove safely at the intersection.

The rest of this chapter is organized as follows. Sec. 2.1.1 introduces motion-planning algorithms. Sec. 2.2 explains the proposed branch road detection and the multi-task network for intersection navigation. The experimental setup and results are presented in Sec. 2.3. Finally, this chapter is concluded in Sec. 2.4.

2.1.1 Related Works

The conventional autonomous navigation studies [17–19] use a global path with localization data. The global path consists of multiple waypoints that the vehicle can pass through intersections. The localization data is obtained through the global positioning system or by the simultaneous localization and mapping

technique. However, in a semi-structured environment with narrow roads or complex obstacles, the localization data can be inaccurate, which increases the possibility of collision with obstacles [20–22].

Studies have been conducted to navigate using local sensor information, referencing inaccurate localization data rather than tracking the global path [23]. There is a study [18, 19] that recognizes obstacles with the 3D laser scanner and generates an avoidance path that can follow the global path including the intersection. In [19], a local path is found on a road segmented through 3D-LiDAR by considering the global path and inaccurate localization data. These studies reduce the possibility of collisions compared to using global paths alone. However, when a large localization error occurs more than about 1 m, the global path cannot be properly reflected to find the local path, and the vehicle may not reach a destination. In addition, real-time path generation is difficult in situations with complex obstacles.

To address the problems of the model-based motion-planning methods, there are studies using deep learning [24–28]. Control commands are obtained through the deep neural network which receives the camera image and the navigation information. The deep neural networks and training data are separately configured and trained, according to the navigation information (straight, left, and right). The driving policy can be trained on data collected in various complex environments that are difficult to handle with the model-based navigation methods. Besides, the parameters in the driving policy do not need to be manually adjusted for various situations. However, these methods require much human effort and training time because the multiple training data and networks according to the navigation information are separately collected and trained.

Navigating methods by detecting road and obstacles with local sensor data, especially vision, have been studied rather than tracking the global path with

inaccurate localization data [29–32]. For motion-planning at the intersection, it is necessary to recognize the intersection and detect the branch road existing at the intersection. Model and learning-based methods have been studied for branch road detection.

The model-based detection methods recognize and distinguish the branch road according to a pattern of its shape, size, and direction. In [29–31], a road is segmented and contours of the road are recognized. The branch roads are distinguished according to the difference in position and direction between the vehicle and the road boundary, and entry points are recognized at the widest area of the branch road. Yang Yi *et al* [32] proposed a method of converting the road distance from the vehicle into a histogram using the road segmentation image. Then, the branch road is classified according to the histogram distribution. However, model-based detection methods do not accurately detect branch roads, especially in environments with changes in the width or curvature of the branch road, or in the presence of obstacles [33]. These methods are also sensitive to sensor noise in non-accurately distinguished road boundaries.

Deep learning-based methods have been studied to address challenges in the model-based method. In [34], vehicle kinematic information, point cloud acquired by 3D-LiDAR, and open street map (OSM) are used to find branch roads at an intersection using machine learning. However, this method [34] uses global data, OSM, and a high-cost 3D-LiDAR sensor. The camera image is passed through a long-term recurrent convolutional network to recognize vehicles passing through an intersection [35]. However, this method does not distinguish branch roads at the intersection. A method in [36] distinguishes branch roads in a form of region of interest on a front view image to determine whether a vehicle is driving at an intersection. However, this method is applied only in a structured environment where the width and curvature of the branch road

are constant, not in semi-structured environments. Further, detected branch roads are not applied to the intersection navigating method, and a separate intersection navigating algorithm is used.

2.2 Proposed Method

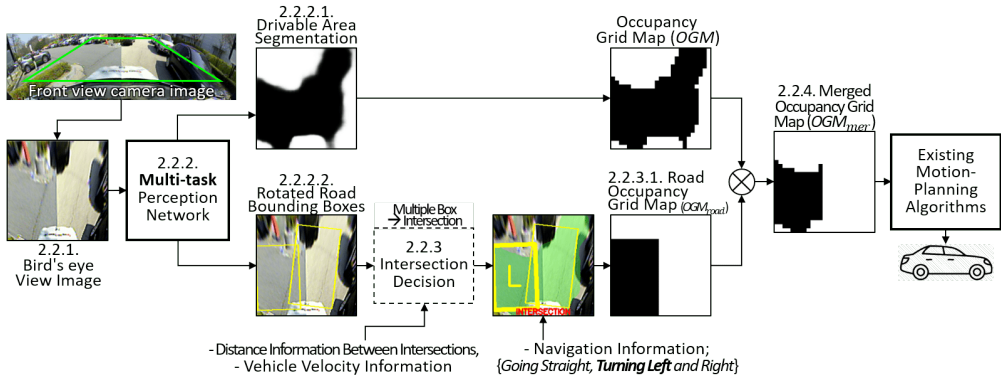


Figure 2.1: System architecture of vision-based drivable area segmentation and branch road detection for navigation, including intersections.

This chapter proposes a method to detect branch roads existing at an intersection in semi-structured environments using deep learning. In addition, a method to obtain inputs for the use of existing motion-planning algorithms at intersections is proposed by combining the data from detected branch road, global planning, and drivable area segmentation. The overall system architecture of the proposed method is shown in Figure 2.1. The method uses vision, road distance information, vehicle velocity, and navigation information. It does not use the global path and localization data. The navigation information gives the command to vehicles going straight, turning left, or turning right when a vehicle is passing through the intersection. This information is obtained through global planning using a topological map consisting of intersections and roads,

before starting the navigation [37].

A multi-task network is proposed to perform two tasks: segmenting the drivable area and detecting the branch roads as rotated bounding boxes. The drivable area segmentation image is used to obtain an occupancy grid map (OGM). Through the detected boxes, it is determined whether a vehicle is navigating at the intersection, and after which one branch road is selected using the navigation information. A road occupancy grid map (OGM_{road}) is obtained that considers the inside of the selected road as the drivable area. By merging OGM and OGM_{road} , a merged occupancy grid map (OGM_{mer}) is acquired so that the vehicle can drive toward the selected road at an intersection while avoiding obstacles. Thus, OGM_{mer} can be used as an input to existing motion-planning algorithms.

2.2.1 Bird's-Eye-View Image Transform

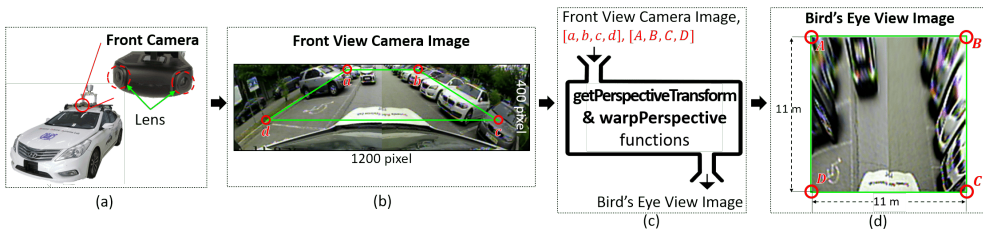


Figure 2.2: Process of transforming front to bird's eye view image.

A camera sensor consists of two lenses and is used to detect drivable/non-drivable areas as well as branch roads at an intersection. The front view camera image is transformed to the bird's eye view image which is depicted in Figure 2.2. The world coordinates ($[A, B, C, D]$), as seen in Figure 2.2(d)) can be calculated to obtain an $11\text{ m} \times 11\text{ m}$ occupancy grid map by considering the vehicle position. The world-to-pixel coordinate relationship is obtained us-

ing the extrinsic parameters, polynomial coefficients, and the camera position and orientation from the vehicle. Through the relationship, a trapezoidal shape ROI, which is the coordinate of the pixel corresponding to the obtained world coordinate $([a, b, c, d])$, the green area shown in Figure 2.2(a)), is calculated. As shown in Figure 2.2(b), the front view camera image vertices $([a, b, c, d])$ and vertices of the bird’s eye view $([A, B, C, D])$, as seen in Figure 2.2(c)) are passed through a *getPerspectiveTransform* function which aids in the acquisition of a perspective transform matrix. The bird’s eye view image is obtained by using a *warpPerspective* function with the obtained matrix. These functions are in the OpenCV library. The size of this image is 200×200 pixels, and its real world size is 11×11 m.

2.2.2 Multi-Task Perception Network

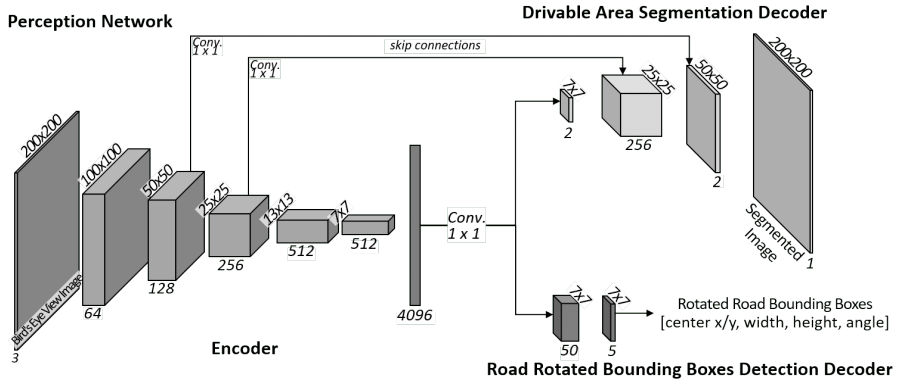


Figure 2.3: Multi-task perception network for intersection navigation.

The perception network receives the transformed image of the bird’s eye view. As shown in Figure 2.3, the perception network shares one encoder, which is used in YOLOP [38], DLT-Net [39], and Multinet [40]. Then, the output of the encoder is passed on to two decoders. Each decoder performs the drivable area

segmentation and rotated road bounding box detection tasks, which receives the same abstracted features for the drivable area from the one encoder. Therefore, the network size, computation time, and GPU usage can be reduced. It also decreases the possibility of overfitting by learning more generalized shared expressions to simultaneously fit multiple tasks.

The structure of the encoder and the segmentation decoder is the same as the Multinet [40]; however, the detection decoder is different. The encoder is based on the VGG16 network [41] which is widely used for training 2D data and shows high accuracy while ensuring real-time computation due to its simple structure. Each decoder and its output are described in detail below subsections. The sizes of the encoder layers are set according to the 200×200 input. This is shown on the left side of Figure 2.3. The encoders' weights are initialized using pre-trained weights from the ImageNet data [42]. Each decoder and its output are described in detail below subsections.

2.2.2.1 Drivable Area Segmentation (Occupancy Grid Map (OGM))

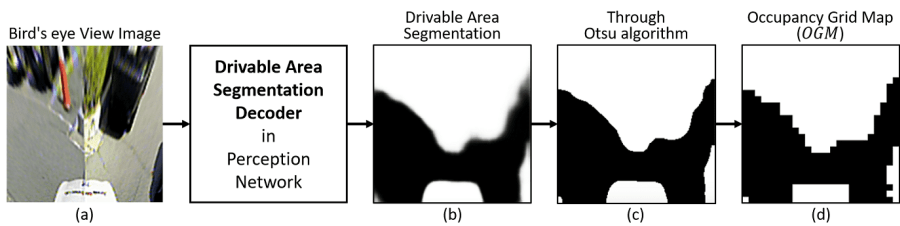


Figure 2.4: Process of drivable area segmentation to obtain occupancy grid map.

The structure of the segmentation decoder is shown in the upper left side of Figure 2.3. Features abstracted by the encoder have a low resolution 7×7 with a 1×1 convolution layer. These features are passed through a convolution layer

and up sampled by three transposed convolution layers [42, 43]. The size of the segmentation decoder layers is related to the encoder and is the network size for outputting 200×200 . In addition, each convolution layer of the encoder is combined with the decoder’s layers with skip connections [44] to extract high-resolution features from the lower layers (see Figure 2.3). The convolution layers are initialized using the scheme in [40] which performs bilinear upsampling to segment two classes: drivable/non-drivable.

The output of the segmentation decoder is a probability of the drivable at each pixel in the input image, which is shown in Figure 2.3(b). The closer the pixel is to black, the more likely it is drivable. The Otsu algorithm [45] is used to calculate the threshold value (Figure 2.3(c)). This algorithm divides the pixels into two classes by randomly setting a boundary value, and repeatedly obtains the intensity distribution of the two classes. Then, it selects the boundary value that makes the distribution of the values of the two classes most uniform. In other words, an optimal threshold value at which the ratio difference between binary-classified pixels can be smallest is obtained. The size of the segmented image is 200×200 , and it is converted to the occupancy grid map (*OGM*). This map is a 2D map that divides the image into 25×25 grids as shown in Figure 2.3(d). The grid is considered occupied even with only a single non-drivable pixel in the grid cell. That is, only drivable pixels exist in the unoccupied grid.

The data labeling criteria for training the drivable area segmentation image are in the image as follows: roads, road marks, a stop line, and crosswalk are labeled as the drivable area. The other area except the drivable area is designated as the non-drivable area. Sidewalks, parking spaces (including parking line), road boundary lines, pedestrian walkways, and vehicles are regarded as the non-drivable area.

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 sidewalks in the drivable area. 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 distinguished because the occupancy grid map is a 2D map (i.e., bird’s-eye-view). Thus, the vehicle can avoid nearby obstacles preferentially or consider distant obstacles in advance.

2.2.2.2 Rotated Road Bounding Box Detection

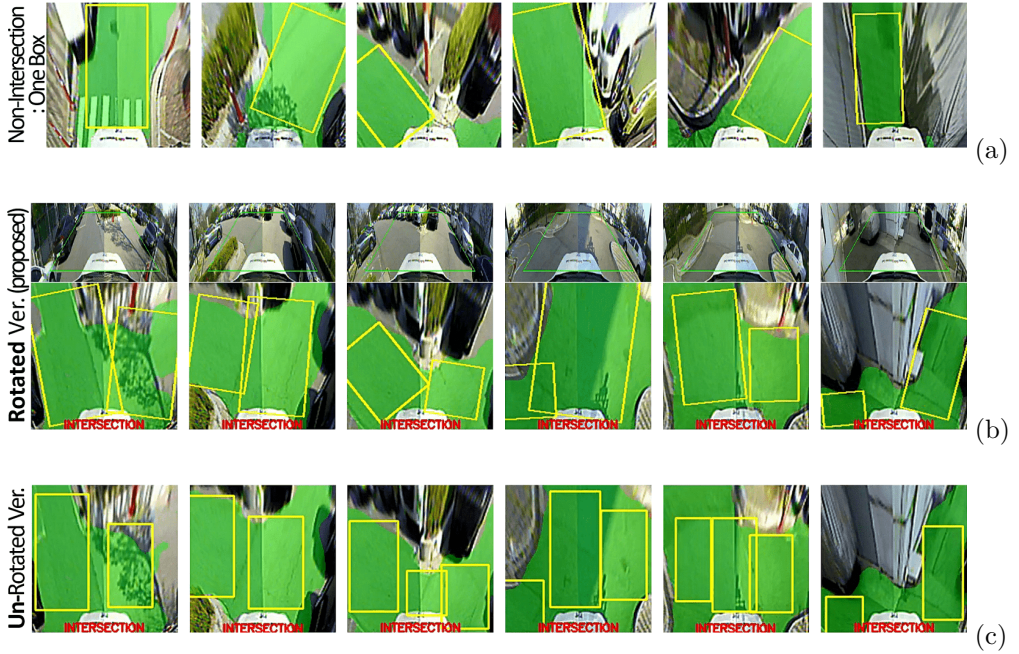


Figure 2.5: Rotated road bounding box detection results.

- (a) Non-intersection case, (b) Intersection case with rotated bounding box, and (c) Un-rotated bounding box

The rotated bounding box is used to detect and distinguish the branch roads. In general, the bounding box is recognized as an unrotated form in the

image frame. In this thesis, the rotated bounding box is used because, when a vehicle is located at an intersection, the direction of the road is differs from that of the vehicle. In this case, if the unrotated bounding box is used, it is not possible to cover all the drivable area without including obstacles inside the box (see Figure 2.5(c)). Moreover, an area that is not an actual branch road can be mistaken as a branch road (see the third (center box), fourth (right box), fifth (center box) images in Figure 2.5(c)). The rotated bounding box can detect one branch road as one box and cover the drivable area as much as possible (see Figure 2.5(b)), and the detection network can accurately find the box with a feature of the branch road.

A structure of the detection decoder is illustrated at the bottom right side of Figure 2.3 and is like the YOLO network [46]. The abstracted features, and the output of the encoder, pass a 1×1 convolutional layer with 50 filters. These passed features are divided into 7×7 grids g , and a $7 \times 7 \times 50$ shape tensor is obtained. Then, this tensor is passed through another tensor with $7 \times 7 \times 5$ shape to obtain a box's detail information, where 5 in $7 \times 7 \times 5$ means the number of the channels of the rotated bounding box, b ; the first to fourth channels represent the road bounding box coordinates. The x and y coordinates of the box's center, the width and height ratio of the box to the image. The fifth channel is a rotated angle value of the rotated box.

Each grid g is assigned a bounding box b . The origin of the grid and box coordinates is the upper right corner, and the box labels are parametrized based on a grid's position.

$$g_x = \frac{x_b - x_g}{w_g}, \quad g_y = \frac{y_b - y_g}{h_g}, \quad g_w = \frac{w_b}{w_g}, \quad g_h = \frac{h_b}{h_g}, \quad (2.1)$$

where x_g, y_g and x_b, y_b are the center coordinates of g and b ; w and h denote width and height, and w_g and h_g are the grid size; These units are pixels. The

loss value in each grid is calculated by the following equation.

$$\mathcal{L}(g, \hat{g}) = g_p(|g_x - \hat{g}_x| + |g_y - \hat{g}_y| + |g_w - \hat{g}_w| + |g_h - \hat{g}_h| + |g_a - \hat{g}_a|), \quad (2.2)$$

where g_p indicates whether a box exists in the grid, and if it exists, it is 1, otherwise 0. That is, a valid loss is reflected only when the box exists on the grid.; g is the ground-truth of the grid and \hat{g} its prediction value; g_a is the angle of the rotated box in radians. The loss per image is the average over the losses of all grids.

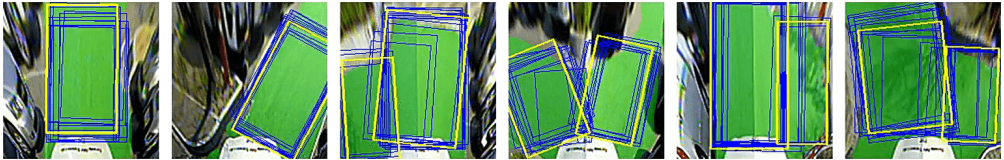


Figure 2.6: Non-maximum suppression algorithm.

Blue box: raw output of detection decoder (not grouped); Yellow box: finally detected (grouped) box through the blue boxes.

Through the $7 \times 7 \times 50$ shape tensor, 50 boxes are predicted, and boxes for the case where a confident value of the box is less than a threshold is filtered out (blue boxes in Figure 2.6). Here, the confidence value is calculated as the largest confidence value of the width and height channels. Through the non-maximum suppression algorithm [47], a box with the highest confident is selected among other sets of boxes where the overlapping area between boxes exceeds 50 % (yellow boxes in Figure 2.6).

The data labeling criteria for training the rotated road bounding box are as follows. 1) The drivable area between the front of the vehicle’s bonnet and the front end of the image is labeled as a rectangle. In this case, the width of the rectangle should not exceed the vehicle’s width. 2) Additional labeling is needed if there are unlabeled drivable areas on the side of the image. 3) The

maximum number of boxes is three. 4) The size of the box should exceed $7 m^2$. 5) Overlapping between two different boxes is possible but should be labeled to avoid overlapping area exceeding $5 m^2$. 6) When labeling the rotated bounding box, one or two corners of the box can be outside the image to label it as large as possible.

2.2.3 Intersection Decision

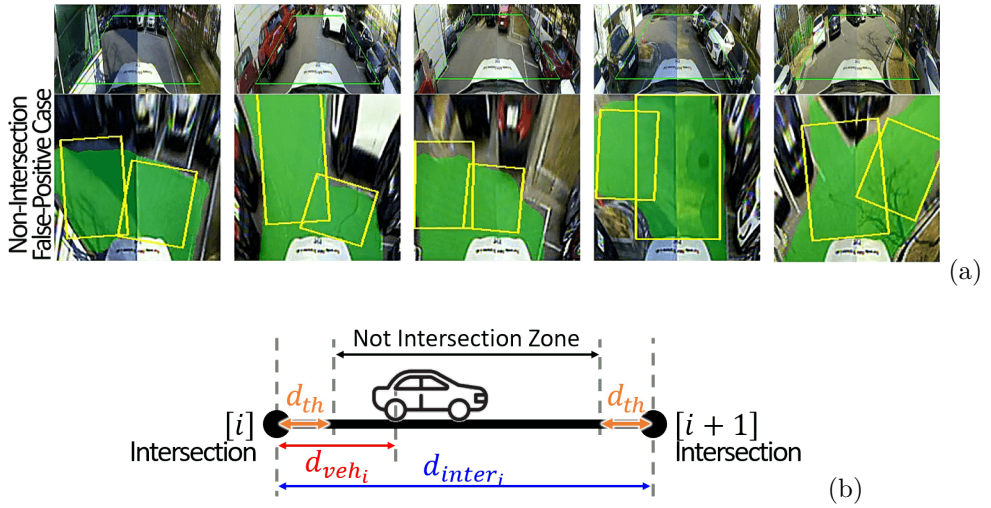


Figure 2.7: Intersection decision process.

(a) Two or more road boxes are detected in a non-intersection road: False positive case. (b) Intersection decision using a distance between intersections and the driving distance additionally.

When two or more boxes are recognized, it is regarded that the vehicle is driving at an intersection. As shown in Figure 2.7(a) and Figure 2.8(d), multiple boxes can be detected at a non-intersection road, which is a false positive case. To deal with this, a road distance between the center of intersections d_{inter_i} and an accumulated distance d_{veh_i} are defined. d_{inter_i} is defined as an edge distance (road) between two nodes (intersection) of the topological map,

which is between the i 'th and $i+1$ 'th intersections. Here, i is an index according to the order of the intersection, and a case of $i = 0$ indicates the starting point (not an intersection). The order of visiting the intersection on this map is determined according to the navigation information. d_{veh_i} is the accumulated distance moved by a vehicle from the i 'th intersection, and is calculated using the vehicle's velocity.

If two or more boxes are recognized, and a difference between d_{inter_i} and d_{veh_i} is lower than d_{th} , it is determined that the vehicle is driving at an intersection:

$$Intersection\ Flag = \begin{cases} True & |d_{inter_i} - d_{veh_i}| \leq d_{th} \ \& \ \text{'Number of box} \geq 2' \\ False & |d_{inter_i} - d_{veh_i}| \leq d_{th} \ \& \ \text{'Number of box} < 2' \\ False & |d_{inter_i} - d_{veh_i}| > d_{th}. \end{cases} \quad (2.3)$$

Thus, if the vehicle is located at a distance of d_{th} between the intersection i 'th and $i+1$ 'th as shown by the black arrow in Figure 2.7(b), it is regarded as not passing the intersection, even if more than two boxes are recognized. i is increased when *Intersection Flag* in (2.3) changes from *True* to *False* and $|d_{inter_i} - d_{veh_i}|$ is larger than d_{th} together.

2.2.3.1 Road Occupancy Grid Map (OGM_{road})

When it is determined that the vehicle is passing the intersection, one box is selected according to the navigation information, such as *Going Straight*, *Turning Left*, and *Turning Right*. This information is obtained through a global plan to visit all the roads in the topological map [37]. This process is shown in Figure 2.1, and Figure 2.8. The criteria for selecting one box are as follows.

- *Going Straight*: the box closest to the middle of the image.
- *Turning Left*: the leftmost box in the image.
- *Turning Right*: the rightmost box in the image.

Through the selected box, a road occupancy grid map (OGM_{road}) is obtained, as shown in Figure 2.8. An inside region of the selected box is regarded as the drivable area, whereas its outside is considered as the non-drivable area. The image segmented according to the selected box is converted into OGM . If only one box is recognized, it is determined that the vehicle is not driving at the intersection, so all area of OGM_{road} is regarded as drivable area. This is shown in Figures 2.8(c) and 2.8(d).

2.2.4 Merged Occupancy Grid Map (OGM_{mer})

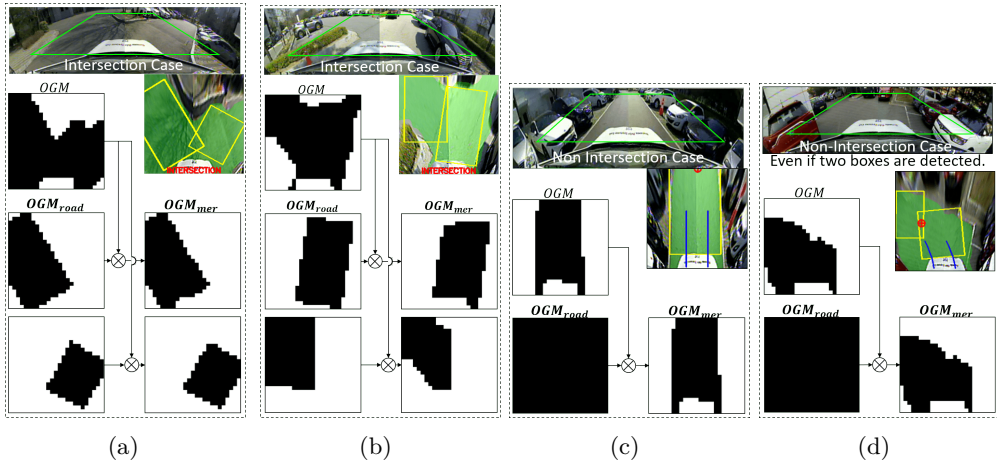


Figure 2.8: Processes of obtaining merged occupancy grid map (OGM_{mer}).

To navigate intersections, using existing motion-planning algorithms that do not take the intersection navigation into account, a merged occupancy grid

map (OGM_{mer}) can be used as an input for these algorithms (see Figure 2.8). At intersections, several branch roads exist in the drivable area in OGM , and these algorithms cannot calculate the action of which branch drivable area to head to. To address this problem, the drivable area of OGM_{road} consists of the drivable area of OGM_{mer} . In addition, to consider obstacles existing inside OGM_{road} , only the common drivable between OGM_{road} and OGM is drivable in OGM_{mer} .

$$\text{Drivable Area of } OGM_{mer} = \text{Drivable Area of } (OGM_{road} \cap OGM). \quad (2.4)$$

The drivable area on each map is defined as the “true” value (the black area (grid) in Figure 2.8), and the non-drivable area as the “false” value (the white area (grid) in Figure 2.8). Each grid in OGM_{mer} is calculated by the ampersand operator (&) between OGM and OGM_{road} , and becomes a true value only when the grids of both maps are the “true” value. Thus, at an intersection, in OGM_{mer} , only one branch road among several branch roads is the drivable area as if it were not the intersection. At non-intersection roads, OGM_{mer}

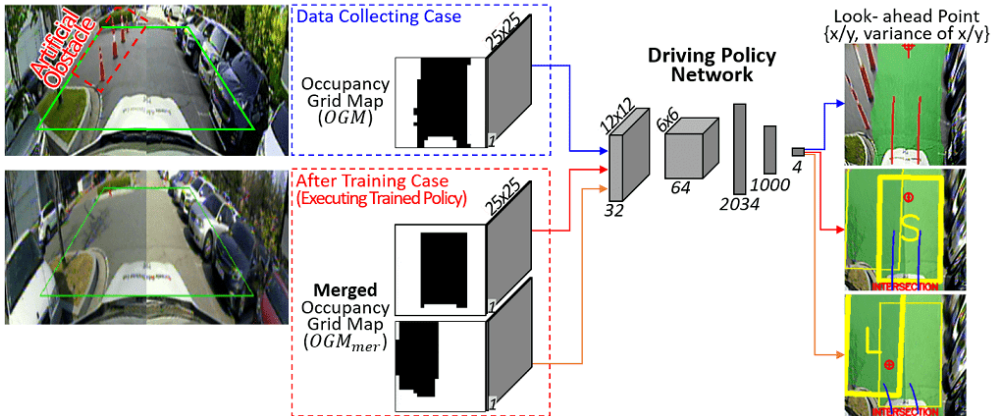


Figure 2.9: Driving Policy Network.

equivalent to OGM , since all grids of OGM_{road} is regarded as the drivable area (see Figures 2.8(c) and 2.8(d)). Therefore, by using OGM_{mer} , the motion-planning algorithm can calculate the vehicle’s action as it heads to the drivable area of one branch road while avoiding obstacles.

In Chapter 3, a method of driving a vehicle toward the drivable area using imitation learning is proposed. The driving policy is trained to imitate an expert’s driving towards the drivable area while avoiding obstacles in real-time in a non-intersection environment (see the upper side of Figure 2.9). As shown in the lower side of Figure 2.9, by using OGM_{mer} , the driving policy can acquire a look-ahead point at which the vehicle heads to the drivable area corresponding to only one branch road.

2.3 Experiment

2.3.1 Experimental Setup

2.3.1.1 Autonomous Vehicle

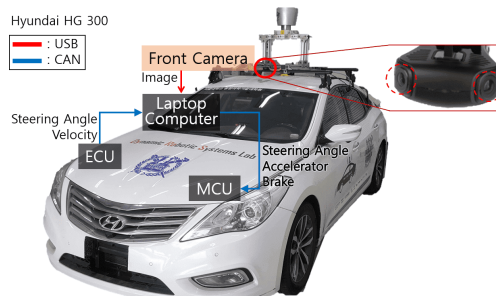


Figure 2.10: Autonomous vehicle.

As shown in Figure 2.10, the vehicle used in the experiments was a Hyundai HG 240. 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.

Pure pursuit algorithm [48] was used to calculate the steering angle command (δ) 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. θ_l is the look-ahead heading, which is the difference between the heading of the vehicle and the heading of the vector from the vehicle to the look-ahead point. The range of δ was -540° to 540° .

The velocity command v used to reach the look-ahead point was proportional to a_y which is the longitudinal distance between this point and the vehicle. Thus, $v = \frac{a_y}{2.24}$, where the final v was set to half of a_y for safety reasons. The range of v was $0.5 \sim 2.2$ (desired velocity) m/s. The accelerator and brake commands for controlling the velocity were calculated using the PI controller.

A front camera was attached 1.55 m above the ground and 0.25 m away from the center of the vehicle. Moreover, it was rotated about 20° downward for the ground direction. This is to minimize the shaded area in the conversion of the image to the bird's eye view image. This camera consists of two lenses to widely recognize the environment, and the two images obtained are concatenated. The field of view of each lens is 120° so that the branch roads can be sufficiently recognized. Distortion of the image is corrected by obtaining the focal length, principal point, and distortion coefficients.

2.3.1.2 Multi-task Network Setup

A method for training a multi-task-based perception network followed a classic fine-tuning pipeline [40]. The weights of each decoder were calculated and individually updated with the different loss values. To update the weights of the encoder, two-loss values were added with different weightings (segmentation: 25%; detection: 75%).

$$\mathcal{L}_{enc} = (\alpha\mathcal{L}_{seg} + (1 - \alpha)\mathcal{L}_{box})/2, \quad (2.5)$$

where \mathcal{L}_{enc} is multi-task loss function of the encoder; \mathcal{L}_{seg} and \mathcal{L}_{box} are losses for the drivable area segmentation and the rotated bounding box detection; α is a ratio that adjusts the training importance between \mathcal{L}_{seg} and \mathcal{L}_{box} , and is set to 0.25. The weights of each decoder were calculated and individually updated with \mathcal{L}_{seg} and \mathcal{L}_{box} . Through this process, different data, and training hyper parameters could be applied to each decoder.

The loss function of the drivable area segmentation (\mathcal{L}_{seg}) decoder was Soft-Max cross-entropy. It infers the drivable and non-drivable probability values for each pixel. The average value of all pixels becomes a loss value of segmentation. The road rotated bounding detection (\mathcal{L}_{box}) decoder was trained with the L1 (regression) losses for each five-channel value in the grid of 7×7 cells. The five channels are values for the box’s position (x/y), width, height, and angle. These values are summed with equal weight. If the grid is non-drivable, the sum of the detection loss becomes zero. Additionally, the confidence values of the width and height channels are trained with the cross-entropy loss function.

The Adam optimizer with a 10^{-5} learning rate was used to train the perception network. The weight decay of $5 \cdot 10^{-4}$ was applied to all layers, and a dropout with 50 % probability was applied to all 1×1 convolution layers of

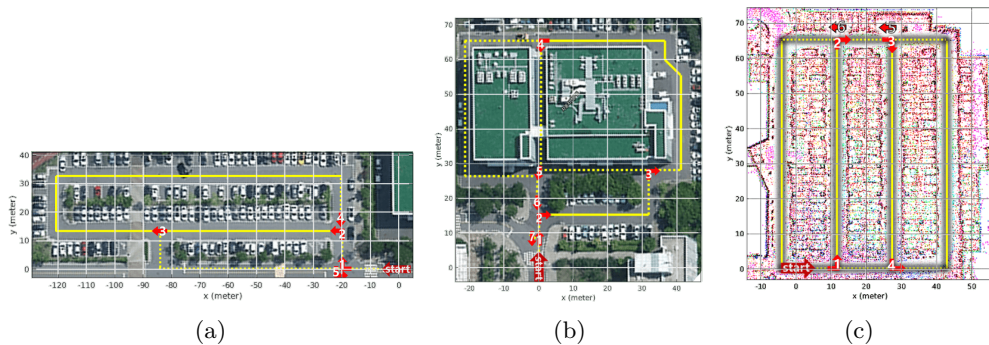


Figure 2.11: Parking lot environments for collecting perception data.

(a) and (b) outdoor parking lot, (c) indoor parking lot; the map image was built by LiDAR-based SLAM [49]. A topological map was built to obtain the navigation information. Through an algorithm [37], an order of visiting the intersection was calculated, and the navigation information was obtained by using the calculated order, which is shown in the red arrows and numbers.

the detection decoder. Epochs were 10k, and the batch size was set to 128. Training time was about 3 hours when using a laptop. Weights were assigned before training to initialize the network for efficient training. The encoder was initialized with weights trained on ImageNet data.

The dataset was collected in three parking lots having 18 intersections (see Figure 2.11). One bird’s-eye-view image per second was collected while driving, and 1,069 images were collected. In each task, 80 % of the dataset was used for training, and the remaining portion of the dataset was used for validation. The entire image was used to segment the drivable area by using a pixel annotation tool [50].

To label the rotated bounding box data, Bbox label tool [51] was used. Among the collected images, half of the non-intersection data were excluded, and the road bounding box was trained using 772 images. The reason not to use the entire collected data is to balance the ratio between the non-intersection

and intersection data by excluding a similar situation from the non-intersection data. Therefore, the detection accuracy could be higher at the intersection than when using the entire data.

2.3.1.3 Model-based Branch Road Detection Method

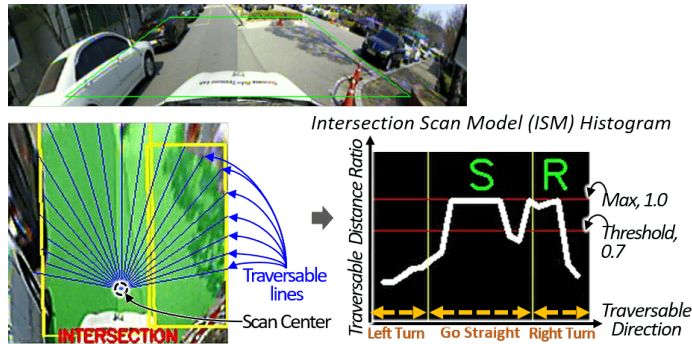


Figure 2.12: Results for *intersection scan model (ISM)* method.

The *intersection scan model (ISM)* method [32] is used to compare the performance of detecting branch roads with the proposed method. The *ISM* method recognizes branch roads at the intersection using the segmented image, without the global information. *ISM* [32] defines 21 traversable lines from a scan center point to the end of the image according to a traversable direction $[13^\circ, 21^\circ, \dots, 165^\circ, \text{ and } 173^\circ]$. The traversable directions are divided into the left turn (6 traversable lines), straight (9 traversable lines), and right turn (6 traversable lines) on the horizontal axis of the histogram.

If there are obstacles on the traversable line, an obstacle distance from the scan center point to the closest obstacle on the traversable line is calculated (blue lines in Figure 2.12). The traversable distance ratio of the traversable line's length to the obstacle distance is obtained. These ratios according to the traversable direction are used to obtain an *ISM* histogram (see the red

dashed box in Figure 2.12). If more than half of the traversable distance ratio exceeding the threshold (0.7) exists in the traversable direction section, it can be determined that the branch road exists. Here, the distance ratio is the vertical axis of the *ISM* histogram.

2.3.2 Experimental Results

2.3.2.1 Quantitative Analysis of Multi-Task Network

The performance of the multi-task network was tested with the validation dataset that was not used to train the network. The performances of the drivable area segmentation and rotated road bounding box detection tasks were measured by different metrics.

The pixel accuracy metric was used to evaluate the performance of the drivable area segmentation:

$$Pixel\ Accuracy = \frac{number\ of\ correctly\ classified\ pixels}{number\ of\ total\ pixels}, \quad (2.6)$$

where the numerator is the number of cases matching the output of the network and the label in the data, in each pixel. The denominator which was 200×200 , is the number of pixels in the bird’s-eye-view image.

Detecting result of the rotated road bounding boxes was used over the union, the intersection over union (IoU) metric:

$$IoU = \frac{area\ of\ overlap}{area\ of\ union}, \quad (2.7)$$

where the numerator represents the size of the overlap between the label box and the prediction box. The denominator represents the region for the union of

the label box and the prediction box. High IoU indicates that the performance of the proposed network is high. Table 2.1 shows the performance of the network for each task in Figure 2.11 parking lots using the validation images.

Table 2.1: Results of the Perception Network

		Parking Lots		
		Figure 2.11(a)	Figure 2.11(b)	Figure 2.11(c)
Drivable Area Segmentation (Pixel Accuracy, (2.6))		95.81 %	94.15 %	93.73 %
Branch Road Detection (IoU, (2.7))	Rotated Box (Proposed)	98.58 %	97.31 %	97.10 %
	Un-rotated Box	97.73 %	95.17 %	94.79 %

As shown in Figure 2.13, the resulting images of the segmentation task are indicated as the green area in the bird’s-eye-view image or the black area in *OGM*. The detection task result is indicated by the yellow squares. The inference speed of the perception network was an average of 23.7 fps.

2.3.2.2 Comparison of Branch Road Detection Method

To evaluate the intersection recognition performance, the proposed rotated road bounding box detection algorithm and *ISM* [32] algorithm were tested on the same image. A branch road detection accuracy was defined as follows:

$$Intersection\ Accuracy = \frac{N_{match}}{N_{inter}}, \quad (2.8)$$

where N_{match} represents the number of matched cases between a label branch box for the validation dataset and an output branch box by each algorithm. N_{inter} represents the number of images containing the intersection. For example, if there were turning left and straight boxes in the data, and the output of the proposed algorithm was the turning left and straight boxes, it is regarded as the correctly recognized case. In the same situation, if *ISM* recognizes only the

straight navigation information, it is not considered as a correctly recognized case. The results are shown in Table 2.2.

Table 2.2: Intersection Detection Results

	Rotated Box (Proposed)	Un-rotated Box	<i>ISM</i> Algorithm [32]
Intersection Accuracy (2.8)	97.2%	95.4%	82.7%

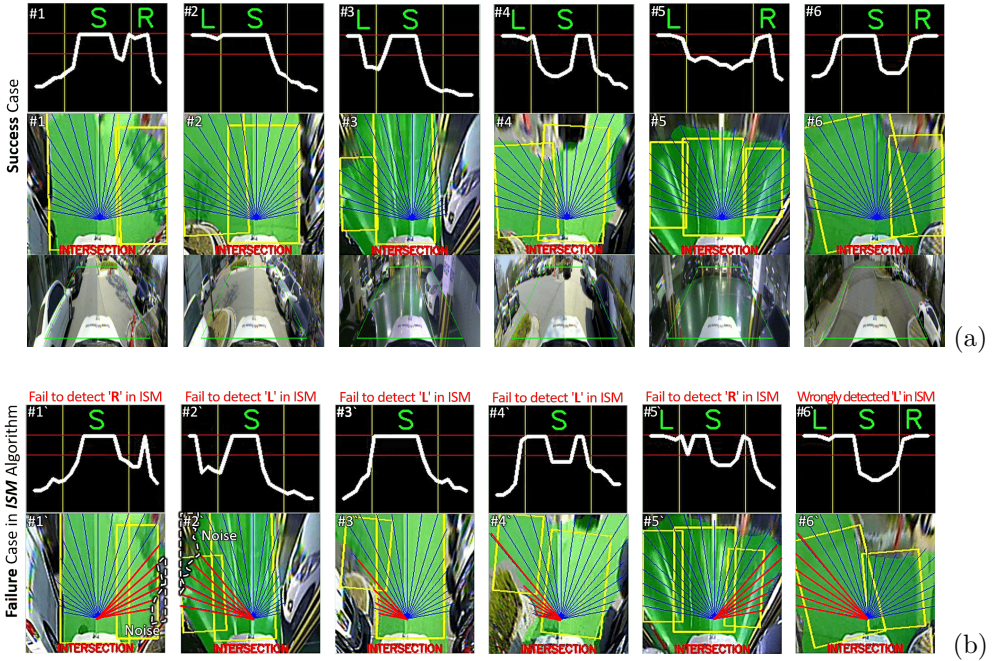


Figure 2.13: Branch roads detection results at Intersection.

(a) and (b) are results of different *ISM* performance at different spots at the same intersection; (a) Success case, (b) Failure case in *ISM* algorithm [32]

‘#n’ in Figure 2.13(a) and ‘#n’ in Figure 2.13(b) are results in scenes for slightly different spots at the same intersection. Figure 2.13(a) represents situations where two algorithms correctly detected the branch roads. Figure 2.13(b) shows that the proposed algorithm was recognized accurately; however *ISM* did not. There were cases in which a specific branch road at an intersection was not recognized. #1’ and #2’ in Figure 2.13(b) are examples where some drivable

area was recognized as non-drivable (perception noise), so the traversable distance ratio in *ISM* was calculated shorter (indicated as the red lines in #1' and #2' of Figure 2.13(b)) than a real ground-truth traversable distance ratio. Even if the inside of the rotated bounding box is recognized as not all drivable, the proposed detection algorithm had high accuracy, because it inferred the overall shape and distribution of the drivable area inside the box.

Cases #3', #4', and #5' of Figure 2.13(b) are cases where the branch road was not detected in *ISM* since the area behind the obstacle was recognized as the non-drivable area due to the shadow of the camera. The traversable distance ratios of the left (#3' and #4') and right (#5') turn sections were short, and each branch road was not detected. In #6', the straight branch road was too wide, and the drivable distance ratio of the left turn was recognized as too long, and was mistaken as the left branch road. Situations #3', #4', #5', and #6' may be addressed by properly tuning the model parameters of *ISM*, although it is unclear whether these parameters can be applied to other situations. However, the proposed learning-based method was able to accurately find the branch roads without finding the model parameters.

The computation time of the proposed multi-task network was 27.3 fps and *ISM* showed 29.5 fps. Although, *ISM* showed a higher calculation speed, the difference between the two methods are not much different. These methods are faster than the driving control period, 20 fps.

The driving results using the output of the proposed multi-task perception network for motion-planning algorithm are shown in Section 3.5.2.5.

2.4 Conclusion

This chapter proposes a method that detects branch roads at an intersection using vision and deep learning, which can be used alongside the existing motion-planning algorithm for navigating in a semi-structured environment. The proposed multi-task network distinguishes the branch roads at an intersection as the rotated bounding box. At the intersection, the inner area of a box selected through the navigation information is regarded as the drivable area. Furthermore, this network segments the drivable area so that obstacles existing inside the box can be recognized.

The proposed method was tested in three parking lot with 18 intersections. It detected the branch roads more robustly than the model-based method using the distance and direction histogram of the branch road in the cases where branch roads varied in size and shape, or the drivable area was detected noisy. In addition, the vehicle successfully navigated the intersection by applying the proposed perception method to the existing motion-planning algorithms such as the tentacle, field, imitation learning algorithms without using global information. In the future, the experiment will be conducted in more diverse environments.

Chapter 3

DATA AGGREGATION (DAGGER) ALGORITHM WITH LOOK-AHEAD POINT FOR AUTONOMOUS DRIVING IN SEMI-STRUCTURED ENVIRON- MENT

3.1 Introduction

Autonomous driving technology for semi-structured environments such as parking lots and alleyways is important for fully autonomous driving. Moreover, it is more difficult than driving in structured environments. In a structured environment, autonomous driving involves a global plan using a road network, and a vehicle keeps within a lane via lateral control and maintains a safe distance from vehicles in front while following a target speed through longitudinal control. Furthermore, in a semi-structured environment, the curvature can rapidly change, such as at right-angled corners, and the drivable area 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. Typically,

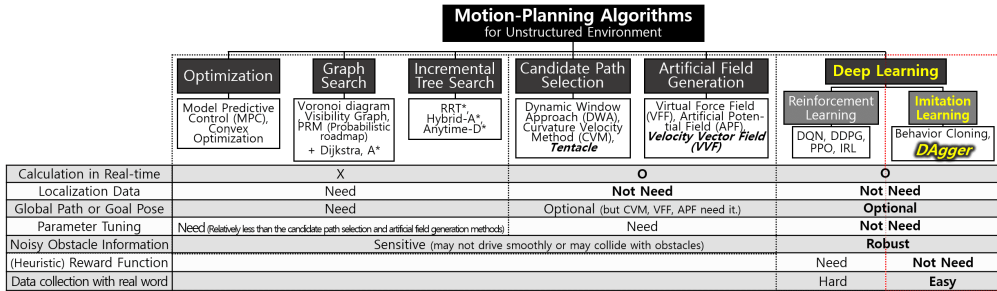


Figure 3.1: Motion-planning algorithms and comparison between them.

such obstacles are unknown in advance. Navigating such a situation is difficult even in a static environment, and existing motion-planning algorithms are unable to handle such settings.

A representative approach for driving in an semi-structured environment is to generate a global map on a global path to reach the destination. The vehicle tracks the path using localization data (i.e., the position and heading of the vehicle relative to the path) [52]. While tracking the global path, the vehicle checks for obstacles in its path. Object detection algorithms detect the position and shape of obstacles using camera or LiDAR sensors. If obstacles are detected near the global path, motion-planning is used to search for a local path or waypoint that can reach the global path without collision. The planned solution must also satisfy kinematic constraints on the motion of the vehicle. These motion-planning algorithms developed for robotics have been used in autonomous vehicles [53]. These can be categorized according to the method and calculation time. An overview of motion-planning algorithms is shown in Figure 3.1, which includes optimization, graph search, and incremental tree search planning methods to find a solution for the local area.

The path planning method using optimization theory, such as model predictive control (MPC) [54] and convex optimization [55], uses a vehicle’s kinematic

or dynamic model 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 the 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 [56], Visibility graph [57], and Probabilistic roadmap (PRM) [58] 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. The graph is used to search for the minimum path length using the Dijkstra or A* graph search algorithm. The searched path is interpolated via spline algorithms to satisfy vehicle constraints and obtain a smooth path.

An incremental search path planning method uses tree exploration algorithms, which iteratively expand a tree into free space at the end of the tree to reach a goal. Rapidly exploring random trees* (RRT*) algorithm [59] extends the tree with samples randomly selected in the configuration space. The hybrid-A* [60] and anytime-D* [61] algorithms expand the tree in grid units. The path with the minimum length is searched for to reach the goal pose while satisfying the non-holonomic constraints of the vehicle. The non-holonomic constraint refers to a motion that cannot move directly sideways, so a vehicle must drive forward or backward to rotate.

However, these methods have three problems [62]. First, if the local area is large or complex, a long computational time is required to generate the path, and the solution may not be found within a control loop (i.e., not real-time). Second, selecting a goal pose in the global path to search for the local path is heuristic. Third, when an algorithm is implemented, accurately recog-

nizing whether an obstacle is close to the global path and tracking the path without collision is difficult because of inaccurate localization data. In semi-structured environments, various types of obstacles are complexly placed in the drivable area. Thus, obtaining accurate localization data at every point in semi-structured environments is difficult.

Rather than searching and tracking a path, alternative methods can be used that allow the vehicle to drive toward the global path while reactively avoiding obstacles. The candidate path selection and artificial field generation methods find a solution close to a vehicle that can be calculated in real-time. They select a candidate path or waypoint and calculate the control commands.

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 the ride comfortable. Three algorithms have been used to achieve these: the dynamic window approach (DWA) [63], the curvature velocity method (CVM) [64], and tentacle [65] 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 it additionally considers vehicle accelerations. 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 an attractive field toward the global path. These fields are combined with different weights, and a vehicle is guided by the combined field's vector. There are three algorithms available that differ in how they model the fields: namely, the virtual force field (VFF) [66], the artificial potential field (APF) [67], and

the velocity vector field (VVF) [68] 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. 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 limitations that make them difficult to be used in semi-structured environments. First, the parameters in the objective function or field model may differ to cope with the various complex situations of such environments. It is difficult to identify 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. 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 [69]. Moreover, a vehicle may drive out of the drivable area or toward an obstacle. To address these limitations, this chapter proposes a deep learning-based method for selecting the look-ahead point to drive toward the drivable area while avoiding obstacles in real-time without the use of global information such as the global map and localization data. This is shown in Figure 3.3. Vision data is segmented into the drivable area and non-drivable area using deep learning, and this is represented as an occupancy grid map; it does not recognize whether obstacles exist close to the global path and can ignore irrelevant information for driving, improving the generality of driving policy in untrained environments.

The proposed method uses imitation learning as the motion-planning algorithm, which is an alternative to general ones. Imitation learning obtains a

safe driving policy by collecting expert driving data for various complicated situations that occur in semi-structured environments such as large changes in the curvature and width of the drivable area. Thus, it is not necessary to manually model the policy and tune parameters heuristically to handle such situations. The data include cases where the occupancy grid map is incorrectly recognized correctly or is noisy because of shadows, ensuring that the driving policy is robust in these situations. Furthermore, imitation learning obtains the policy faster than reinforcement learning in real environments because it does not require trial-and-error and heuristic reward function modeling.

The proposed imitation learning method trains the driving policy to select the look-ahead point on the occupancy grid map [70]. The look-ahead point is a target waypoint for a vehicle to reach, which is calculated from the pure pursuit algorithm [48] that is commonly used in autonomous driving. The velocity is calculated according to the longitudinal distance between the look-ahead point and vehicle. There are several advantages to using the look-ahead point. First, selecting the look-ahead point that can avoid obstacles while driving fast on the occupancy grid map has a clearer pattern relationship than a front-view image and steering-velocity relationship which is common in imitation learning. The driving policy can properly train driving patterns and is safer. Second, the trained driving policy and expert behavior can be shared, allowing the data aggregation (DAgger) algorithm [71] to be applied to the autonomous vehicle, which enhances the imitation learning performance

The rest of this chapter is organized as follows. Sec. 3.2 introduces DAgger algorithms for autonomous vehicle and background of the DAgger algorithm. Sec. 3.3 explains the proposed DAgger algorithm with the look-ahead point. The experimental setup and results are presented in Sec. 3.4 and Sec. 3.5. Finally, this chapter is concluded in Sec. 3.6.

3.2 Related Works & Background

This section introduces related works applying the DAgger algorithm of imitation learning to the autonomous driving. In addition, the behavior cloning algorithm, which is the base of imitation learning, and DAgger is explained.

3.2.1 DAgger Algorithms for Autonomous Driving

Method	State	Action	Algorithm	Expert Model	Experiment
Bojarski et al. (2017)	Image	Steering	Behavior Cloning	Human	Simulation & Real-world (Normal Env., 2% Falty)
Zhang and Cho (2017)	Image	Steering + Brake	(Safe) DAgger	Rule-based Controller by Modeling	Simulation
Bicer et al. (2019)	Image	Steering + acc. + (driving situation) Class	(Selective Safe) DAgger	Rule-based Controller by Modeling	Simulation
Liet et al. (2019)	Images	Steering	DAgger	Rule-based Controller from Simulation	Simulation
Aditya, et al (2020)	Image	Steering + acc.	DAgger (with Replay Buffer)	Rule-based Controller from Simulation	Simulation
Kelly et al. (2019)	Image	Steering + acc.	(Human-gated) DAgger	Human	Real-world (Joystick required, straight road, 300 m)
Pan et al. (2020)	Image + speed	Steering + acc.	DAgger	Warm-start (by Human) MPC	Simulation & Real-world (Track, mini car)
Proposed Method	Occupancy Grid Map	Look-ahead point	DAgger	Human	Real-world (4 Parking lot, 3453 m)

Figure 3.2: Behavior cloning and DAgger algorithms for autonomous driving.

Figure 3.2 shows studies applying imitation learning to autonomous driving, and their state, action, expert model, and experimental configurations. All studies use the image obtained from the front camera of the vehicle as the state, and the action is composed of the steering angle, and velocity or acceleration. The study proposed by M Bojarski *et al.* [72] uses the behavior cloning algorithm, and experimented in simulation and real environments, but 2 % of driving resulted in a situation in which human intervention is required in a simple driving situation.

Based on the DAgger algorithm, [73] proposes a SafeDAgger algorithm that collects data when there is a large difference between expert and trained network actions in the training process. A Selective-SafeDAgger algorithm classifies driving situations such as sharp curves, selects data and collects more [74]. These algorithms use rule-based controller as an expert model, and are tested only in simulations. A study [75] introduces an optimized DAgger performance method in simulation. In [76], a method for training by efficiently sampling from DAgger additional data is proposed, and similarly validated in simulations.

A HG-DAgger algorithm [77] proposes a method for humans to perform DAgger as an expert model, which is applied to a simple real-world experiment. A study [78] proposes applying the MPC algorithm to DAgger data labeling, and it is tested in a real environment but not to an actual vehicle. Unlike the aforementioned studies, the proposed method has the advantage that a human consists as an expert model. Compared to using rule-based controllers, forward simulations, and MPCs, consisting the expert model human is easier to apply to various and complex situations and enables real-time calculations. In addition, the proposed method can be tested with the real vehicle, and the reason is explained in detail in Section 3.3.4.

3.2.2 Behavior Cloning

Imitation learning involves mimicking a behavior of an expert in certain states. While an expert in driving, state action pairs of data are collected. The driving policy π_{net} (i.e., deep neural network) is trained with the data through a process known as *behavior cloning*, which is a single training step in imitation learning.

behavior cloning is performed to train a function approximator $\pi_{net=BC}$ that can imitate an expert’s behavior. $\pi_{BC}(s_t; \theta)$ is parameterized by θ , and it

produces an action value a given state s . The action value a is the behavior of the expert for state s . The process of training $\pi_{BC}(s_t; \theta)$ is similar to that of supervised learning. To train the function approximator $\pi_{BC}(s_t; \theta)$, the parameter θ is optimized to minimize the loss function \mathcal{L} . The loss function \mathcal{L} is the difference between $\pi_{BC}(s_t; \theta)$ and the action value a , which is expressed as \mathcal{L}_{Gau_t} for s_t . A large number N of state action data pairs $D = \{(s_t, a_{exp,t})\}_{t=1}^N$ is used to optimize θ .

However, if $\pi_{net=BC}$ encounters states that are not similar to dataset D or are noisy, π_{BC} may produce unsafe or unsafe actions. As shown in Figure 3.5(b), a noisy state is when the boundary of the drivable area or shadow area is not accurately recognized. Furthermore, the location and type of obstacles differ when the dataset for π_{BC} is collected and executed. Here, the vehicle cannot sufficiently avoid obstacles; this is known as the *data mismatch problem*, which occurs when the data for unsafe or near-collision situations are included in D less. Thus, the policy π_{net} cannot reflect these situations in π_{BC} well; this is known as the *data imbalance problem*. Moreover, when these problems occur in a driving situation, the error may magnify afterward because π_{BC} has not learned recovery behavior; that is known as the *compounding error problem*.

3.2.3 DAgger Algorithm

To address the limitations of *behavior cloning*, *DAgger* [71] is used to collect additional data by executing the trained *behavior cloning* policy and retraining π_{net} . This process is repeated until the best policy is obtained.

Algorithm 1 represents the basic structure of *DAgger*. First, *DAgger* initializes the policy $\pi_{net,i=1}$ and dataset D as those obtained from *behavior cloning*. The DAgger iteration i and $\hat{\eta}_i$ representing the performance of the trained pol-

Algorithm 1: Pseudo-code of *DAgger* Algorithm

*Note The **blue** font is related to *WeightDAgger* introduced in Chapter 4.

```
1 function DAgger( $\pi_{BC}, D_{BC}$ )
2 Initialize  $\pi_{net,1} \leftarrow \pi_{BC}$ 
3 Initialize  $D \leftarrow D_{BC}$ 
4 Initialize  $i \leftarrow 1, \hat{\eta}_i \leftarrow 0.0$ 
5 while  $\hat{\eta}_i \leq \eta$  do // Do Until Obtaining Desired Policy
6    $D_i, \hat{\eta}_i \leftarrow$  Sample unsafe or near-collision data using
   Data-Sampling Function( $\pi_{net,i}$ )
7   Weight_Update( $D, D_i$ ) using Algorithm 4 // for WeightDAgger
   (Chapter 4)
8   Aggregate dataset  $D \leftarrow D \cup D_i$ 
9   Train policy  $\pi_{net,i+1}$  on  $D$  using Eq. 3.3
10   $i += 1$ 
11 return  $\pi_{net,i}$ 
```

icy $\pi_{net,i}$ are initialized. When the iteration is started ($i = 1$), the additional dataset D_i is collected by the *data-sampling function* as described in the next chapter, which samples the only data for unsafe or near-collision situations (line 6 in Algorithm 1). Following the driving via the *data-sampling function*, the additional dataset D_i collected is aggregated to the existing dataset D (line 8). The aggregated dataset D is used to retrain the policy π_{net} with (3.3) (line 9). After training, a policy $\pi_{net,i+1}$ that causes fewer unsafe or near-collision situations than $\pi_{net,i}$ can be obtained. This process is repeated until the desired policy is obtained.

As more data from these problem situations are aggregated, $\pi_{net,i}$ becomes more capable of dealing with the situations, which is proven in [71]. *DAgger* repeats this process until the problem situation rarely happens (Algorithm 1 in line 5). This can be judged by $\hat{\eta}_i$ which is the ratio of executed network actions among the total executed actions. If $\hat{\eta}_i$ is greater than the threshold η , the iterations of *DAgger* are terminated. Finally, a policy $\pi_{net,i}$ that does not cause unsafe or near-collision situations is obtained (line 12).

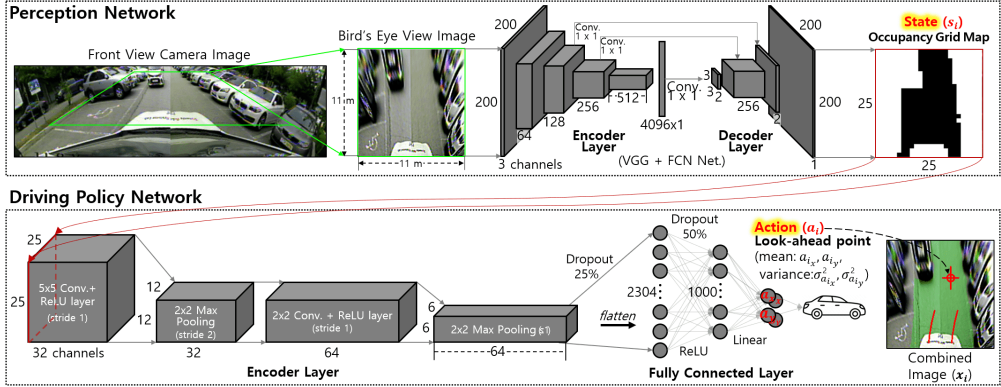


Figure 3.3: System architecture of the proposed driving method.

3.3 Proposed Method

3.3.1 DAGger with Look-ahead Point Composition (State & Action)

The dataset comprises 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 (25×25 grid $\in \{\mathbf{0}$ (black): drivable(unoccupied), $\mathbf{1}$ (white): non-drivable(occupied) $\}$), which is used for the input of the driving policy π_{net} .

The action a_t is a command of an expert and the output of π_{net} . In this thesis, the look-ahead point is used as the action $a_t \in \{a_{t_x}, a_{t_y}\}$, 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 in executing the DAGger algorithm. This reason is explained in detail in Section 3.3.4. The output of the policy for a state is expressed as follows:

$$a_{net,t} = \pi_{net}(s_t), \quad (3.1)$$

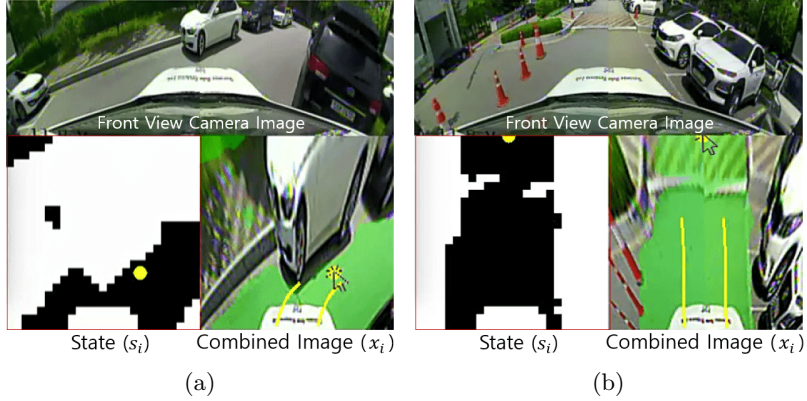


Figure 3.4: Dataset collection process of imitation learning (behavior cloning). The yellow look-ahead point is the action $a_{exp,t}$ selected by an expert. The expert selects $a_{exp,t}$ in the combined image x_t . The yellow lines are the future trajectory that the vehicle will drive towards $a_{exp,t}$ along during a certain time. The state s_t is the occupancy grid map. The white area of the grid represents obstacles, and the black area represents the drivable area.

where $a_{net,t} \in \{\bar{a}_{net,t_x}, \bar{a}_{net,t_y}, \sigma_{a_{net,t_x}}^2, \sigma_{a_{net,t_y}}^2\}$ are the mean and variance of the look-ahead point. The variance of the look-ahead point is calculated using the Gaussian process (GP) to quantify the uncertainty or confidence of π_{net} [79].

To collect training data, the expert selects the look-ahead point $a_{exp,t} \in \{a_{exp,t_x}, a_{exp,t_y}\}$, and the vehicle is controlled in real-time to reach the selected look-ahead point. The pure pursuit algorithm [48] is used to calculate the steering angle command. The velocity command is proportional to the distance between this point and the vehicle. The dataset $D = \{(s_t, a_{exp,t})\}_t$ is stored for every period t as the vehicle is moving, and numerous data can be easily collected. This process is repeated continuously until the driving is completed.

The expert selects the look-ahead point $a_{exp,t}$ using a mouse pointer in the

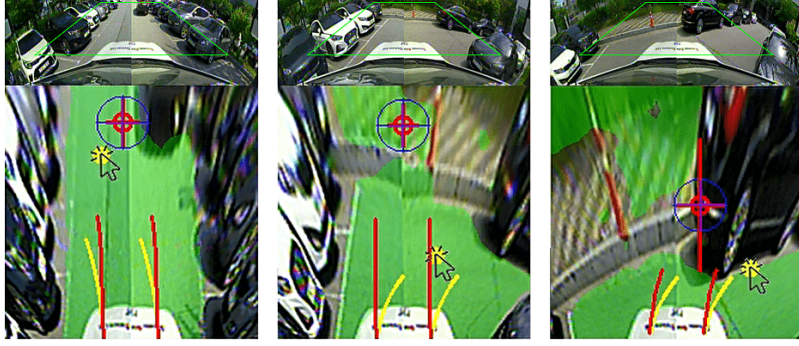
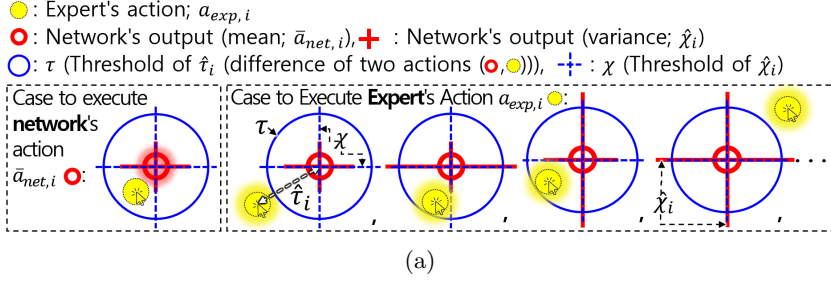


Figure 3.5: Illustration of the data-sampling function of *DAgger*.

(a) Whether executing the network's action ($\bar{a}_{net,t}$) or the expert's action ($a_{exp,t}$). (b) Unsafe or near-collision situations and collecting the additional dataset. In this example, *DAgger* is in iteration $i = 1$, and the network $\pi_{net,i=1}$ has the *behavior cloning* policy, π_{BC} . The yellow point is the newly labeled action $a_{exp,t}$ of the expert while π_{BC} is being executed. The red point in the combined image x_t is the mean of the output by $\pi_{net,i}$: the network's action $\bar{a}_{net,t}$. The blue circle is the threshold τ of $\hat{\tau}_t$ which is the difference between the actions $\bar{a}_{net,t}$ and $a_{exp,t}$. The red lines centered at $\bar{a}_{net,t}$ represent the variance of the output of $\pi_{net,i}$: $\hat{\chi}_t$. The blue dashed lines centered at $\bar{a}_{net,t}$ represent the threshold of $\hat{\chi}_t$ which is the variance of the output of the network $\pi_{net,i}$: χ .

combined image x_t instead of the occupancy grid map s_t (see As Figure 3.4):

$$a_{exp,t} = \pi_{exp}(x_t), \quad (3.2)$$

where π_{exp} indicates the behavior of the expert. The combined image x_t is an image of transparently combining the information about the drivable area to

the *RGB* image: $x_t \in \{RGB \text{ with green: drivable, } RGB \text{ only: non-drivable}\}$, which is because, if s_t is inaccurate (i.e., noisy), the expert may wrongly select the look-ahead point. This situation is shown in Figures 3.4(b) and 3.5(b).

The selection of the look-ahead point in the state (occupancy grid map s_t) takes into account the distribution of obstacles and the drivable area. With this, the following three criteria can be proposed to driving in semi-structured environments, which the expert can refer to and select the look-ahead point a_t :

- (i) The look-ahead point must be within the drivable area.
- (ii) The expert selects the look-ahead point where obstacle avoidance is possible by referring to future trajectories calculated on the basis of the kinematic bicycle model indicated in Figs. 3.4 and 3.5(b).
- (iii) The look-ahead point is selected as far as possible while satisfying the first and second conditions so that the vehicle can move fast.

The look-ahead point can have a geometric relationship with the state through these criteria, so the action pattern for the state can be more clearly found than the steering angle and velocity. Based on these criteria, the vehicle can avoid obstacles and drive toward the drivable area 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 area (see Figure 3.4(a)). At 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 area (see Figure 3.4(b)). At this point, the vehicle can drive at high speed with a small steering angle difference.

3.3.2 Loss Function

The collected data can be used to train the policy π_{net} in a process similar to that of supervised learning. π_{net} is expressed as $\pi_{net}(s_t; \theta)$ parameterized by θ for the state s_t . The process of training $\pi_{net}(s_t; \theta)$ is the process of optimizing θ to minimize the loss function \mathcal{L}_{Gaut} for s_t . This is expressed as $\mathcal{L}_{Gaut}(\pi_{net}(s_t; \theta), a_{exp,t})$, and its detailed expression is given in (3.3). A large number T of dataset $D = \{(s_t, a_{exp,t})\}_{t=1}^N$ is used to optimize θ :

$$\min_{\theta} \sum_{t=1}^T \mathcal{L}_{Gaut}(\pi(s_t; \theta), a_{exp,t}), \quad (3.3)$$

where \mathcal{L}_{Gaut} is the multivariate Gaussian log-likelihood loss function. Through \mathcal{L}_{Gaut} , the policy infers the mean and variance of the look-ahead point [79]:

$$\mathcal{L}_{Gaut} = \frac{1}{n} \sum_j \frac{1}{2} \frac{\mathcal{L}_{t_j}}{\sigma_{t_j}^2} + \frac{1}{2} \log |\sigma_{t_j}^2|, \quad (3.4)$$

where n is the dimension of the look-ahead point and j is the index of n , so j belongs to x and y , and n becomes two. \mathcal{L}_{t_j} in \mathcal{L}_{Gaut} is the non-weighted loss function used to infer the look-ahead point:

$$\mathcal{L}_{t_j} = (a_{exp,t_j} - \bar{\pi}_{net,a}(s_t; \theta)_j)^2, \quad (3.5)$$

where $\bar{\pi}_{net,a}(s_t; \theta)_j$ is the mean of the policy output (look-ahead point) while training; a_{exp,t_j} is label of the look-ahead point. $\sigma_{t_j}^2$ is the variance of the policy output:

$$\sigma_{t_j}^2 = (0.0 - \bar{\pi}_{net,\sigma}(s_t; \theta)_j)^2, \quad (3.6)$$

where $\bar{\pi}_{net,\sigma}(s_t; \theta)_j$ is the policy output while training; 0.0 is a labeled variance value that the network is trained to output a low variance. When this process is performed only once, the trained policy π_{net} is denoted as π_{BC} . When a vehicle

drives with π_{BC} in an environment similar to the trained environment, π_{BC} calculates a look-ahead point similar to that of the expert.

However, if π_{BC} encounters states that are not similar to dataset D or are noisy, π_{BC} may output unsafe or unsafe actions. As shown in Figure 3.5(b), a noisy state is when the boundary of the drivable area or shadow area is not accurately recognized. Furthermore, the location and type of obstacles differ when the dataset for π_{BC} is collected and executed. Here, the vehicle cannot sufficiently avoid obstacles; this is known as the *data mismatch problem*, which occurs when the data for unsafe or near-collision situations are included in the training dataset D less often than situations of driving in a relatively large drivable area or with no misrecognition problems. Thus, the policy π_{net} cannot reflect these situations in π_{BC} well; this is known as the *data imbalance problem*. Moreover, when these problems occur in a driving situation, the error may magnify afterward because π_{BC} has not learned recovery behavior; that is known as the *compounding error problem*.

3.3.3 *Data-sampling Function in DAgger*

The *data-sampling function* is based on *EnsembleDAgger* [80]. This function quantifies the similarity and confidence for the output of the trained policy $\pi_{net,i}$ to determine whether the driving situation of $\pi_{net,i}$ is unsafe or near-collision. The outputs of $\pi_{net,i}$ and the expert behavior π_{exp} are obtained simultaneously (lines 5 and 6 in Algorithm 2) and compared before either is used to control the vehicle (lines 7-9). The discrepancy (error) between the actions of $\pi_{net,i}$ and π_{exp} is calculated (line 7), which is defined by the *SafeDAgger* algorithm [81]. To quantify the confidence of $\pi_{net,i}$, the variance of $\pi_{net,i}$ is obtained: $\hat{\chi}_t$ (line 8) used in the *EnsembleDAgger* algorithm [80].

Algorithm 2: Pseudo-code of *Data-sampling Function* in *Dagger*

*Note The **blue** font is related to the proposed *Dagger* algorithm, *WeightDagger*, introduced in Chapter 4.

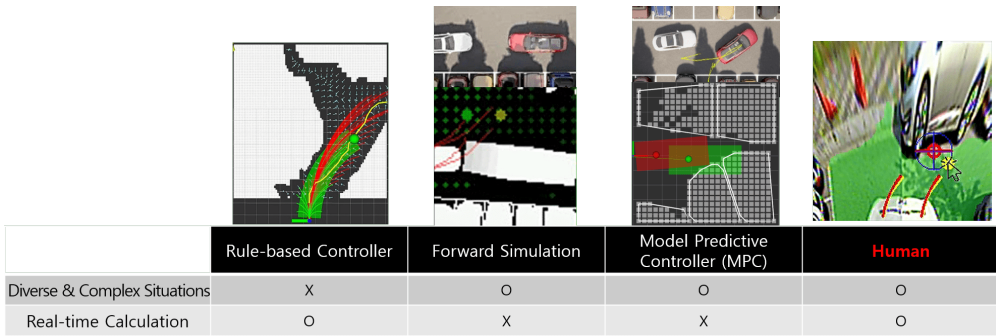
```
1 function Data-sampling Function( $\pi_{net,i}$ )
2 Initialize  $D_i \leftarrow \emptyset$ 
3 Initialize  $n_{tot} \leftarrow 0, n_{net} \leftarrow 0$ 
4 for  $t = 0$  to End of Execution do
5    $\bar{a}_{net,t}, \sigma_{a_{net,t}}^2 \leftarrow \pi_{net,i}(s_t)$  // Output of trained policy
6    $a_{exp,t} \leftarrow \pi_{exp}(x_t)$  // Action of expert
7    $\hat{\tau}_t \leftarrow \|\bar{a}_{net,t} - a_{exp,t}\|_2$  (calculated as (4.3)) // Action discrepancy
8    $\hat{\chi}_{t_{j \in x,y}} \leftarrow \sigma_{a_{net,t}}^2$ 
9   if  $\hat{\tau}_t < \tau$  and  $\hat{\chi}_{t_x} < \chi$  and  $\hat{\chi}_{t_y} < \chi$  then
10    Control the vehicle with  $\bar{a}_{net,t}$ 
11     $n_{net} += 1$ 
12  else
13    Control the vehicle with  $a_{exp,t}$ 
14     $D_i.append(\{s_t, a_{exp,t}, \hat{\tau}_t\})$ 
15    // Collecting  $\hat{\tau}_t$  for WeightDagger in Chapter 4
16   $n_{tot} += 1$ 
17  $\hat{\eta}_i \leftarrow \frac{n_{net}}{n_{tot}}$  // Ratio of executed network actions ( $n_{net}$ )
18 return  $D_i, \hat{\eta}_i$ 
```

By determining whether $\hat{\tau}_t$ or $\hat{\chi}_t$ is less than threshold values τ or χ , an unsafe or near-collision situation can be identified (line 9, see Figure 3.5(a)). In situations of Figure 3.5(b), $\hat{\tau}_t$ is greater than τ (blue circle). In the rightmost case, $\hat{\chi}_t$ (red lines) is greater than χ (blue lines). In these cases, if the vehicle follows the action of the network (red circle), the distance between the vehicle and the obstacle decreases, and the possibility of collision increases. The expert action (yellow circle) is used to control the vehicle to avoid unsafe situations (line 13). Moreover, only the state s_t of this situation and the expert action $a_{exp,t}$ are collected with the additional dataset D_i (line 14). This is used to intensively train the network to overcome unsafe and near-collision situations.

By using the criteria for $\hat{\tau}_t$ and $\hat{\chi}_t$ (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.

3.3.4 Reasons to Use Look-ahead Point As Action



	Rule-based Controller	Forward Simulation	Model Predictive Controller (MPC)	Human
Diverse & Complex Situations	X	○	○	○
Real-time Calculation	○	X	X	○

Figure 3.6: Human labeling is effective.

In the process of performing DAgger training, it is effective to label actions by humans as experts. Other expert models can be used instead of humans, but each has limitations. Rule-based controller model requires modeling and parameter tuning to obtain the action for the state. It calculates actions in real-time, but it is difficult to find suitable models and parameter values for various and complex situations. Another method is forward simulation, which predetermines a set of candidate actions and searches for the one that best satisfies the objective function. The other method, MPC, computes an action that satisfies an objective function and constraints. These two methods can deal with various and complex situations, but they are not suitable for semi-structured environ-

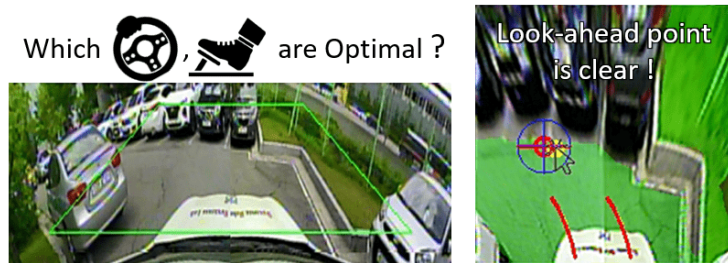


Figure 3.7: Labeling look-ahead point on the state is more clear than using steer-acc/brake.

ments because the more complicated the situation, the longer the search and calculation time. When a human with sufficient experience obtains the action for the state, it is suitable for a complex semi-structured environment because human can find the optimal action in real-time even if the situation is complex.

In autonomous driving, it is appropriate to use the look-ahead point to label the action by human. If the steering-accel/brake is used as the action, the expert suffers two problems in executing *DAgger*, and these can be addressed using the look-ahead point.

First, the network action and expert behavior should be obtained simultaneously as shown in lines 4 and 5 of Algorithm 2. When the vehicle is being controlled by a network action, the expert action cannot be obtained simultaneously if the steering accel/brake is used as the action. In the HG-DAgger [77] data collection process, the joystick (steering wheel and accelerator/brake pedal) must be additionally mounted on the autonomous vehicle. 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, the expert cannot clearly and instantaneously find a steering-accel

and brake value that the vehicle can drive as safe and fast as possible when performing *DAgger*, even if the action is set as the steering-accel/brake and the expert action can be obtained simultaneously with the network action (see Figure 3.7). This is because, when the vehicle is controlled by the network and expert intervention is required, the expert cannot calculate an action value considering the current network action used for vehicle control. Normally 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 value (i.e., amount of change).

In this thesis, the expert selects the look-ahead point that the vehicle should reach on the combined image x_t by referring to the three criteria mentioned in the previous subsection 3.3.1). 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 pattern relationship to be clearly identified, so a neural network can learn the driving pattern more clearly.

3.4 Experimental Setup

3.4.1 Driving Policy Network Training

The deep neural network was used as the driving policy π_{net} and it comprised two pairs of convolutional and max-pooling layers with 32 and 64 channels, respectively. The flattened and fully connected layers with 1,000 nodes were connected to these layers with 25 % and 50 % dropouts. Finally, the fully connected layer with four nodes was linked to predict the position of the look-

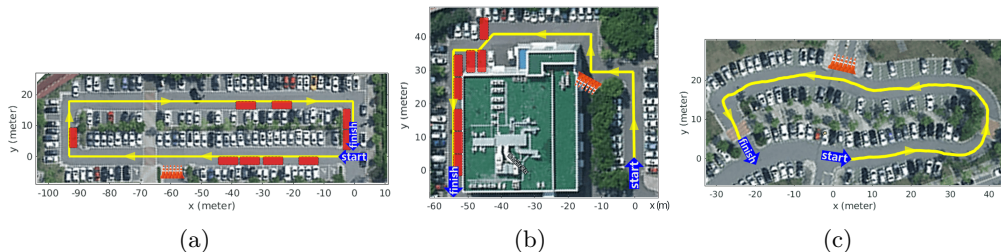


Figure 3.8: Parking lots used in the real autonomous driving experiment.

At intersections, traffic cones are used to guide vehicles to drive in one direction. The yellow line is the center of the drivable area. 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.

ahead point and its confidence. The Adam optimizer with a 10^{-5} learning rate was used, whereas pre-training weights were not used. The epochs were set to 10 k, and the batch size was 512.

The training dataset for the real autonomous vehicle was collected for the one parking lot shown in Figure 3.8(a). The vehicle was driven from the start point to the finish point and from the finish point to the start point (totaling 460 m). Data were collected at intervals of 0.05 s as the vehicle was being driven, which was recorded as a video¹. The final policy was obtained after 5 DAgger iterations ($i = 5$) by using a weighted loss function of the *weightDAgger* algorithm explained in Chapter 4. Increasing the number of DAgger iterations can improve performance, but not significantly. Table 3.1 presents the number of collected data and the percentage of executed network actions.

¹<https://youtu.be/KOXFTEYL-xs>

Table 3.1: *Dagger* Results in Actual Autonomous Vehicle

	BC	$i = 1$	2	3	4	5
Number of Data [ea]	6425	9683	11946	14034	15539	16270
Training Time [min]	45	73	94	114	129	139
Network/Total ($\hat{\eta}_i$)	-	0.44	0.64	0.65	0.74	0.90

**Note.* Equation of $\hat{\eta}_i$ in the third row is represented in line 16 of Algorithm 2.

3.4.2 Model-based Motion-Planning Algorithms

The *tentacle* [65] and *VVF* [68] algorithms were used to compare with the proposed method, which are representative and general algorithms for the candidate path selection and artificial field generation methods, respectively. The occupancy grid map was used as the input for these algorithms. The steering angle was calculated using each algorithm, and the velocity was set to be inversely proportional to the calculated steering angle.

Tentacle Algorithm [65]: 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 using the objective function, and the candidate path with the smallest value is selected. In the experiment, a set of candidate paths of 2.2 m/s was used. The application ratio of the clearance, flatness, trajectory, and forward terms in the objective function was; 1:0:0:0.3. The flatness term was not used because the occupancy grid map in this thesis did not have the occupied probability. Moreover, 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 the narrow drivable area.

VVF Algorithm [68]: Like the artificial potential field algorithm, *VVF* 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 in the APF field. The look-ahead point is searched by descending along the gradient of the field's direction from the front of the vehicle to drive along the combined field. In the experiment, 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 thesis. 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 a narrow drivable area.

3.5 Experimental Result

This section only describes results for driving, and results for perception are described in Chapter 2. The driving experiments were conducted in three parking lots without intersections, as shown in Figure 3.8. The results regarding intersection driving are analyzed in Chapter 2. In the driving policy test, the *Dagger* algorithm was compared with the *tentacle* [65] and *VVF* [68] algorithms, and limitations of these algorithms were analyzed for each situation (see video²).

²<https://youtu.be/0Q1s9fDgiaA>

3.5.1 Quantitative Analysis of Driving Policy

3.5.1.1 Collision Rate

The *collision rate* was used as an evaluation metric to quantify the performance of each driving policy algorithm. This metric showed the number of collisions per 100 m as the vehicle was driven in each parking lot: $100 \frac{cnt_{col}}{len_{path}}$, where cnt_{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 cnt_{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 Figure 3.8. At this point, the vehicle could drive without a collision. The length of the reference path was len_{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 3.2: Collision Rate

		Figure 3.8(a) Trained Environment	Figure 3.8(b) Untrained Environment	Figure 3.8(c) Untrained Environment
Imitation Learning	<i>Dagger</i> (proposed)	0 (0)	0 (0)	0 (0)
Model-based Motion Planning	<i>Tentacle</i>	1.12 (0.95)	1.87 (2.01)	1.47 (1.15)
	<i>VVF</i>	1.38 (1.29)	2.01 (2.15)	1.07 (0.93)

**Note.* The values represent the average *collision rate* over 5 trials. The parentheses indicate additional test results where the vehicle drove from the finish to start points.

Table 3.2 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 *Dagger* 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.

3.5.1.2 Safe Distance Range Ratio

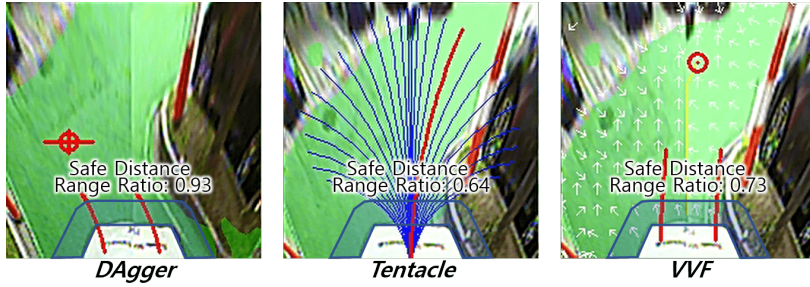


Figure 3.9: Results of safe distance range ratio.

The blue area is the safe distance range, and its ratio is a measure of how much drivable area (green) exist within the blue area.

Moreover, in order to evaluate collision safety with obstacles, a ratio of the drivable area within 1.0 m range from the end of ego vehicle's bumper, *safe ratio*, was measured: $\frac{N_{dri}}{N_{ran}}$, 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 the blue range in Figure 3.9. Measuring this ratio can measure how safely the vehicle can maintain a safe distance from obstacles on average. This range and ratio are shown in Figure 3.9 and Table 3.3. *Dagger* has the highest safe distance range ratio.

Table 3.3: Safe Distance Range Ratio

		Figure 3.8(a) Trained Environment	Figure 3.8(b) Untrained Environment	Figure 3.8(c) Untrained Environment
Imitation Learning	<i>Dagger</i> (Proposed)	0.83	0.72	0.91
Model-based Motion Planning	<i>Tentacle</i>	0.69	0.63	0.81
	<i>VVF</i>	0.71	0.64	0.85

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

3.5.2 Qualitative Analysis of Driving Policy

3.5.2.1 Limitations of *Tentacle* Algorithm

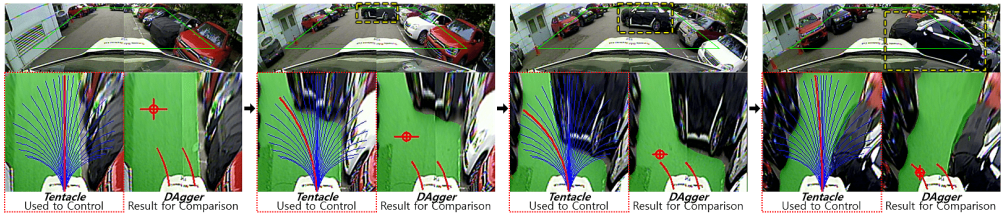


Figure 3.10: Problems of the *tentacle* algorithm.

The vehicle using *tentacle* did not drive in the middle of the drivable area and did not avoid obstacles safely. The blue lines in the *tentacle* image represent the candidate paths. The red line represents the selected path to track.

In the *tentacle* algorithm test, the vehicle drove near the boundary between the drivable and non-drivable areas rather than the center of the drivable area after avoiding obstacles or escaping the corner, which is shown in the leftmost image in Figure 3.10. This is because *tentacle* selects the most forward-facing candidate path with no obstacle among the candidate paths. Then, the vehicle drove at the minimum distance from side obstacles, increasing the possibility of collision. In the same situation, *Dagger* tried to direct the vehicle toward

the center of the drivable area. 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 Figure 3.10 show that, when the vehicle was driving on the side of the drivable area 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 area and avoided obstacles, it did not avoid the obstacle with sufficient clearance, which is because the *tentacle* algorithm chose the candidate path with the least spacing to avoid obstacles. In contrast, *Dagger* tried avoiding obstacles with sufficient safe distance in advance.

3.5.2.2 Limitations of *VVF* Algorithm

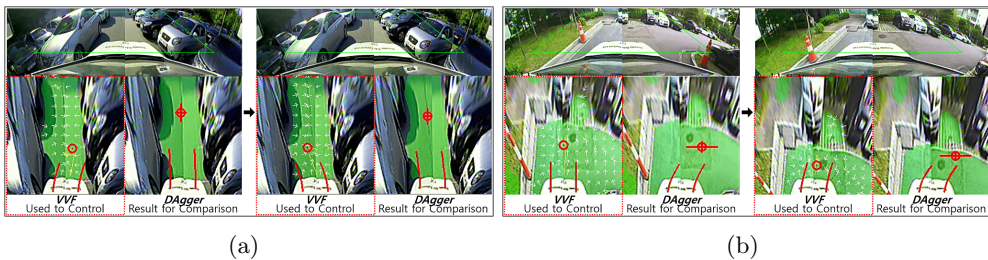


Figure 3.11: Problems with the *VVF* algorithm.

The white arrows represent the field direction. (a) Oscillation in a narrow drivable area. (b) The vehicle could not enter the right side of the drivable area in advance at a right-angled corner.

In the *VVF* test, an oscillation problem occurred in narrow drivable areas where the vehicle frequently turned left and right (see Figure 3.11(a)). In such spaces, the magnitudes of the fields from the two obstacles were almost the same because only a repulsive field was applied, 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 range of the repulsive force. However, the probability of collision would be increased in other situations, especially where the curvature changed significantly. Whereas, with *Dagger*, the vehicle drove stably without oscillation by imitating the expert who drove toward the middle of the drivable area even in narrow spaces.

As shown in Figure 3.11(b), with *VVF*, the vehicle could not enter the drivable area when the curvature changed rapidly, such as in right-angled corners. This problem may be addressed using the goal point obtained by the global information would be used as an attractive field. In contrast, this problem did not occur with *Dagger* because when data was collected for this situation, the expert selected a look-ahead point where the vehicle could drive the furthest without collision.

3.5.2.3 Limitations of Both *Tentacle* and *VVF*

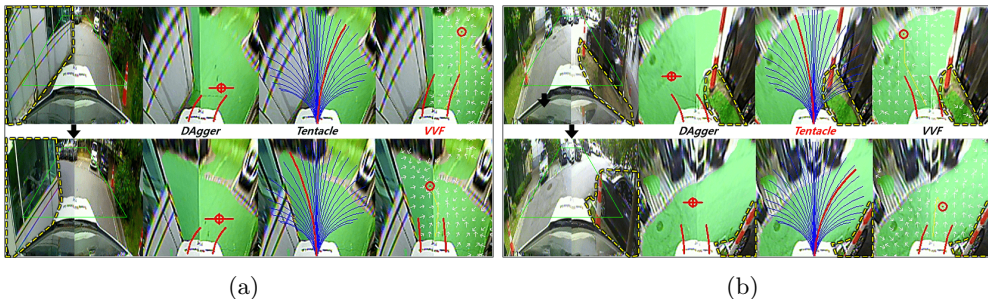


Figure 3.12: Problems for driving in narrow drivable area with large curvature changes: (a) *VVF*, (b) *Tentacle*.

Figure 3.12 shows the problems of the *VVF* and *tentacle* algorithms when the curvature and width of the drivable area changed more than the space where the vehicle was currently driving. The vehicle headed into the drivable

area on the side of the adjacent obstacle before sufficiently avoiding it. This is because the *tentacle* algorithm 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.

3.5.2.4 Driving Results on Noisy Occupancy Grid Map

The occupancy grid map was not recognized accurately in complex and shadowy environments (i.e., noisy input) 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 shown from the experiment in Figure 3.13, a vehicle could drive without collision, even though there was noise in one trained environment and two untrained environments. However, the *Tentacle* and *VVF* algorithms encountered several problems.

As shown in Figure 3.13(a), the boundary between the speed bump and the road was erroneously recognized as a non-drivable area (i.e., noise). *Dagger* was not affected by this 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 area.

Figures 3.13(b) and 3.13(c) show situations where the noise was caused by shadows. With *Dagger*, the vehicle drove towards the drivable area with

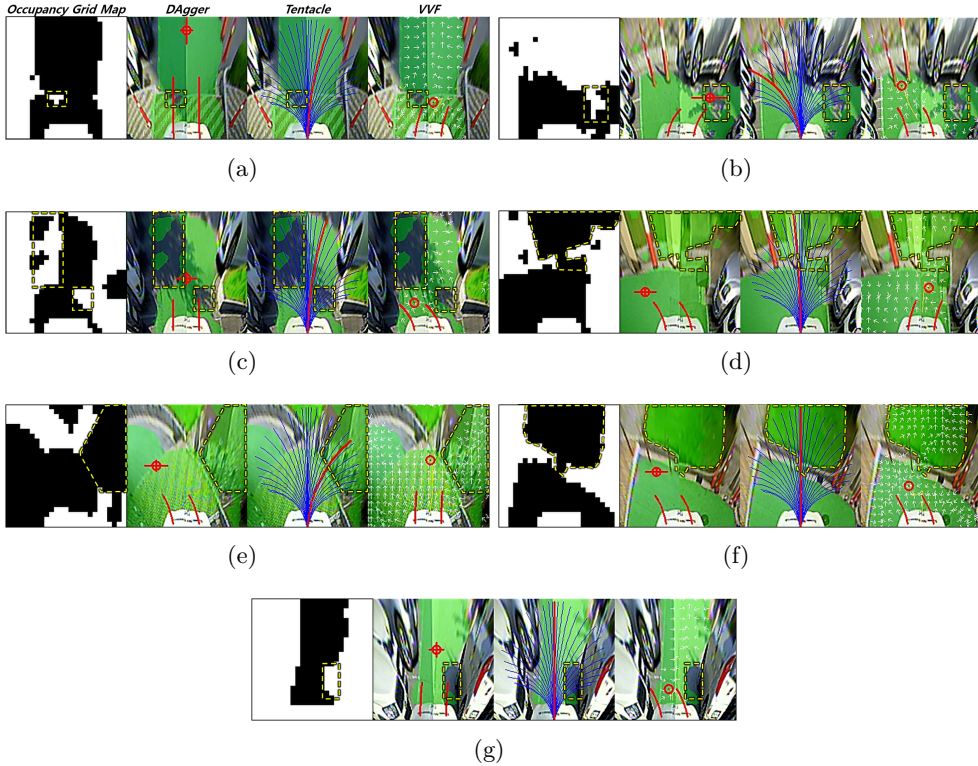


Figure 3.13: Driving results with *DAgger* and noisy occupancy grid map.

DAgger did not encounter any problems in this situation. However, the vehicle could not drive smoothly or headed toward 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.

fewer oscillations than *tentacle* and *VVF*. This is because a training dataset for *DAgger* contained similar situations, where the expert selected action without being affected by the noise. With the *tentacle* and *VVF* algorithms, however, the vehicle in the Figure 3.13(b) situation avoided the shadows and then drove toward the largest drivable area blocked by obstacles, making it unable to drive any further. These algorithms also had more oscillation problems than *DAgger* especially in Figure 3.13(c) situation.

Figures 3.13(d), 3.13(e), and 3.13(f) present situations in which a non-

drivable area was recognized as a drivable area. In detail, not only the non-drivable area at the curb (i.e., the boundary of the drivable area) but also the space behind the curb was recognized as the drivable area. Except for the curb, the vehicle attempted to drive toward the largest drivable area using *Dagger*. However, *tentacle* was influenced by the noise at the curb, which is detected to be a drivable area. 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 area (see Figs. 3.13(d) and 3.13(e)).

As shown in Figure 3.13(g), the vehicle with the *VVF* algorithm took action 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.

3.5.2.5 Intersection Navigation

The model-based motion-planning algorithms (*Tentacle* [65] and *VVF* (Velocity Vector Field) [66]) and learning-based motion-planning algorithm (*Dagger* (Data Aggregation), described in Chapter 3 [70]) were used for the intersection driving test. These algorithms utilized OGM_{mer} as input and did not require the localization data, global path, and goal point. These were tested in three parking lots (see Figure 2.11), and the driving results are shown in Figure 3.14. They were able to calculate an action toward the branch road and could avoid obstacles existing at the branch road because obstacles were regarded as the non-drivable area in OGM_{mer} (see Figure 3.14(d)). Among the total 18 intersections, the learning-based algorithm, *Dagger*, did not encounter any problems in the above-mentioned two intersections and was able to navigate the entire intersection in the three parking lots. The driving result in the indoor parking

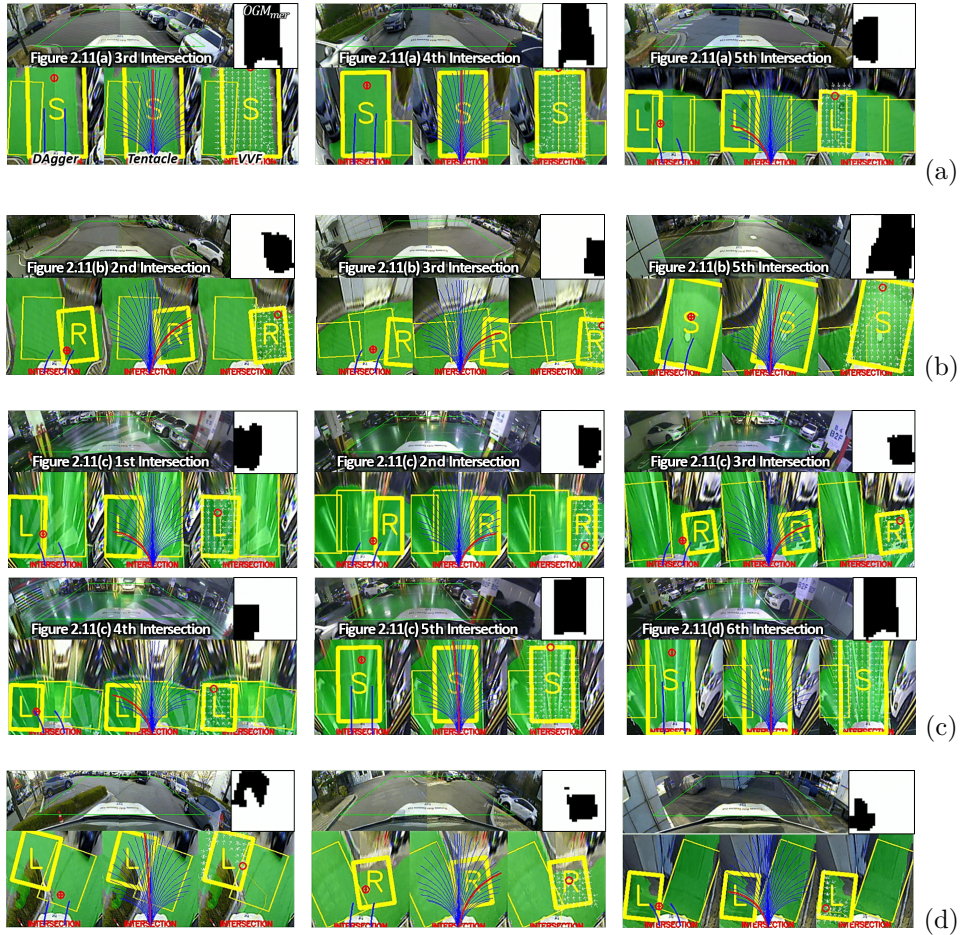


Figure 3.14: Intersection navigating results using motion-planning algorithms. (a) map1 (Figure 2.11(a)), (b) map2 (Figure 2.11(b)), and (c) map3 (Figure 2.11(c)), (d) Case with obstacles existing in branch road box.

lot (Figure 2.11(c)) was recorded as video³.

The vehicle using *Tentacle* and *VVF* drove through intersections successfully except for two intersection situations where two intersections were close together (Figure 2.11(a) 1st and 2nd intersections) and the situation in which the branch road was narrowed by an obstacle (Figure 2.11(b) 4th intersection).

³<https://youtu.be/b63HBKptKV4>

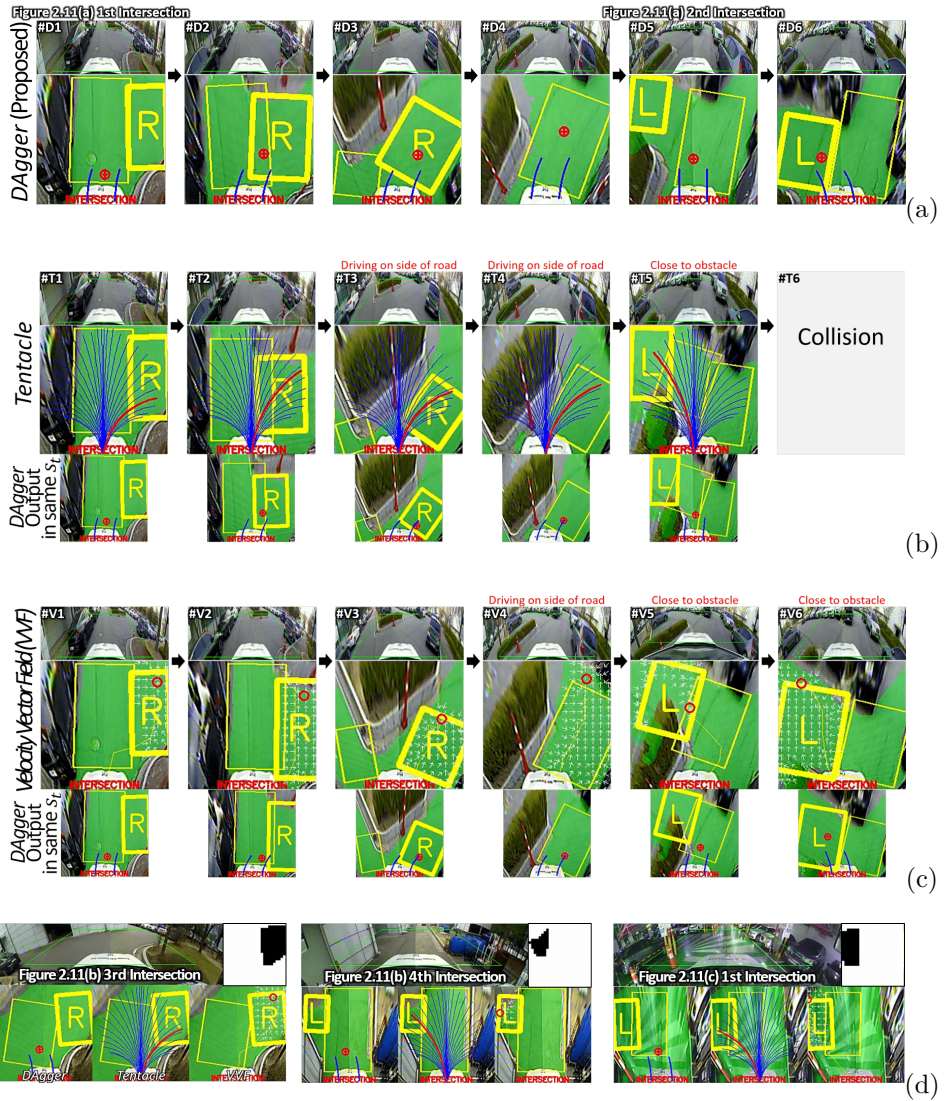


Figure 3.15: Problems of model-based motion-planning algorithm at intersection.

(a) *DAgger*: no problem, (b) *Tentacle*: collision occurrence, (c) *VVF*: close to obstacles. (d) Problems *Tentacle* and *VVF* of when approaching an intersection.

Figure 3.15 shows an example of the first failure situation. After passing the first intersection shown in #T3 and #T4 of Figure 3.15(b), and #V4 of Figure 3.15(c), the vehicle using *Tentacle* and *VVF* drove close to obstacles and

did not drive into the center of the drivable area. This situation is shown in #T5 of Figure 3.15(b), #V5 and #V6 of Figure 3.15(c), and Figure 3.15(d). This was because there was no way to preferentially avoid obstacles close to the vehicle existing between the vehicle and the branch road in *Tentacle* and *VVF*. On the other hand, the vehicle with *Dagger* preferentially avoided obstacles close to it, because the driving policy was trained on data on driving corner roads taking into account the vehicle’s turning radius.

3.6 Conclusion

In this chapter, an autonomous driving method using a vision-based occupancy grid map and imitation learning is proposed to deal with semi-structured environments such as parking lots. The occupancy grid map obtained via the U-net-based deep neural network is used as an input for imitation learning. Through the geometric relationship between the occupancy grid map and the look-ahead point, the expert clearly labels this point when collecting the training data, and the *Dagger* algorithm is used for autonomous driving in semi-structured environments.

In real-environment and simulation experiments, a vehicle with the proposed method could drive toward the drivable area while avoiding obstacles reactively in real-time without using a global map and localization. In the actual experiments, the vehicle with *Dagger* could drive more smoothly and safely than with the *tentacle* and *VVF* algorithms in environments where the width and curvature of the drivable area varied significantly. Especially, *Dagger* was more robust when the occupancy grid map was not accurately perceived or was noisy due to a shadow. *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 for various environments with intersections and dynamic obstacles.

Chapter 4

WEIGHT DAGGER ALGORITHM FOR REDUCING IMITATION LEARNING ITERATIONS

4.1 Introduction

The *Dagger* algorithm is introduced in Sec. 3.2.3 of Chapter 3, and this chapter proposes a method to reduce *Dagger* iterations by further increasing a *Dagger* accuracy. *Dagger* are often used in robotics when a single imitation learning step cannot obtain the desired policy. *Dagger* repeatedly retrains policies by aggregating additional data obtained by executing a prior trained policy. *Dagger* variant algorithms have been proposed to reduce *Dagger* iterations. They sample data that the policy cannot accurately imitate the expert behavior. Nevertheless, additional *Dagger* iterations are required if these data are insufficiently sampled, requiring considerable human effort.

Therefore, this thesis proposes a new *Dagger* training algorithm, *WeightDagger*, that can imitate expert behavior more accurately, particularly when the data are insufficiently sampled [70]. *WeightDagger* defines the weight value in

the dataset and uses the weighted loss function. This value is calculated through the discrepancy between policy and expert actions during DAgger execution. Based on the weighted values, the policy is trained with a high learning rate for normally failing situations. *WeightDAgger* is applied to autonomous driving in semi-structured environments and several walking robots. It obtains the desired policy in fewer DAgger iterations compared with existing DAgger algorithms.

The remainder of this chapter is structured as follows. DAgger variant algorithms and their explanations are explained in Sec. 4.2. Sec. 4.3 introduces *WeightDAgger*. Sec. 4.4 and Sec. 4.4.3 compare the proposed and variant DAgger algorithms application to autonomous driving and various types of walking robots. Finally, Sec. 4.5 concludes this chapter.

4.2 Related Works & Background

Most studies on imitation learning use a data aggregation (*DAgger*) algorithm, which collects additional data from trained and expert policies [71]. These data are aggregated with existing training data and used to retrain policies. The accuracy of imitating expert action can be increased by increasing the amount of aggregated data. Thus, the desired policy comes from repeating *DAgger*.

DAgger variant algorithms [74,76,77,80–86] have been proposed to aggregate additional data selectively and reduce the data imbalance problem in imitation learning. Thus, DAgger iterations and the human effort required to collect data can be reduced. References [82, 83] sample out-of-distribution or unseen states in the training dataset by comparing state similarity, and a proportion of these states increases in the dataset. There are algorithms [81, 84, 85] (*SafeDAgger*) for sampling data that the policy cannot accurately imitate by calculating the discrepancy between the expert action and policy action. BAgger [86] samples

data with high variance of the policy action, which means it, has low action confidence. *EnsembleDagger* [80] collects data with low accuracy/confidence during DAgger execution, ensuring the safety of robot control. In *HG-Dagger* [77], the expert judges sub-optimal and failure situations, and only the data for these situations are sampled during DAgger execution.

Algorithms [74, 76] distinguish easy tasks and sample fewer data for them. Hence, they can focus on data for difficult tasks in the training process. However, these algorithms are only applied if the data can be classified. According to [76], data that are not similar to the sampled data in the training dataset are excluded (down-sampled) using the replay buffer to increase the learning rate of the sampled data. However, this may result in reduced training accuracy for the down-sampled data. The sampled dataset is further augmented to the training dataset in other DAgger variant algorithms by [87, 88]. However, it is unclear how to augment the data for which state, depending on the scenario or task other than those mentioned in [87, 88].

A policy trained on the sampled data with the aforementioned algorithms has higher accuracy than a policy trained on the unsampled (entire) data in the same DAgger iteration. This is because, as the proportion of sampled data in the training dataset increases, the policy can be trained with higher weights on the sampled data. Even with these algorithms, quite low accuracy/confidence data may not be sufficiently sampled, and the data imbalance problem is not resolved completely. Hence, further DAgger iterations are needed to sufficiently gather data, which requires considerable human effort.

Algorithm 3 is the data-sampling function included in Algorithm 1 (basic structure of *Dagger*). This function shown in this chapter is similar to that shown in Sec. 3.3.3 of Chapter 3 and contains more detailed description of DAgger variant algorithms, such as *VanillaDagger* [71], *SafeDagger* [81], *En-*

Algorithm 3: Data-sampling function in *Dagger*

*Note The black, cyan, brown, green, and blue fonts denote *VanillaDagger* [71], *SafeDagger* [81], *EnsembleDagger* [80], *HG-Dagger* [77], and *WeightDagger*.

```
1 function Data-Sampling Function( $\pi_i$ )
2 Initialize  $D_i \leftarrow \emptyset$ 
3 for  $t = 0$  to End of Execution do
4    $\bar{a}_t, \sigma_t^2 \leftarrow \pi_i(s_t)$ 
5    $a_{exp,t} \leftarrow \pi_{exp}(s_t)$ 
6    $\hat{\tau}_t \leftarrow \|\bar{a}_t - a_{exp,t}\|_2$  (calculated as (4.3))
7    $\hat{\chi}_t \leftarrow \sigma_t^2$ 
8   if Condition according to Dagger algorithms [71] [81], [80], [77]
     is satisfied then
9     Control robot with expert action,  $a_{exp,t}$ 
10     $D_i.append(\{s_t, a_{exp,t}\})$  // In weightDagger, this line is replaced
    by (4.4) to use the weighted loss function.
11  else
12  Control robot with trained policy action,  $\bar{a}_t$ 
13 return  $D_i$ 
```

sembleDagger [80], and *HG-Dagger* [77]. Except for *VanillaDagger*, these representative algorithms can distinguish unsafe or near-collision situations caused by the trained policy based on different criteria. Actions of the trained (π_i) and expert (π_{exp}) policies are obtained simultaneously (lines 4 and 5 in Algorithm 3). In *SafeDagger* and *EnsembleDagger*, the action discrepancy $\hat{\tau}_t$ is calculated as the distance between these actions (line 6). Moreover, *EnsembleDagger* obtains the variance of π_i , i.e., $\hat{\chi}_t$, to quantify the confidence of the policy action \bar{a}_t (lines 4 and 7); $\hat{\chi}_t$ is obtained using Gaussian process regression [80].

Dagger variant algorithms sample data by defining different conditions (line 8). In *VanillaDagger*, the expert action controls the robot with probability $\lambda^i \beta_0 \in [0, 1]$ ($\lambda \in (0, 1)$). Thus, as the Dagger iteration increases, more trained policy action is selected. Under *SafeDagger*, if the action discrepancy $\hat{\tau}_t$ is larger than the threshold τ , the state visited by π_i is regarded as situations wherein π_i

cannot accurately imitate $a_{exp,t}$. *EnsembleDAgger* considers whether the action discrepancy or the variance are larger than thresholds, i.e., $\tau \leq \hat{\tau}_t$ or $\chi \leq \hat{\chi}_{t_x}$ or $\chi \leq \hat{\chi}_{t_y}$. In *HG-DAgger*, the expert judges the unsafe or near-collision situations.

If the above conditions are satisfied, $a_{exp,t}$ is used to control the robot (line 9), and additional data D_i are gathered (line 10). Otherwise, it is controlled using the policy action \bar{a}_t (lines 11 and 12). The method used herein for action labeling is more specific than that of DAgger variant algorithms; In these algorithms, only the state is sampled from the data-sampling function. After that, the expert labels the action according to the state. However, it is difficult for the expert to label actions without actually controlling a robot. Hence, in *WeightDAgger*, actions are labeled while the robot is being controlled, similar to that presented in the *HG-DAgger* algorithm [77] (line 10).

4.3 Proposed Method

The proposed DAgger training algorithm, *WeightDAgger*, reduces the data imbalance problem that existing DAgger variant algorithms cannot completely resolve. These algorithms train the policy repeatedly by sampling low accuracy or confident data while executing the trained and expert policies. In the proposed algorithm, different weights according to states are calculated as the action discrepancy (through **Step 1**). These weights are paired with the entire training dataset by comparing the similarity (through **Step 2**), by which the policy is trained with a high learning rate on data with low accuracy. Thus, *WeightDAgger* imitates expert action more accurately on these data than existing variants *DAgger* in the same DAgger iteration. Consequently, DAgger iteration and human effort to collect additional data are reduced.

4.3.1 Weighted Loss Function in *WeightDagger*

The weighted loss function is one of the weight balancing methods used to address the data imbalance problem in machine learning; in the classification tasks, the accuracy of distinguishing classes with a relatively small data proportion is lower than that of classes with considerably more data. The weight is calculated high for the small data class to train the policy with a high learning rate. Thus, the accuracy for the small data class is similar to that of the large data class.

A weight value is defined according to the state to apply the weighted loss function to imitation learning. During *Dagger* execution, the state accuracy is quantified by calculating the discrepancy between the policy and expert actions. A state with a large discrepancy has out-of-distribution/unseen states, which exist in a small proportion in the training dataset. The weight is defined as being proportional to the discrepancy, and the policy is trained with a relatively high learning rate for the low-distributed states in the training dataset. Therefore, after training, the accuracy of the state is similar to that of a sufficiently distributed state.

In *WeightDagger*, the non-weighted loss function \mathcal{L}_{t_j} (3.5) is replaced by the weighted loss function:

$$\mathcal{L}_{W_t} = W_t \mathcal{L}_t, \quad (4.1)$$

where W_t is the weight value, and \mathcal{L}_{W_t} is the weighted loss function. In the policy training process, the change amount of parameter θ in the policy $\pi(s_i; \theta)$, is calculated as large as the weight W_t [89]. W_t can be expressed as follows:

$$W_t = (1.0 + \alpha \hat{\tau}_t), \quad (4.2)$$

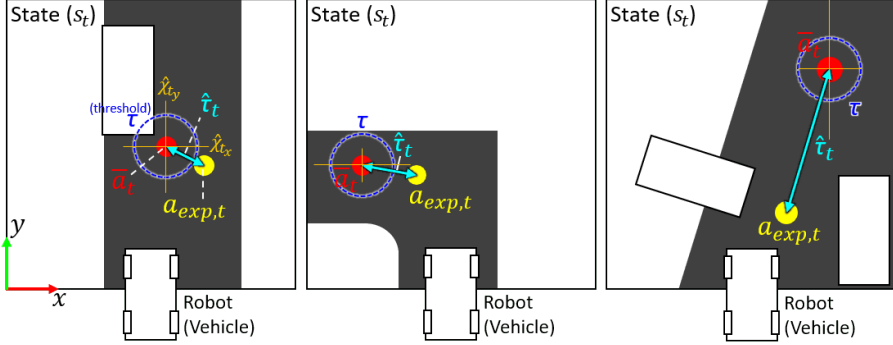


Figure 4.1: *WeightDagger* application to autonomous driving.

The black and white areas in s_t are the drivable/non-drivable areas, respectively. The look-ahead point is used as the action, and the vehicle tracks this point. The yellow ($a_{exp,t}$) and red (\bar{a}_t) circles are the expert and policy actions, respectively. The blue line $\hat{\tau}_t$ is the action discrepancy between \bar{a}_t and $a_{exp,t}$. If $a_{exp,t}$ is selected outside at τ (blue dashed circle) or the variance of the action $\hat{\chi}_t$ (brown line) is larger than threshold (χ), s_t is regarded that \bar{a}_t is unsafe or low-confidence. In this case, $a_{exp,t}$ is used to control the vehicle, and data including $\hat{\tau}_t$ are gathered (see lines 8-10 in Algorithm 3).

where α is the gain for the action discrepancy $\hat{\tau}_t$ mentioned in line 6 of Algorithm 3. $\hat{\tau}_t$ is the normalized value for the distance between the expert action a_{exp,t_j} and the trained policy action \bar{a}_{t_j} , which can be obtained during DAgger execution as follows:

$$\hat{\tau}_t = \sqrt{\frac{\sum_j (a_{exp,t_j} - \bar{a}_{t_j})^2}{n}}, \text{ where } \hat{\tau}_t \in [0, 1]. \quad (4.3)$$

The numerator in (4.3) represents the distance between two actions. The actions of each dimension, j (e.g., $j \in \{x, y\}$; see Figure 4.1), are scaled from 0 to 1; $a_{exp,t_j}, \bar{a}_{t_j} \in [0, 1]$. The denominator is used to normalize the numerator from 0 to 1 according to the action's dimension n (e.g., $n: 2$; see Figure 4.1) to compare the accuracy of policies trained with different DAgger algorithms between 0 and 1.

The degree to which the policy cannot accurately imitate the expert action in a particular state can be quantified as the action discrepancy $\hat{\tau}_t$. By reflecting $\hat{\tau}_t$ to the loss function \mathcal{L}_{Gau_t} in (3.4), it is evident that the loss value is larger than that without using $\hat{\tau}_t$ for low accuracy states, and the policy is trained with a large learning rate. Therefore, a policy trained with $\hat{\tau}_t$ can calculate actions that are more consistent with expert actions than policies trained without $\hat{\tau}_t$.

α in the weight, (4.2), regulates the application rate of $\hat{\tau}_t$ to the weighted loss function. The higher the α , the larger the loss value calculated during training for states where the expert action is inaccurately imitated (large $\hat{\tau}_t$). However, if α is considerably high, the policy may not converge with the lowest loss value, which is similar to failure to converge when using the too large learning rate value of the optimizer. Therefore, to determine a proper value for α , the accuracy was experimentally compared by training the policy with different values. The results are presented in Section 4.4.2.3).

To apply $\hat{\tau}_t$ to the weighted loss function, $\hat{\tau}_t$ is additionally paired with the additional dataset D_i obtained via the existing DAgger variant algorithms. This pairing process is implemented when obtaining D_i (line 10 in Algorithm 3), and ‘ $D_i.append(\{s_t, a_{exp,t}\})$ ’ is replaced by

$$D_i.append(\{s_t, a_{exp,t}, \hat{\tau}_t\}), \quad (4.4)$$

where $\hat{\tau}_t$ is the action discrepancy value obtained in the t -th step when collecting the additional dataset D_i in the i -th DAgger iterations. In the BC dataset D_{BC} , $\hat{\tau}_t$ cannot be obtained; thus, all $\hat{\tau}_t$ values are initialized to zero.

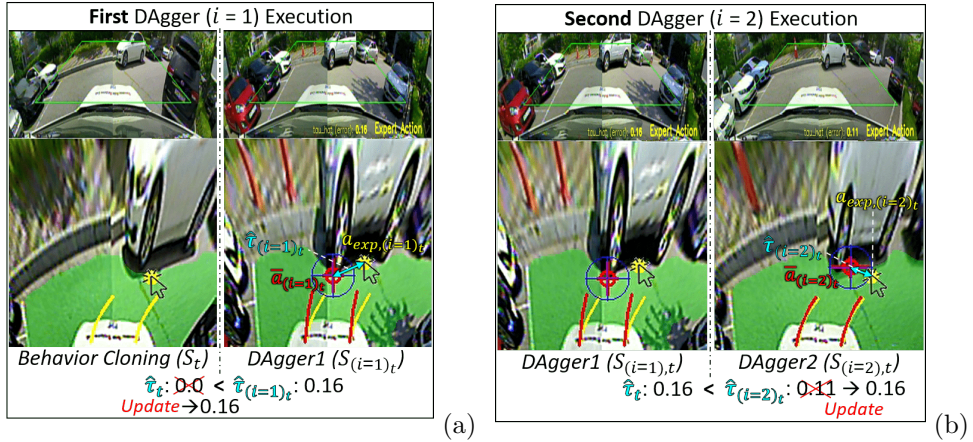


Figure 4.2: Applying the weight update process (*step 2*) of *WeightDagger* Algorithm 4 and line 6 in Algorithm 1 (a) *DAGger* executed once ($i = 1$), where the action discrepancy $\hat{\tau}_{(i=1)_t}$ is paired with data obtained using *DAGger* and updated to similar states in the *BC* dataset (lines 6-7 in Algorithm 4). (b) Action discrepancy $\hat{\tau}_{(i=2)_t}$ still occurs second *DAGger* iterations ($i = 2$); the larger action discrepancy is updated in this state such that the policy is trained to further reduce the discrepancy (lines 8-9 in Algorithm 4).

4.3.2 Weight Update Process in Entire Training Dataset

A weight update process applies to the dataset D as well as the additional dataset D_i collected using *DAGger*. In order to apply the weighted loss function (4.1) to D , the action discrepancy $\hat{\tau}_t$ in D (which is zero) must be updated to a non-zero value. Therefore, data exhibiting a high similarity to the state of D_i are searched among the state of D , and $\hat{\tau}_t$ among these two data is updated to a larger $\hat{\tau}_t$. This is conducted in Algorithm 4. Through this step, the policy is trained with a weight on all similar data relevant to the situation wherein the policy cannot accurately imitate expert action.

Algorithm 4 is added to the *DAGger* algorithm (line 6 in Algorithm 1). D is the dataset used for training the policy π_i (Algorithm 3); t is the data index in

Algorithm 4: Weight Update Process in *WeightDagger* (**Step 2**)

```
1 function Weight Update( $D, D_i$ ) // ( $D$ : existing dataset  $D_i$ :  
   additional dataset)  
2 for  $i_t = 0$  to Len of  $D_i$  (In  $i_t, t$  is a variable) do  
3   for  $t = 0$  to Len of  $D$  do  
4      $\hat{\varepsilon} \leftarrow \text{SIMILARITY}(s_{i_t}, s_t)$   
5     if  $\varepsilon \leq \hat{\varepsilon}$  then // Cases where states of  $D_i$  and  $D$  are  
       similar.  
6       if  $\hat{\tau}_t < \hat{\tau}_{i_t}$  then  
7          $\hat{\tau}_t \leftarrow \hat{\tau}_{i_t}$   
8       else  
9          $\hat{\tau}_{i_t} \leftarrow \hat{\tau}_t$   
10 return  $D, D_i$ 
```

D, D_i is the additional dataset obtained through the data-sampling function; i_t is the index of the data in D_i . The similarity between s_{i_t} and s_t is calculated and denoted as $\hat{\varepsilon}$ (line 4 in Algorithm 4). Weight updating is conducted when $\hat{\varepsilon}$ is larger than the similarity threshold ε (line 5). This process comprises two cases (first case lines 6-7 and second case lines 8-9).

In the first case, if $\hat{\tau}_{i_t}$ is larger than $\hat{\tau}_t$ (line 6), $\hat{\tau}_t$ is replaced by $\hat{\tau}_{i_t}$ (line 7), where $\hat{\tau}_{i_t}$ and $\hat{\tau}_t$ are the action discrepancies paired to s_{i_t} and s_t , respectively (see Figure 4.2(a)). A large action discrepancy (larger than τ) occurred in the state $s_{(i=1)_t}$, because no sufficient data were similar to $s_{(i=1)_t}$ in the *BC* dataset. The action discrepancy was paired to s_t , which is similar to $s_{(i=1)_t}$ in the *BC* dataset, via **Step 2**. Therefore, even with the *BC* dataset, the policy was trained using the weighted loss function.

In the second case, if $\hat{\tau}_t$ is larger than $\hat{\tau}_{i_t}$ (line 8), $\hat{\tau}_{i_t}$ is replaced by $\hat{\tau}_t$ (line 9). For example, as shown in Figure 4.2(b), the action discrepancy $\hat{\tau}_{(i=2)_t}$ for state $s_{(i=2)_t}$ was slightly reduced after the first *DAGGER* iteration ($i = 1$). Nevertheless, in this situation, the action discrepancy $\hat{\tau}_{(i=2)_t}$ still existed. In

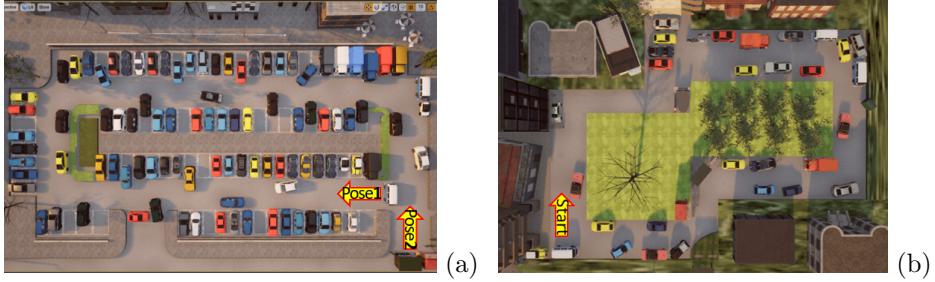


Figure 4.3: Semi-structured environments used for autonomous driving test created using UNREAL ENGINE4 (a) Trained environment (data collection and driving test) and (b) **U**ntrained environment (only driving test).

Step 2, the policy $\pi_{(t=3)}$ was trained with the weight $\hat{\tau}_t$, which was larger than $\hat{\tau}_{(i=2)_t}$, so that the associated data was trained with a larger weight.

4.4 Experiments

4.4.1 Experimental Setup

WeightDagger was applied to the DAgger variant algorithms [71, 77, 80, 81] mentioned in Section 4.2. These performances were evaluated in autonomous driving experiments in parking lots. These environments were built by Unreal Engine 4 and had several complex obstacles; the width of the drivable space was narrow, and the change in its curvature was large (see Figure 4.3), requiring multiple DAgger executions. Using the CARLA simulator, the training data were collected, and the trained policy was executed [90] I uploaded the environment configurations, code, training dataset, and policies on Github¹.

The state was a 25×25 occupancy grid map with a 0.4 m grid (see Fig-

¹<https://github.com/joonwooahn/WeightDagger>

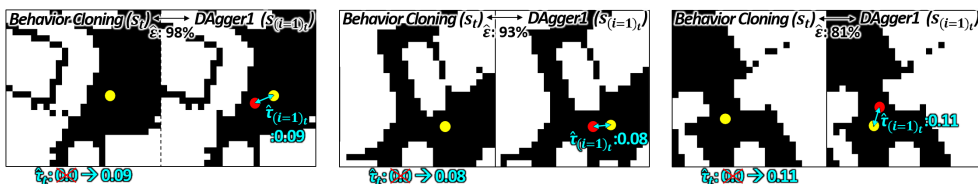


Figure 4.4: Weight update processes in *WeightDagger* (*Step 2*)

ure 4.4). the look-ahead point used as the policy action. These are the same as the imitation learning-based driving policy mentioned in Sec. 3.3 in Chapter 3.

A deep neural network was used as the policy $\pi(s; \theta)$, comprised of two convolutional and max-pooling layers with 32 and 64 channels, respectively. The flattened and fully connected layers with 1,000 nodes were connected to these layers; 25% and 50% dropouts were applied prior to connecting each layer. Finally, the fully connected layer with four nodes was linked to predict the position of the look-ahead point and its confidence. An Adam optimizer with a 10^{-5} learning rate was used, and pre-training weights were not used. The epochs were set to 10 k, and the batch size was 512.

The expert selected the look-ahead point using a mouse pointer to collect the training data, and the state action pair data were collected every 0.025 s (see video²). The parameters of *VanillaDagger*, i.e., β_0 and λ , were set to 1.0 and 0.5 respectively. τ (threshold of $\hat{\tau}_t$) in *SafeDagger* and *EnsembleDagger* was 0.05. χ (threshold of $\hat{\chi}_t$) in *EnsembleDagger* was 0.05. In *HG-Dagger*, the expert took the action when situations were unsafe or near-collision. The structural-similarity index measure algorithm was used to quantify the similarity between states using the occupancy grid map (line 4 in Algorithm 4; see Figure 4.4) [91].

The thresholds of $\hat{\tau}_t$ (action discrepancy) and $\hat{\chi}_t$ (confidence of action), τ

²https://youtu.be/FvuF9gg7_YY

and χ , were both set to 0.05. With these thresholds, unsafe or low confident data are collected as much as possible without colliding with the obstacles [80]. To collect the training data, the expert selected the look-ahead point using a mouse pointer, and the state-action pair data were collected every 0.025 s. The expert drove the vehicle from Pose1 to Pose2, and from Pose2 to Pose1 (see Figure 4.3(a) and video³). The structural similarity index measure algorithm [91] was used to quantify the similarity between the states, the occupancy grid map (line 4 in Algorithm 4; see Figure 4.4).

4.4.2 Experimental Results

The performance of *WeightDagger* may vary depending on the similarity threshold ε (line 5 in Algorithm 4) and parameter α in (4.2). A parameter analysis was performed using *EnsembleDagger* as a representative among several *Dagger* algorithms, and the value that obtained the highest accuracy was applied equally to [71, 77, 81]. Overall, 80% of the dataset was used as the training set, and the rest was used for the test set. The test dataset was used to measure the accuracy calculated as $1.0 - \hat{\tau}$ (4.3). Additionally, it was classified into two types: accurately and inaccurately trained states. If $\hat{\tau}_t$ was less than τ , then this corresponds to the first data type, and otherwise, this corresponded to the second data type. Here, $\hat{\tau}_t$ is the t -th data that exists in first *Dagger* dataset.

4.4.2.1 Ablation Study According to τ

In the process of executing *Dagger*, data is collected only when the difference between actions is greater than τ (see line 9 in Algorithm 2). The larger the τ , the higher the probability of collecting data from a different distribution

³<https://youtu.be/GEMmQeIw1MY>

compared to the previously collected data distribution. However, less additional data is collected and the risk of collisions with obstacles is high.

After performing the first DAgger iteration, the accuracy ($1.0 - \hat{\tau}$) was compared according to τ . In this test, ε was set to 70, and α was 10. The result for this is shown at 4.1. As τ increased, the accuracy increased, and a value of 0.05

Table 4.1: Accuracy According to τ

	$\tau = 0.01$	0.025	0.05	0.1	0.15 ~
Accuracy	0.9734	0.9791	0.9873	0.9869	Collision in data collecting

showed the largest value. Even if τ was larger than 0.05, there was no significant performance improvement, and rather collisions occurred in the DAgger process of collecting additional data.

4.4.2.2 Ablation Study According to ε

The policy was trained with a dataset where different similarity thresholds ε : 30%, 50%, 70%, 90%, and 100%, and the proportions of data similar to $D_{(i=1)}$ in BC dataset D were 90%, 52%, 23%, 7%, and 0%; α was set to 10 in this test.

Table 4.2: Accuracy Comparison According State Similarity Threshold

		Test Dataset		
		<i>Accurately Trained State, $\hat{\tau}_t < \tau$</i>	<i>Inaccurately Trained State, $\hat{\tau}_t \geq \tau$</i>	<i>Entire State</i>
State Similarity Threshold ε (%)	30	0.9766 {5}	0.9614 {4}	0.9750 {5}
	50	0.9831 {4}	0.9775 {2}	0.9823 {4}
	70	0.9869 {3}	0.9844 {1}	0.9866 {1}
	90	0.9872 {2}	0.9631 {3}	0.9847 {2}
	100	0.9893 {1}	0.9413 {5}	0.9845 {3}

**Note.* The value in $\{\}$ is the ranking of the accuracy value in each column.

Table 4.2 shows results for the test dataset with different ε values. When $\hat{\tau}_t$ was updated for clearly similar states, such as $\varepsilon = 70\%$ (see Figure 4.4),

the accuracy increased for the inaccurately trained state and the entire state. When ε was 100, $\hat{\tau}_t$ was not updated to D like not using similarity, ε , and the accuracy of “*Accurately Trained State*” was the highest, but “*Inaccurately Trained State*” was the lowest. When $\varepsilon = 30\%$, $\hat{\tau}_t$ was updated for similar and not very similar states, the accuracy of which was low for all datasets.

4.4.2.3 Ablation Study According to α

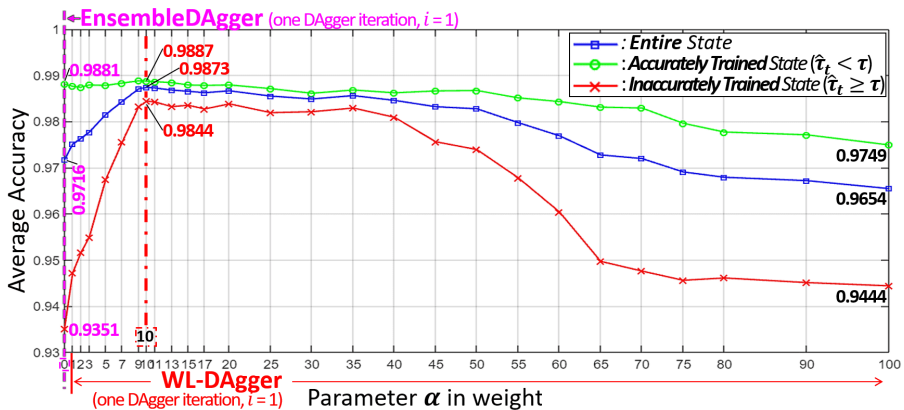


Figure 4.5: Test results of the average accuracy according to parameter α . α of 0 is the same case as the policy training of *EnsembleDagger*.

The policy was trained by 27 different α values (see x-axis of Figure 4.5) with the dataset obtained via *EnsembleDagger*. In this test, ε was set to 70. Figure 4.5 shows that the accuracies of the accurately trained state (green line) were less sensitive to α . For the inaccurately trained state (red line) and entire state (blue line), the accuracy increased with an increase in α from 0 to 10. Conversely, when α was between 10 and 40, the accuracy was slightly reduced. If α was excessively large (about 45 or greater), the accuracy was rather low, similar to when using the too-large learning rate of the optimizer in the policy training. The accuracy was the highest when α was 10.

4.4.2.4 Driving Test Results

Behavior Cloning		EnsembleDagger			with Weighted Loss Function (proposed)		
		After Dagger Iteration, $i=1$	$i=2$	$i=3$	After Only One Dagger Iteration, $i=1$		
Number of Data: 12043		13173	14039	14753	13173		
Training Time: 85 min.		175 min.	275 min.	385 min.	190 min.		
Acc. for Entire State, S_T : 0.9659		0.9716	0.9791	0.9854	α in (4) = 1	$\alpha=5$	$\alpha=10$
Inaccurately Trained, S_T : 0.9228		0.9351	0.9526	0.9809	0.9750	0.9814	0.9873
Average Collision (100 Laps)	Trained Env. (Fig. 3(a)): 4.0	3.48	1.68	-	2.00	0.34	-
	Untrained Env. (Fig. 3(b)): 3.0	2.00	2.00	1.00	2.00	1.00	-

Figure 4.6: Driving test results of *EnsembleDagger* with/without weight.

the green line is the vehicle trajectory; the red circle indicates the collision spot; The closer the circle is to the solid line, the higher the collision frequency is. To obtain the policy that the vehicle can be driven without collision, *EnsembleDagger* with the weighted loss function required only one DAGger iteration. However, *EnsembleDagger* without weighting needed three iterations.

Figure 4.6, Table 4.3, and video⁴ show the driving results with implemented policies trained through different algorithms: *BC*, *VanillaDagger* [71], *SafeDagger* [81], *HG-Dagger* [77], and *EnsembleDagger* [80]. The proposed DAGger algorithm was applied to each of these algorithms, and the average number of collisions encountered within 100 laps was measured. When the vehicle collided with obstacles, driving was resumed 3 m in front of the collision spot on a collision-free trajectory.

In trained and untrained environments, the number of collisions reduced with increased accuracy. The vehicle using a policy trained via DAGger variant algorithms with the weighted loss function was driven without collisions in a single DAGger execution, except for *VanillaDagger*. The *Dagger* with the

⁴<https://youtu.be/FeMckGLgQNA>

Table 4.3: Driving Test Results for *VanillaDagger*, *SafeDagger*, *HG-Dagger*, *EnsembleDagger* and **with Weighted loss function**

After DAGger Iteration, i	<i>VanillaDagger</i>		with weight		<i>SafeDagger</i>		with weight		<i>HG-Dagger</i>		with weight		Ensemble -DAGger		with weight	
	$i=1$	$i=2$	$i=1$	$i=2$	$i=1$	$i=3$	$i=1$	$i=1$	$i=1$	$i=2$	$i=1$	$i=1$	$i=3$	$i=3$	$i=1$	
Number of Data [ea]	18277	21076	18277	21108	13094	14453	13094		13625	14340	13625		14753	13173		
Training Time [min.]	183	285	203	305	174	382	188		177	280	197		385	190		
Network/Total (η_i)	0.49	0.76	0.49	0.76	0.85	0.91	0.85		0.88	0.94	0.86		0.94	0.69		
Accuracy	<i>Entire State</i>		0.973	0.975	0.979	0.983	0.970	0.984	0.985	0.976	0.983	0.986	0.985	0.987		
	for	Inacc. Trained	0.931	0.948	0.954	0.978	0.932	0.980	0.982	0.945	0.979	0.984	0.980	0.984		
Average	Trained Env.		3.50	2.00	1.50	-	3.34	-	-	2.50	-	-	-	-		
Collision	Untrained Env.		3.00	2.00	2.00	1.00	3.00	1.00	-	2.00	1.00	-	1.00	-		

**Note.* “with weight” means the DAGger with the weighted loss function (proposed). In this test, ε and α were set to 70% and 10. In “Training Time”, data collection time and training time were included, when GeForce RTX 2080 TI GPU was used to train the policy.

weighted loss function trained the policy with a higher weight in collision spots, particularly on roads with narrow or large curvature changes. *VanillaDagger*, *SafeDagger*, *HG-Dagger*, and *EnsembleDagger* required more human effort and time to collect additional data than *Dagger* with the weighted loss function.

4.4.3 Walking Robot Experiments

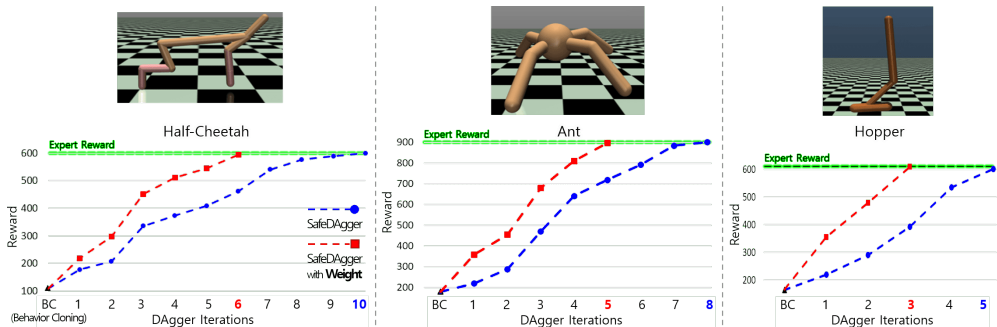


Figure 4.7: Half-Cheetah, Ant, and Hopper walking robots in MuJoCo.

In addition to autonomous driving, *WeightDagger* was applied and tested to Half-Cheetah, Ant, and Hopper, walking robots that exist in the MuJoCo simulator [92] (see Figure 4.7). The expert policy is obtained by reinforcement learning provided by RL Baselines Zoo [93], which is used to implement imi-

tation learning. The training network consists of three fully connected layers with 96 nodes using the Relu activation function. A fully connected layer that uses a linear activation function with nodes according to the dimension of each action, is connected at the end of the network. The epochs were set to 30, batch size to 64, learning rate to 0.001, and the Adam optimizer were used.

In this experiment, the effectiveness of *WeightDagger* is shown through comparison between *SafeDagger* and *SafeDagger* with weight algorithms. τ , which is the threshold difference between the expert and network actions in *SafeDagger*, was set to 0.075, 0.01, and 0.01 for Half-Cheetah, Ant, and Hopper, respectively. Besides, α in the weighted loss function (4.2) was set to 5, 20, and 15, respectively. These values are the best performing values among several candidate values. In this experiment, the weight update (*Step2*) method according to the state similarity was not applied. This is because it is difficult to obtain similarity because the state contains various contents (position, acceleration, angular acceleration, etc.).

Figure 4.7 shows results of *SafeDagger* and *SafeDagger* with weight algorithms (*WeightDagger*). The reward, which is the y-axis, has a higher value as the robot walks further, and is an average value over 10 tries. Both algorithms recorded a higher reward than behavior cloning (BC) as the *Dagger* was repeated. To achieve similar performance to the expert’s reward, *WeightDagger* required fewer *Dagger* iterations than *SafeDagger*.

4.5 Conclusion

In this chapter, a new *Dagger* training algorithm *WeightDagger* is proposed to reduce the data imbalance problem. The weight values are calculated via the action discrepancy between the trained policy and expert actions obtained

during *Dagger*. These are paired to the additional data sampled by *Dagger* as well as to the entire training dataset with a high state similarity. Thus, the policy is trained with a high learning rate for high-discrepancy data among the entire training dataset.

The effectiveness of *WeightDagger* was evaluated by conducting an autonomous driving experiment in parking lots with complex obstacles and walking robots in the MuJoCo simulation such as Half-Cheetah, Ant, and Hopper. The test results showed that a policy trained with the weight value more accurately imitated expert action in the same *Dagger* iteration than *Dagger* without the weight. Therefore, it achieved the desired policy with fewer *Dagger* iterations and less human effort for data collection. In future work, the performance improvement of the proposed algorithm will be further studied.

Chapter 5

DAGGER USING ADVERSARIAL AGENT POLICY FOR DYNAMIC SITUATIONS

5.1 Introduction

In order to deal with static as well as dynamic obstacles, motion planning methods using deep learning have been studied actively [24, 76, 94–107]. In these studies, policies of the dynamic obstacles are manually modeled to train the ego agent that avoids dynamic obstacles. There are studies modeling dynamic obstacles by depicting pedestrian movements in real-world environments [94–96]. To model dynamic vehicles in a general road, their behaviors in road environments such as highways, alleys, roundabouts, and unprotected intersections are depicted [24, 76, 97–99]. Multiple mobile robots move along paths with various geometric patterns to uniformly cover a specific space [100–102]. To implement the parking situation, pre-planned parking path is used, and the agent tracks this path [103–105]. In [106], dynamic situations involving pedestrians are modeled with simple geometric curves, and overtaking/lane-change/lane-keeping situations are modeled in [107].

However, manually modeling the policy of the dynamic obstacle has some limitations. The ego agent policy is trained only on a limited set of modeled dynamic situations, making it difficult and heuristic to train for various situations that do not overlap with previous training steps. Moreover, since the ego agent policy is difficult to be trained for environments more complex and difficult than the limited dynamic situations, the performance of this policy is difficult to be improved gradually.

In order to address the aforementioned limitations, an adversarial agent policy has been adopted to the ego agent policy’s training process without modeling the dynamic obstacle’s policy [108–113]. During the training process, the ego and adversarial agent policies are jointly trained to cope with the behavior of each policy, and this process is repeated. Thus, various dynamic situations can be generated. In addition, the performance of the adversarial and ego agent policies are gradually improved by considering each other’s trained policies.

The adversarial agent policy has been applied only to reinforcement learning (RL) [108–113] and has not been applied to imitation learning, especially the data aggregation (DAgger) algorithm. DAgger repeatedly performs imitation learning training process and obtains an agent policy with increasingly better performance [71]. There are advantages to using DAgger instead of RL to train the ego agent policy. First, DAgger can gather the training data more efficiently than RL, so convergence can be faster and applicable to the real-world. DAgger is more sample efficient than RL because it is trained to imitate expert behaviors that have succeeded in complex dynamic situations without trial-and-error [114]. Second, DAgger clearly generates training data to perform complex tasks such as avoiding dynamic and static obstacles together with expert behaviors without modeling a heuristic reward function.

Therefore, this chapter proposes a novel training framework that adopts the

adversarial agent policy to DAgger for ego agent policy training in dynamic situations. The proposed framework has the advantage of applying an adversarial agent policy to learning for dynamic situations as well as being able to train the ego agent policy with DAgger. Contributions of the proposed method are summarized as follows:

- To the best of our knowledge, this is the first study in which adversarial agent policy is applied to handle dynamic situations with DAgger.
- The proposed framework deal with dynamic situations that are more diverse and difficult than those of real-world by modeling the adversarial agent policy to compete with the ego agent policy.
- Applying the adversarial agent policy to DAgger reduces the number of DAgger training iterations by automatically acquiring non-redundant data that is not sufficiently trained in previous DAgger training steps.

The remainder of this section is structured as follows: Related works and the DAgger algorithm are reviewed in Section 5.2; Section 5.3 introduces the DAgger training framework using adversarial agent policy method; Section 5.4 analyses the proposed framework through navigation task experiments and Section 5.5 concludes the study.

5.2 Related Works & Background

5.2.1 Motion-planning Algorithms for Dynamic Situations

Existing model-based obstacle avoidance algorithms have been extended to obtain the ego agent policy in dynamic situations. The dynamic obstacle’s trajectory is reflected to the cost function of the dynamic window approach (DWA)

algorithm [63]. Studies using the model predictive controller (MPC) propose a cost function that includes bounds for dynamic obstacles [94, 99]. If an avoidance path planned by the search-based planner is not valid due to dynamic obstacles, this path is modified with the optimization-based path planner [105]. In [107], the velocity difference between the ego agent and the dynamic obstacle is reflected in the cost of the constrained iterative linear quadratic regulator (LQR). An avoidance strategy is determined using the learning method and is reflected in an optimization based collision avoidance (OBCA) algorithm [103]. However, these methods are difficult to deal with various situations other than those modeled in a specific situation. It can be improved through complex modeling and parameter tuning, but this is also limited.

Using learning methods, the ego agent policy can be trained for a variety of dynamic situations. There are studies applying the tree expansion process of the monte-carlo tree search (MCTS) algorithm to cope with various dynamic situations by training it with RL [100, 104]. Through RL training, one of the safe path is selected according to the speed prediction of the oncoming vehicle obtained from the neural network [97]. A sub-goal considering obstacles and global path is acquired from the network, and dynamic obstacles are considered while tracking the sub-goal using MPC [102]. In [101], LSTM-based RL method copes with multiple dynamic obstacles by adding a reward that maximizes the distance from dynamic obstacles. A method using generative adversarial imitation learning (GAIL) calculates the desired direction and speed of the mobile robot by predicting the social force according to pedestrian gait images [95]. In [106], whether to perform re-planning is trained to handle passing, following, and crossing situations between a robot and a pedestrian by imitation learning. Although these methods can be trained for various situations, this requires modeling different dynamic obstacle policies to obtain data.

Without modeling the dynamic obstacle policy, there are studies using an adversarial agent policy for dynamic situations. Through this policy, not only training data for various situations can be obtained through interaction with the ego agent, but also the performance of the ego agent policy can be improved gradually. A study conducting multiple mobile robot training on cooperation and competition tasks transforms a reward function structure to use adversarial policies in the RL process of GAIL [115]. In a multi-agent task with competing policies such as games, each policy is modeled as an adversarial agent policy and trained by RL to enhance each other’s performance [108, 109]. Adversarial agents are trained with RL to collide with the autonomous vehicle to find cases of its failures [110]. In a study of applying RL to autonomous driving at intersections, failure cases are found by training an adversarial driving agent, and robustness of autonomous vehicles is improved through re-training [111]. Through RL, avoidance situations of the unmanned aerial vehicle and adversarial attack situations against them are handled [112]. The robot arm is trained to catch a moving object with RL, and the object is modeled as an adversarial policy and is trained not to get caught [113].

5.2.2 DAgger Algorithm for Dynamic Situation

Algorithm 5 represents a basic structure of the DAgger algorithm. π_{BC} is an ego agent policy trained by behavior cloning (BC), and D_{BC} is BC dataset. BC is an algorithm in which imitation learning is performed only once. These are used to initialize π_1 and D (line 2 in Algorithm 5).

A data-sampling function (line 4, the original version) takes an expert policy (π_{exp}), a trained ego agent policy (π_i), and a modeled dynamic obstacle policy (π_{dyn}), and returns an additional dataset D_i , where i is a DAgger iteration. If π_{dyn} is not used, the ego agent policy is to be trained in a static

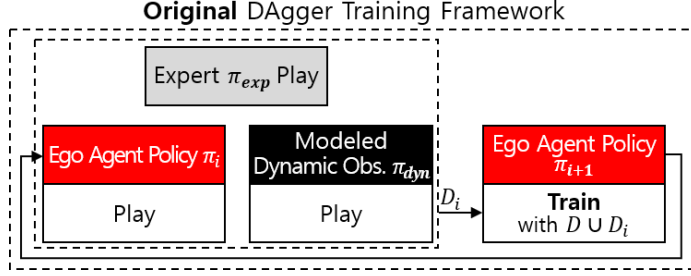


Figure 5.1: Original DAgger for dynamic situations

Algorithm 5: DAgger Algorithm for Dynamic Situations

*Note The **blue** font is the proposed DAgger algorithm.

```

1 function DAgger()
2 Initialize  $\pi_1 \leftarrow \pi_{BC}$ ,  $D \leftarrow D_{BC}$  // BC means behavior cloning
3 for  $i = 1$  to  $N$  do // Do Until Obtaining Desired Policy
4    $D_i \leftarrow$  Data-Sampling Function( $\pi_{exp}$ ,  $\pi_i$ ,  $\pi_{dyn}$ ) // Original
   Version
5    $D_i \leftarrow$  Data-Sampling Function( $\pi_{exp}$ ,  $\pi_i$ ,  $\pi_{adv_i}$ ) // Proposed
   Version, replace line 4
6   Aggregate data  $D \leftarrow D \cup D_i$ 
7   Train ego agent policy  $\pi_{i+1}$  on  $D$ 
8   Train adversarial agent policy  $\pi_{adv_{i+1}}$  // Added by proposed
   algorithm
9 return  $\pi_i$ 

```

environment. As shown in Figure 5.1, π_{dyn} does not improve in the repeated DAgger training loop. In the data-sampling function, π_{exp} , π_i , and π_{dyn} are executed simultaneously (see left three boxes in Figure 5.1). While executing, if an action discrepancy between the actions of π_{exp} and π_i is greater than the threshold, the robot is controlled by the expert action and D_i is collected [81].

The additional dataset D_i is aggregated with the existing dataset D (line 6), which is used for training π_{i+1} (lines 7). These steps are repeated N times until a desired policy is obtained (line 3 and 9).

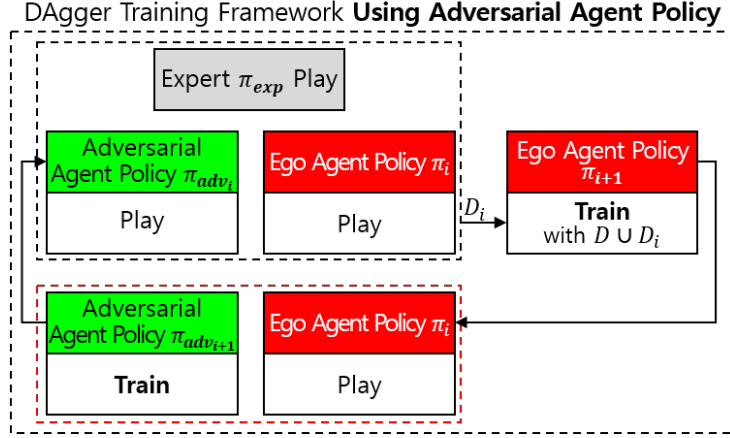


Figure 5.2: Proposed DAGger for dynamic situations

5.3 Proposed Method

5.3.1 DAGger Training Framework Using Adversarial Agent Policy

The proposed framework adds the adversarial agent policy (π_{adv_i}) to the DAGger training process to generate data for the ego agent policy training. A schematic diagram of the proposed DAGger training framework is shown in Figure 5.2. The modeled dynamic obstacle policy π_{dyn} used in the original DAGger for dynamic situations is replaced by π_{adv_i} (see line 5 in Algorithm 5). The process of training π_{adv_i} is shown in the red box of Figure 5.2 and indicated in the line 8 of Algorithm 5.

The adversarial agent policy π_{adv_i} is configured to take an action that competes with the ego agent policy π_i and is trained through RL. For example, if π_i behaves to avoid dynamic obstacles, π_{adv_i} acts to block the ego agent (π_i). This π_{adv_i} does not actually exist in the real-world, and it is proposed to generate data for π_i training. Thus, π_i can be trained for more difficult situations than

dynamic situations that could happen in the real-world.

Figure 5.3 shows the detailed training loop process. In the first step of the DAgger training loop ($i = 0$), the ego agent policy π_{BC} is trained with BC in a static environment because there is no trained π_{adv_i} . In the subsequent process, π_i is trained with a play of π_{adv_i} , and then $\pi_{adv_{i+1}}$ is trained by playing trained π_i . During this process, π_i is trained to perform better against π_{adv_i} , and $\pi_{adv_{i+1}}$ is also trained to perform better on π_i . The performance of π_i and π_{adv_i} improves to outperform each other over and over again, and $\pi_{i=N}$ obtained by performing N training times finally shows close to the best performance.

Among training data samples that can be generated for π_i training in dynamic situations, the distribution of practically valid data samples is partial. This is depicted in the black dotted box of Figure 5.3. For example, in dynamic situations of autonomous driving, valid data samples include situations in which ego vehicle’s future driving trajectory is overlapped with that of dynamic obstacle while having various directions and speed relationships. However, it is not easy to obtain such samples by modeling π_{adv_i} manually or with the traditional methods. On the other hand, the proposed framework that trains π_i using π_{adv_i} in DAgger can generate the valid data according to following two factors.

First, the proposed π_{adv_i} automatically generates valid training data samples by considering the updated π_i performance in repeated DAgger training steps without manual modeling. At each DAgger training step, additional data may be collected with a distribution different from that of the previously collected data. Since π_i can be trained on situations that have not been sufficiently covered in previous DAgger training steps, only the data necessary for training are gradually aggregated into the DAgger training dataset. Thus, the number of DAgger iterations can be reduced without wasted training steps.

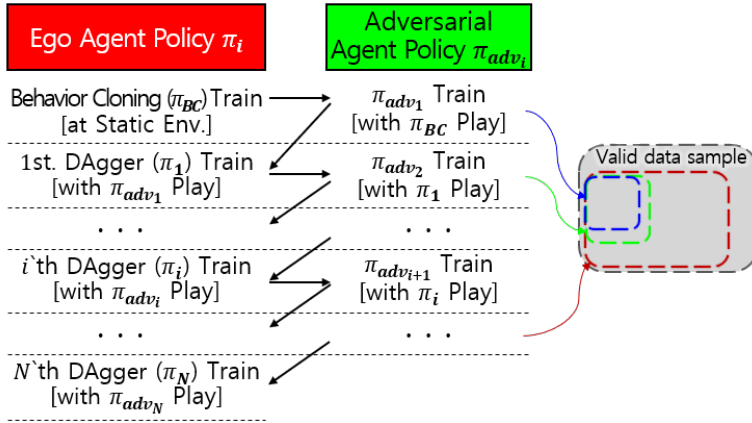


Figure 5.3: Detail proposed training framework and its effect

Second, the training process of the proposed framework is similar to the curriculum learning process [116], which can help π_i to be trained better through situations that gradually illustrate increasingly complex tasks. In the initial course of DAgger training, π_i begins to be trained about the easy π_{adv_i} that has not been learned sufficiently. As the DAgger training process progresses, π_i is trained for increasingly difficult situations with π_{adv_i} that has been improved by the training process. This increases the learning efficiency and robustness of the training, which can speed up the convergence rate of π_i training and help generalization [116].

5.3.2 Applying to Oncoming Dynamic Obstacle Avoidance Task

In this chapter, the proposed training framework is applied to a dynamic obstacle avoidance task in a semi-structured environments of autonomous driving. As shown in Figure 5.4, the ego agent policy (π_i) and the adversarial agent policy (π_{adv_i}) drive from opposite side on a road with no lanes and a width of about three cars, such as a parking lot or an alley. π_i behaves to avoid a dynamic obstacle without colliding with static obstacles. The dynamic obstacle agent acts

to block the driving of π_i and consists of π_{adv_i} . Both agents have autonomous driving systems that use only in-vehicle sensors without global information and predicted trajectory of a dynamic obstacle.

5.3.2.1 Ego Agent Policy

The ego agent policy π_i is trained using the DAgger algorithm. π_i is approximated as a convolutional recurrent neural network (CRNN). An input (state) of π_i consists of a set of 10 sequential occupancy grid maps (OGMs). This map considers obstacles and lanes as occupied grids, and the ground to be unoccupied. The area of the future trajectory of the dynamic obstacle is additionally reflected as occupied on this map. The action of the ego agent policy is the look-ahead point [70], which is the point the ego agent will reach, and the steering angle and velocity are obtained through the pure pursuit algorithm [48].

The expert policy (π_{exp}) for collecting BC and DAgger training data consists of a forward simulation algorithm, not a human-in-the-loop design that human must intervene during DAgger training. This is because, in repeated DAgger iterations, this algorithm can obtain state-action relationships in a consistent pattern. However, humans are heuristic by manually labeling data in human-in-the-loop. In addition, the DAgger execution process can be configured automatically as human intervention is not required. Although the process of obtaining data with this algorithm requires a lot of computation time, the output of the network trained on this data can be computed in real-time.

Through the forward simulation, look-ahead points (L_{free}) in unoccupied grids of OGM (state) are searched:

$$L_{free} \subset L_{OGM}, \quad (5.1)$$

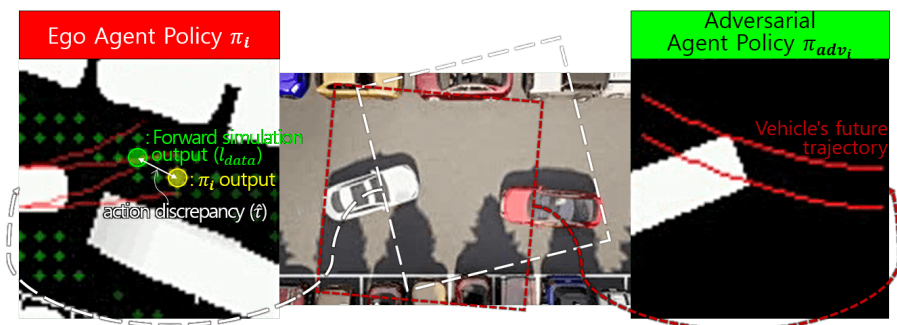


Figure 5.4: Proposed training framework is applied to dynamic situation of autonomous driving. The ego agent (white, left) avoids static and dynamic (adversarial agent) obstacles. The adversarial agent (red, right) blocks the ego agent’s driving.

where L_{OGM} is a set of look-ahead points of all grids existing in OGM. The cost of each look-ahead point $l_{free} (\in L_{free})$ is calculated through the objective function, $O(OGM, l_{free})$. Among L_{free} , one look-ahead point having the largest objective value is obtained and is denoted as l_{data} . During the DAgger execution, if the action discrepancy ($\hat{\tau}$) between l_{data} and π_i output is greater than a threshold (τ), l_{data} is labeled as the action and collected as additional data (D_i) along with the state (see line 5 in Algorithm 5).

The objective function to obtain l_{data} is as follows:

$$\begin{aligned}
 O(OGM, l_{free}) = & \alpha \times FreeTraj(OGM, l_{free}) \\
 & + \beta \times Dist_{Long}(OGM, l_{free}) \\
 & + \gamma \times Free_{Lat}(OGM, l_{free}),
 \end{aligned} \tag{5.2}$$

where α, β, γ are gains for each sub-objective functions. $FreeTraj$ returns a ratio of an unoccupied area for an area around the ego agent’s future driving trajectory. This trajectory is obtained using the kinematic bicycle model according to the steering angle and velocity from the center of the vehicle’s rear wheel

to l_{free} . $Dist_{Long}$ calculates a longitudinal distance between the ego agent and l_{free} . This distance is normalized by dividing by the maximum longitudinal distance of OGM. $Free_{Lat}$ obtains a ratio of unoccupied region around both sides of the vertical direction from l_{free} to the end of the OGM. Here, the vertical direction is perpendicular to l_{free} from the ego agent. All sub-objective functions have values between 0 and 1.

5.3.2.2 Adversarial Agent Policy

The objective of the adversarial agent policy (π_{adv_i}) is to block the ego agent, and π_{adv_i} is trained by RL. Either DAgger and RL can be used. Since π_{adv_i} is trained to perform a simple task, RL can also achieve sufficient performance. Training through random actions with a simple reward function in RL is more appropriate than defining an optimal expert policy π_{adv_i} for DAgger. π_{adv_i} is trained with the recurrent proximal policy optimization (PPO) algorithm in RL [117].

The state of π_{adv_i} is OGM, and static obstacles are not reflected in the state and the reward function. This is because when π_{adv_i} faces a dynamic obstacle, it rarely collides with the static obstacles, and it is difficult to train π_{adv_i} to avoid the static obstacles while blocking π_i . Besides, unlike the π_i state, the future trajectory of the dynamic obstacle is not reflected in the π_{adv_i} state. The reason for this is to balance the training speed between π_{adv_i} and π_i , so that π_{adv_i} does not diverge. This is because the behavior of π_{adv_i} , which blocks π_i , is easier than that of π_i , and π_{adv_i} is trained faster than π_i . The reward function

settings are as follows:

$$R = \begin{cases} + \frac{d_{max} - d}{d_{max}}, & \text{To get close to ego agent,} \\ - \frac{t}{t_{max}}, & \text{To get close quickly,} \end{cases} \quad (5.3)$$

where d is the distance between the ego agent and the adversarial agent, and d_{max} is the maximum distance that d can have. t is the time step performed in each training episode and t_{max} is the maximum value of t . The action is the steering angle and velocity.

5.4 Experiments

The proposed DAgger training framework using adversarial agent policy method was applied to autonomous driving with dynamic environments, and the performance was analyzed.

5.4.1 Experimental Setup

The proposed method was developed and tested in simulation. Autonomous driving environments including a parking lot were built by Unreal Engine 4. A CARLA simulator [90] was used to collect training data and to implement the policy. To get accurate state (OGM) without occlusion, the camera was positioned at x: 5.7 m (forward), y: 0.0 m (left), and z: 5.7 m (upper) relative to the vehicle center, and the pitch angle was -90° . The size of OGM was 80×80 , and each grid size was 0.125 m. The velocity range was -3.0 to 7.0 km/h.

5.4.1.1 Ego Agent Policy Training

The ego agent policy π_i consisted of convolutional networks with 32 and 64 channels, and the long short-term memory (LSTM) layer with size 16 was connected to them. The flattened and fully connected layers with 32 nodes were connected to these layers with 25 % dropout. The action was predicted by connecting a fully connected layer with 2 nodes. The Adam optimizer with a 10^{-5} learning rate was used. The epochs were 10 k, and the batch size was set to 512.

The gain hyper-parameters α, β, γ of the objective function $O(OGM, l_{free})$ (5.2) were set to 10.0, 1.0, 1.0. If the value of $O(OGM, l_{free})$ was calculated to be less than 9.5, it might be experimentally regarded that a sufficiently safe driving was impossible, and l_{free} data for this case was not collected. Then, the steering angle was fixed at 0° and the vehicle drove backward with -3.0 km/h. Also, if a collision occurred due to a look-ahead point l_{data} obtained through $O(OGM, l_{free})$, data from the starting point to the collision was not collected. When 9 seconds have elapsed since the start of driving, the ego agent was considered to be stuck, and data was not collected. If the distance between the ego agent and the adversarial agent was less than 5.5 m, the adversarial agent was stopped to avoid collision caused by itself.

The starting position of the ego agent was randomly determined in the range of y axis: ± 2.3 m and yaw: $\pm 15^\circ$ with respect to the leftmost-centered axis (see left side of Figure 5.6). While driving, if the x position of ego agent exceeded 20 m, passed an adversarial agent, or a collision occurred, the ego agent was respawned. A total of 20 executions of the above process was one DAgger training iteration.

5.4.1.2 Adversarial Agent Policy Training

The Stable Baselines3-contrib library [118] was used for the adversarial agent policy π_{adv_i} training. The recurrent PPO algorithm was implemented using the RecurrentPPO-CnnLstmPolicy function provided in this library. In the reward function (5.3), the maximum distance between the two agents (d_{max}) was set to 20 m, and the episode length (t_{max}) was set to 128 frame. batch size was 128 and epoch was 10.

The starting position of the adversarial agent was 20 m of x, and y and yaw had random values within ± 2.3 m and $\pm 15^\circ$ (see axis in left side of Figure 5.6). If x position of the adversarial agent was less than 0 m, or if it passed the ego agent, or a collision occurred, this agent was respawned to the starting point. A total of 100,000 frame times of the above process was one π_{adv_i} training execution process. During training, the policy with the maximum reward in an episode was saved and used in the next training iteration. The reward was the accumulated value of frames until t_{max} , occurring collision, or passing π_i .

5.4.2 Experimental Result

In the proposed framework, 10 iterations of DAgger and RL training were performed ($N = 10$). This value was sufficient to check the performance change of the ego and adversarial agent policies as training was repeated.

5.4.2.1 Performance of Adversarial Agent Policy

The performance of the trained adversarial agent policies were tested to check if they were improved with the proposed training framework. As training progressed (i increases), π_{adv_i} was trained by playing the ego agent policy π_{i-1} .

Here, π_{adv_i} is adversarial agent policies trained with RL. After training ends (after N becomes 10), each trained π_{adv_i} was played 100 times with $\pi_{i=10}$, and the number of times that π_{adv_i} blocked the driving of π_{10} was counted.

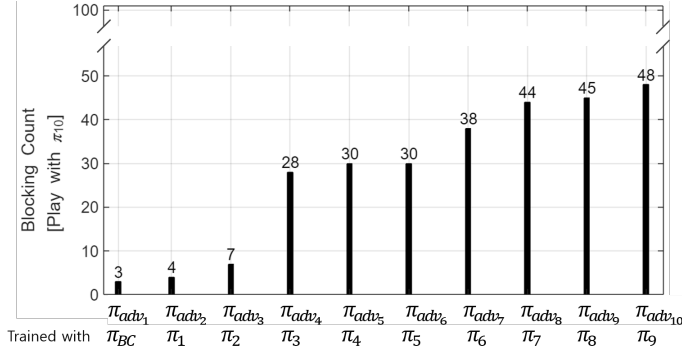


Figure 5.5: Adversarial agent policy that gradually improves as training progresses

As i increased, the number of times π_{adv_i} blocked the driving of π_{10} increased (see Figure 5.5). These results demonstrated that the performance of π_{adv_i} increased gradually as training was performed.

5.4.2.2 Ego Agent Policy Performance Comparisons Trained with / without Adversarial Agent Policy

In DAgger training, the performance of the ego agent policies trained with and without π_{adv_i} was compared. In the original method, as the DAgger training progresses, π_i was trained by playing $\pi_{adv_{i=1}}$ which was trained only once without further performance improvement. The proposed method trains the ego agent policy π_i by playing π_{adv_i} trained in the i learning step. After training is over, each π_i was played with $\pi_{adv_{i=10}}$ 100 times, and the number of times π_i avoided against driving of $\pi_{adv_{10}}$ was counted. Here, $\pi_{adv_{10}}$ showed the best blocking performance for π_i .

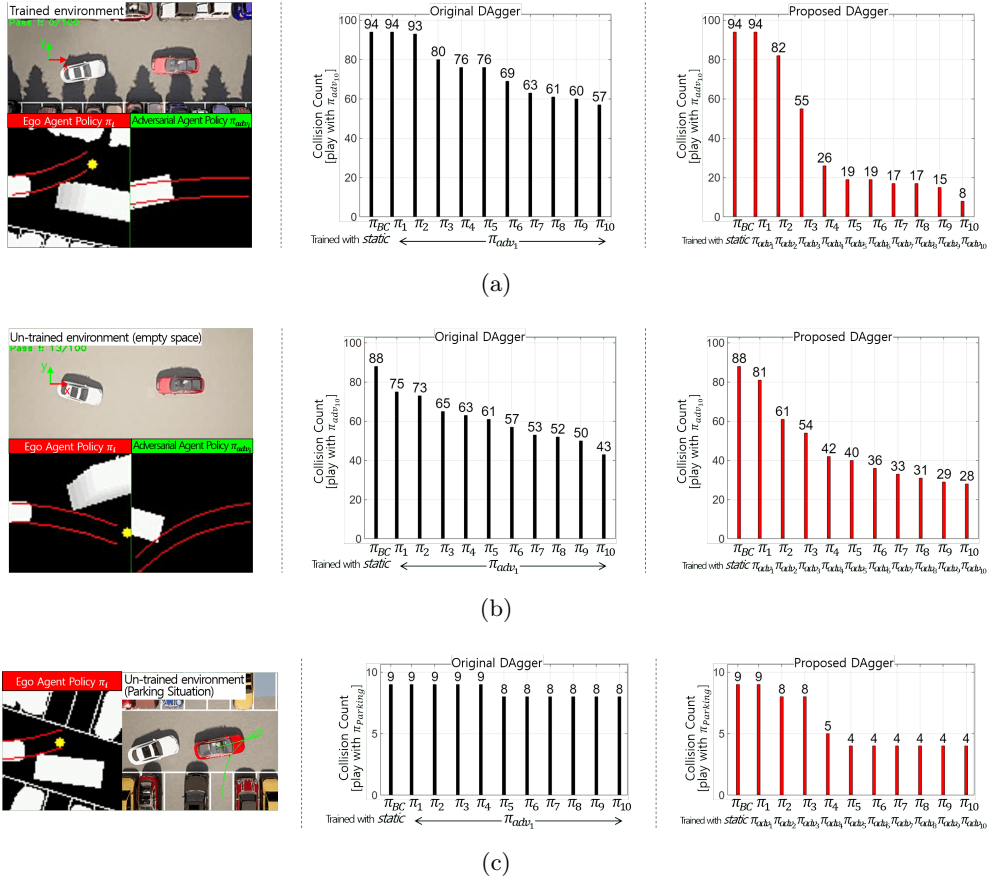


Figure 5.6: Comparison of trained ego agent policy performance with and without adversarial agent policy. The ego agent policy trained at each DAGger training step was played against $\pi_{adv_{i=10}}$ (a-b) or $\pi_{parking}$ (c), and the number of collisions was counted. (a): trained environment (parking lot), (b): un-trained environment (empty space), (c): un-trained environment (parking situation).

The test result is shown in Figure 5.6. In both the original and proposed DAGger methods, as i increased, the number of times that π_i avoided the driving of $\pi_{adv_{i=10}}$ decreased. π_i trained with the proposed method (middle plots with red bar in Figure 5.6) showed a lower number of collisions than the original method (left plots with black bar in Figure 5.6). This could be seen both in

the trained environment (see Figure 5.6(a)¹) as well as in the untrained environment (see Figure 5.6(b)² and 5.6(c)³). In particular, although the parking situation (Figure 5.6(c)) was quite different from the trained environment, the proposed method showed fewer collisions. This is because the adversarial agent policy acted the extreme behavior of blocking the ego agent, and the ego agent had learned to cope with it.

5.5 Conclusion

This chapter proposes the DAgger training framework that uses the adversarial agent policy to deal with dynamic environments. The proposed adversarial agent policy competes with the ego agent and is trained to block the driving of the ego agent. Therefore, training data for various situations in which the ego agent was not trained in the previous DAgger steps can be automatically generated without the need to model dynamic situations. Besides, the ego agent can be trained gradually from easy to difficult situations. In the autonomous driving experiments in dynamic environments, the performance of the ego agent and adversarial agent policy was gradually improved. The proposed method shows a lower number of collisions than a method trained without using the adversarial agent policy in trained and untrained environments. In the future, a sim-to-real method will be studied to apply the policy trained by our method in the simulation to the real environment.

¹Result of Figure 5.6(a): <https://youtu.be/0EbXnJts88>

²Result of Figure 5.6(b): https://youtu.be/e_7JpmM0eFs

³Result of Figure 5.6(c): <https://youtu.be/LF5nrRQREoI>

Chapter 6

CONCLUSIONS

This thesis involves autonomous driving technologies for semi-structured environments with static and dynamic obstacles such as parking lots or alleyways. In such environment, the drivable area of width and curvature vary with no lanes, and obstacles are unknown in advance and existed in various forms. The proposed methods deal with research on learning-based navigation that can ensure safety through enough training and practical application in real environments. These methods use vision sensor and a simple topological map constituting intersections and roads, and do not use the expensive HD-map and localization data that may be inaccurate in the semi-structured environment.

Specifically, the proposed multi-task network obtains an input data for motion-planning algorithms that do not consider intersections. It detects not only the drivable area but also the branch roads at an intersection only with vision data, and the drivable area with one branch road is obtained according to the navigation information. The proposed method obtained higher accuracy than the model-based branch road recognition method in actual parking lots, and successful navigation was possible. To drive towards the drivable area, im-

imitation learning is used, and the proposed imitation learning method selects the look-ahead point on the drivable area. Even when the vehicle is controlled with a trained network action (even in human-loop-loop design), a human expert can easily select the optimal action without a separate joystick device, so DAgger can be applied to actual autonomous driving. As a result of testing in actual parking lots where the width and curvature of the drivable area change significantly and shadows exist, the vehicle using DAgger with look-ahead point navigated more safely than the model-based motion planning algorithm. In addition, a WeightDAgger algorithm is proposed to more accurately imitate the expert behavior in unsafe or near-collision situations. The proposed algorithm can train the policy with a high learning rate on data having low accuracy. By applying WeightDAgger to the existing DAgger algorithm, a policy similar to that of the expert could be obtained with fewer DAgger iterations and less human effort. In order to apply such algorithms to dynamic environments, a DAgger training framework is proposed by modeling dynamic obstacles as the adversarial policy agent competing with the agent. Both agents are trained to perform progressively higher, the agent avoids various and increasingly difficult dynamic situations. In untrained parking and oncoming driving situations, the proposed method showed a lower collision rate than DAgger without using the adversarial policy agent.

The proposed methods are used when the autonomous vehicle enters a semi-structured environment with the topological map without lanes, after driving with existing commercialized technology in a structured environment with lanes. Although these methods focus on the semi-structured environment, they can be applied to any autonomous driving system that has an input with a form of the 2D occupancy grid map in any environment, and can also be used to any vehicles and various mobile robots. However, if the accuracy of the occupancy grid map is low or the vehicle drives at un-trained environments, the proposed

methods may not handle well and danger situations will be occurred. Thus, further studies to address these problems are needed to propose a repetitive training framework by automatically collecting additional perception and driving data for various real environments. Furthermore, by generating additional training data with a small distribution from the data collected while performing DAgger, human effort required for data collection can be reduced.

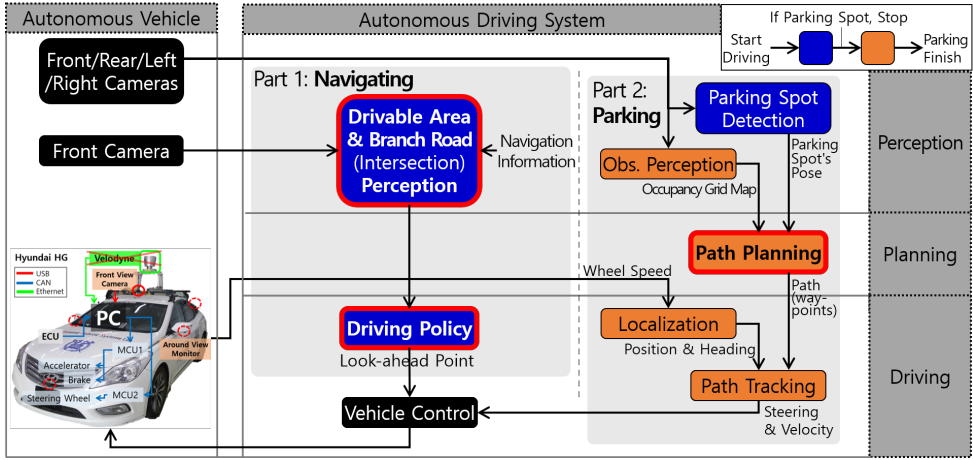
Appendix A

FAST AND ACCURATE VISION-BASED AUTONOMOUS PARKING

This appendix introduces methods for performing autonomous parking using only vision. The overall system is described in Sec. A.1. In particular, methods for performing accurate parking (Sec. A.1.1 and Sec. A.1.2) and for quickly generating a parking path (Sec. A.2) are explained in detail.

A.1 Vision-based Re-plannable Autonomous Parking System

The system architecture of the vision-based autonomous parking system is shown in Figure A.1. Cameras are mounted on the front, rear, left, and right sides of the vehicle. These are converted into a bird's-eye-view image with the distortion correction using the Scaramuzza library [119] and integrated into one image. This image is used to recognize the parking spot and distinguish between drivable/non-drivable areas. The position and direction of the parking spots are detected by three methods, which are described in Sec. A.1.1. The drivable/non-drivable area perception is used for collision-free path generation. The parking path planner generates a path avoiding obstacles while considering



(a)

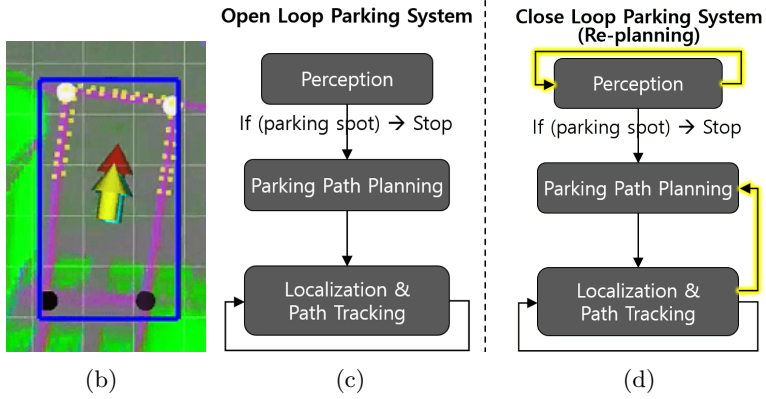


Figure A.1: Vision-based autonomous parking system.

(a) Vision-based parking spot detection methods, (b) Open-loop parking system, (c) close loop parking system (re-planning).

non-holonomic constraints. A detailed explanation of a method to quickly plan the path is introduced in Sec. A.2. The localization data, which is the position and direction of the vehicle relative to the parking path, is used to track the parking path. The dead reckoning method is used as odometry to obtain localization data using 4 wheel speed. The Kanayama algorithm [120] for tracking the parking path by reducing the relative position and direction errors between paths simultaneously.

A.1.1 Parking Spot Detection

Three parking spot detection methods are developed, and their advantages and disadvantages are analyzed. The first method is the bounding box detection method using the YOLO network [46] (see the blue box and the red arrow in Figure A.1(b)). A rectangle containing the lines of the parking area is trained as a bounding box. The center of this box becomes the position of the parking spot. To find the direction of the parking spot, the longest line inside the box is detected as the Houghline transform function.

The second method is to use corner points [121]. The point where the parallel/vertical lines of the parking spot intersect is the corner point. This point is a training method by labeling this point with a bounding box. It is classified into two points: near ('OUT', the white points) and far ('IN' the black points). If two or more points are recognized among these points, the pose of the parking spot can be obtained according to a positional relationship between the vehicle and points. This is indicated in the light blue arrow Figure A.1(b).

Depending on the result of segmenting the parking line, the corner point may not be recognized accurately. To deal with this, a shape that is the set of yellow dots in Figure A.1(b) is defined. The position and direction most matching between this shape and the parking line are searched greedily. To save time, only the parking lines that exist between corner points are searched. The pose of the parking spot is obtained according to the position and direction of the vehicle and shape, as shown in the yellow arrow in Figure A.1(b).

The advantages and disadvantages of each method are shown in Table A.1. Overall, the second method using the corner point had relatively high accuracy for overall parking situations.

Table A.1: Comparison of Parking Spot Detection Methods

	pros	cons
Bounding Box	90 % < accuracy	- < 92 % accuracy - Reverse parking X
Corner Points	Accuracy: 95 % \pm 2	< 97 % accuracy
Shape Matching	Perfect perception result \rightarrow 100%	- Not real-time - Accuracy: 93 % \pm 7

*Note Accuracy is a qualitatively assessed value.

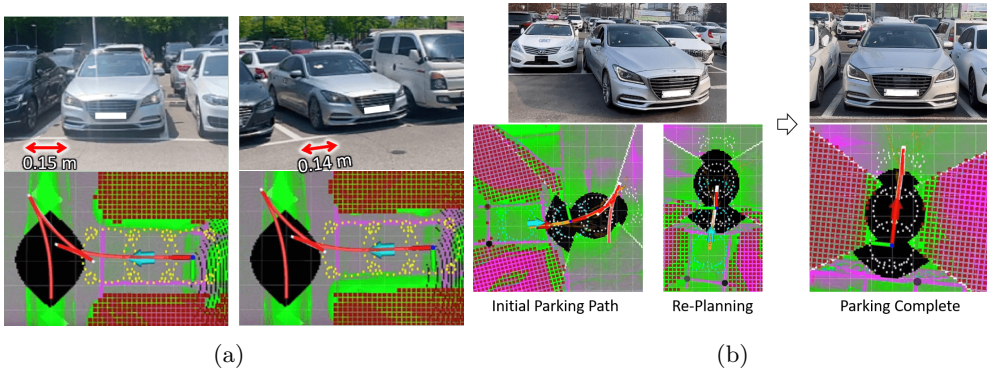


Figure A.2: Results with (a) and without (b) re-planning method.

A.1.2 Re-planning Method

As shown in Figure A.1(c), the open-loop parking system generates and tracks the parking path only once after detecting the parking spot, which has several limitations. It is difficult to accurately detect a parking spot because correcting the image distortion and segmenting parking lines are not precise (see Figure A.1(b)). Thus, performing accurate parking is impossible. In addition, when localization is conducted by dead reckoning, errors occur because the vehicle model cannot be accurately reflected and external factors such as friction arise. In this case, even if there is no tracking error, it is difficult to accurately perform parking. A position error of 0.07 to 0.15 m occurred between the center of the parking spot and the vehicle (see Figure A.2(a)).

To deal with this, the re-planning method of the parking path is proposed, after stopping in a specific situation while tracking the initially planned path. Here, there are 4 specific situations: i) At the forward/backward switching point (the moment the vehicle is stopped); ii) Moments of increased risk of collision; iii) Moment the tracking error becomes large; iv) The vehicle gets closer to the parking spot (only done once).

Even if the above conditions are satisfied, there are exceptions in which re-planning is not performed: i) If the parking point is not detected (the parking path cannot be generated); ii) If it is detected but far from the parking point (due to the presence of detection error); iii) When the difference in distance between the initially recognized parking spot and the newly detected parking spot is less than 0.05 m (no need to plan a new parking path);

Before re-planning the path, the parking spot is re-recognized and the localization is reset. Accordingly, it is possible to detect the parking spot at a location close to it and reduce the error of initial detection at a distant point. In addition, the dead reckoning error can be reset, so the accumulated localization error can be removed. The results are as shown in Figure A.2(b). The large sedan (Genesis, G80) reached the parking spot without collision during 10 tries, and the distance error was within about 0.05 m.

A.2 Biased Target-tree* with RRT* Algorithm for Fast Parking Path Planning

A.2.1 Introduction

For autonomous parking, it is necessary to plan a collision-free path taking into account a vehicle's non-holonomic constraint. As a method of parking path planning, researches have been conducted in the following categories; an optimization-based method for searching a path that satisfies an objective function and constraints [122]; a grid search-based method that extends a tree into a grid, such as the hybrid-A* algorithm [123]; a sampling-based method that extends a tree with randomly selected samples.

Among these methods, RRT* [124], the representative algorithm of the sampling-based method, has been used for parking [59, 125–128]. This is because it can plan a path that is closer to the shortest path faster than other methods, even in complex parking situations. Besides, it is possible to find the path relatively quickly even when many forward/reverse switching are required due to a large minimum turning radius and a narrow parking environment. The path can be searched with a small number of tree expansions without searching for all path sets or all branches for the tree extension in all grids. Nevertheless, if the search space or the parking goal is tight, RRT* may take large planning time.

A target-tree algorithm [126] is used together with RRT* to further reduce the planning time compared to using RRT* alone. It pre-generates a set of the backward paths around the parking spot, called a target-tree. At the initial pose, a path is extended through RRT*, and this extension is stopped when it reaches the target-tree. Hence, a search range of the RRT* path is reduced to the target-

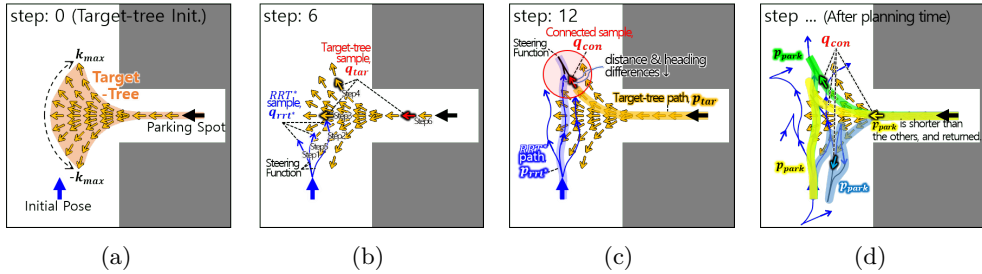


Figure A.3: Process and limitations of target-tree* algorithm with RRT*.

(a) The target-tree is generated at the initial step. (b) The RRT* path is extended in the entire map, and the target-tree samples q_{tar} are sampled within the target-tree. (c) The RRT* path is reached to the target-tree through the connected random sample q_{con} . The RRT* path p_{rrt^*} and the target-tree path p_{tar} are aggregated as the parking path, p_{park} . (d) After the planning time, several parking paths are generated, and the shortest parking path is returned.

tree rather than the parking spot. Furthermore, the target-tree* algorithm with RRT* [129] has been proposed in our previous study to increase the probability of obtaining a shorter parking path than the target-tree algorithm [126]. It searches multiple paths connecting the RRT* path and the target-tree, and selects the shortest one. Given sufficient planning time, the shortest parking path can be found.

However, if the planning time is not sufficient, the target-tree* algorithm [129] may be difficult to plan the shortest path. This is because a sample connected between the RRT* path and the target-tree is randomly searched from the entire range of the target-tree. Due to this randomness, the connected random sample can be selected far from the shortest parking path. Besides, the RRT* path length ratio may be high in the entire parking path. Moreover, the percentage of RRT* path lengths in total parking paths can be high. This reduces the effect of using the target-tree, and the planning time is increased.

To find the shortest path faster than the target-tree* algorithm [129], this paper proposes a biased target-tree* algorithm that additionally defines a biased range over which the connected random sample is mainly searched through deep learning. This range is around of an optimal connected sample where the shortest parking path can be planned as fast as possible. Thus, the probability of obtaining the shortest path becomes higher than that of the connected random samples are searched in the entire target-tree range. This network is trained with an occupancy grid map as the input data and the optimal connected random sample’s location as the label data.

A.2.2 Proposed Method

The proposed algorithm, the biased target-tree*, searches the shortest parking path faster than the target-tree* algorithm [129] by adding the data-driven method to the target-tree* algorithm. It can also further reduce the deviation of the path length, and increase the probability of searching the shortest path to reach the parking spot.

The biased target-tree* algorithm defines a biased region in which the target-tree sample q_{tar} is sampled with a high probability. This range is around the shortest (optimal) parking path. With this biased range, q_{tar} is not sampled randomly over the entire range of the target-tree, but is randomly sampled closer to this biased range with a higher probability. The biased range is obtained once after the target-tree initialization, and this range is used when sampling q_{tar} at each step of extending the RRT* path such as Figure A.3(b) and Figure A.3(c). Thus, the connected random sample q_{con} between the RRT* path and the target-tree is also searched close to the biased range.

In order to obtain the biased range, a deep neural network is used, and it is

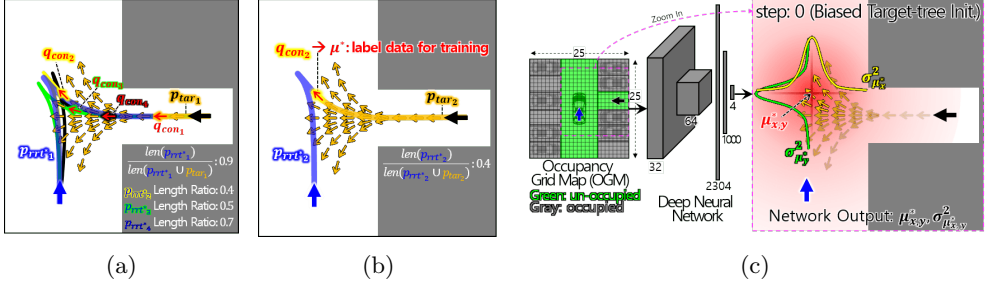


Figure A.4: Process of collecting label data of network, μ^* ; and process of biased target-tree* algorithm.

(a) The target-tree* algorithm with RRT* is conducted with enough time and iterations, and searches for paths that are close to the shortest path. According to the connected random samples, q_{con} , length ratios of the RRT* path to the total parking path are different. (b) To minimize RRT* expansion, the location of q_{con} in a path having the smallest length ratio (0.4) of the RRT* path becomes μ^* . (c) The deep neural network takes the occupancy grid map and returns the biased range's (red area) center point μ and variance $\sigma_{\mu^*}^2$. The darker the target-tree sample, the higher the probability that the target-tree sample q_{tar} is sampled.

trained through supervised learning. One of the network outputs is the center point of the biased range, $\mu^* \in \{x, y\}$, which is the label data of the network. To obtain μ^* for training, the target-tree* algorithm with RRT* is conducted N times during an each given planning time (T). Among the generated N paths, the location of the connected random sample q_{con} of the path is designated as μ^* through the following method.

First, n_{short} paths are found with a sufficient planning time T . Due to T , a path close to the shortest length is likely to be included in n_{short} . (see Figure A.4(a), e.g. n_{short} is 4, the path, $p_{rrt_2}^* \cup p_{tar_2}$, is the shortest.); In detail, among the generated N paths, n_{short} paths are selected in ascending order from the shortest length.

Second, one path is selected that can be searched as quickly as possible

among the obtained n_{short} paths; This path has the smallest p_{rrt^*} length ratio, which is shown in Figure A.4(b). Even if n_{short} are short enough, length ratios between the RRT* path (p_{rrt^*}) and the target-tree path (p_{tar}) in the entire path (p_{park}) may differ depending on where q_{con} is selected. As shown in A.4(a), the positions of q_{con_1} and q_{con_2} are different, and the ratio of path by q_{con_2} is lower than that by q_{con_1} . The shorter p_{rrt^*} reduces RRT* extension time to reach the target-tree, so the path planning time is shortened. Furthermore, the shorter p_{rrt^*} reduces the time required to rewire RRT* to reduce the length of p_{rrt^*} .

A.2.3 Experiments

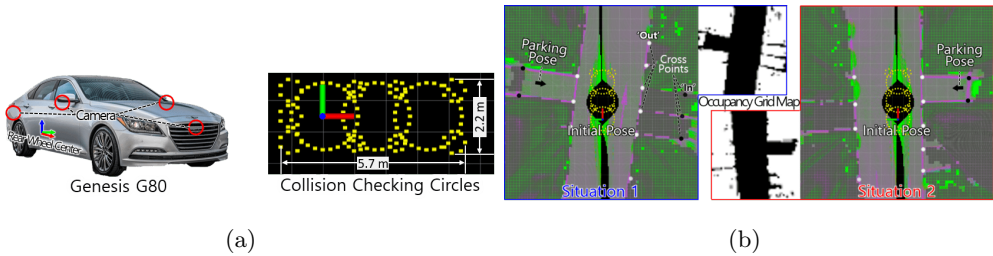


Figure A.5: (a) Autonomous vehicle (provided by PHANTOM AI) and collision checking range and (b) Parking situations for data collection.

System Setup: The path planning algorithms including the proposed algorithm were implemented based on the C++ version of the open motion planning library (OMPL). These algorithms were tested in the real environment with a Hyundai Genesis G80 vehicle (see Figure A.5(a)). An MSI GE 76 Raider laptop computer was used to implement and execute these algorithms, and specifications are as follows; CPU: i9-11980HK (2.6GHz) octa-core / GPU: GeForce RTX 3080 / Memory: DDR4 32 GB. The operating system of this computer is Ubuntu, and the middleware is Robot operating system (ROS Noetic) was used for communication with programs for the autonomous parking experiment.

Occupancy Grid Map (OGM) Setup: In order to obtain OGM, images obtained from left, right, front, and rear cameras were transformed to a composite bird’s eye view by using the Cam2World function in the Scaramuzza library [119]. This image’s size was 17.5 m \times 17.5 m with 200 \times 200 resolution, and it was divided into 25 \times 25 grids (see Figure A.5(b)). A deep neural network was used to segment the drivable area, the obstacle area, the parking spaces, and the parking line. To obtain the parking spot, cross points between the long and short parking lines, were recognized through the YOLO network [121] (see the white and black dots on the pink parking line Figure A.5(b)).

Path Planner Setup: The closest parking spot was regarded as the goal of the path planner, and the center of the rear wheel was set to the initial pose. The minimum turning radius of the vehicle was 7.0 m. The hybrid curvature [130] steering function was used to extend the RRT* path to the random sample, q_{rrt^*} or q_{tar} . Through this function, a continuous curvature path (except for the forward/backward switching point) can be planned.

Twelve collision check circles were configured to cover all areas of the vehicle to ensure that the extended trees do not collide with obstacles (see Figure A.5(a)). The number of backward paths in the target-tree was 21 from $-\kappa_{max}$ to κ_{max} , and the interval between each sample in the path is 0.01 m.

Network Training Setup: To collect the label data μ^* (the center point of the biased range), the target-tree* algorithm with RRT* [129] was conducted $N = 100$ times with $T = 100.0$ sec, and n_{short} was set to 5. These N , T , and n_{short} parameters were set to sufficiently find the shortest (optimal) parking path in the range of OGM and found experimentally. We gathered training data for two parking situations (see Figure A.5(b)). At the same time, OGM, initial pose, and parking spot data were collected at 0.5 m intervals, and a total of 67 data were collected. The multi-variate Gaussian log-likelihood loss function

\mathcal{L}_{Gau_i} [79] was used to predict μ^* and its variance $\sigma_{\mu^*}^2$. The Adam optimizer with a 10^{-5} learning rate was used, whereas pre-training weights were not used. The epochs were set to 10 k, and the batch size was 8.

Results: The sampling-based path planning algorithms for parking were compared; Informed RRT* [131] (*INF*), Bi-directional RRT* [132] (*BID*), Target-tree algorithm with RRT* [126] (*TT*), Target-tree* algorithm with RRT* [129] (*TTS*), Biased Target-tree* algorithm with RRT* (proposed, *BTTS*). The path planning was conducted 100 times in 4 parking situations (all un-trained environments) according to each path planning time. The conducted planning times were set to 0.01, 0.1, 1.0, and 10.0 sec. Here, *TT*, *TTS*, *BTTS* using the target-tree had the target-tree initialization time of about 3 msec. The success rate, which is the number of paths planned to the goal, was measured. Besides, the path length’s minimum, maximum, mean, standard deviation were also calculated.

The network outputs and the planning results are shown in Figure A.6

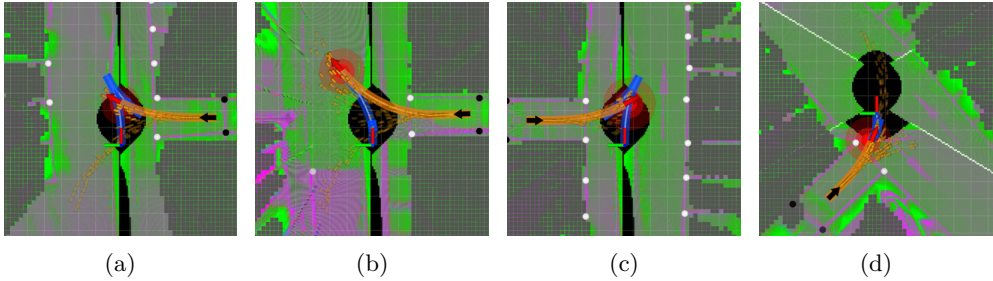


Figure A.6: Path planning results in un-trained environments using biased target-tree* algorithm with RRT* .

The above paths are the shortest path among 100 paths planned in 0.01 sec. The blue path is the RRT* path, and the orange path is the target-tree path. The red area is the biased range obtained by the deep neural network, and the darker it is, the lower the variance. The orange arrows are the target-tree samples, and the darker they are, the more likely they are selected.

Table A.2: Result for Figure A.6(a)

		Success	Min.	Max.	Mean	STDEV
		Rate	[m]	[m]	[m]	[m]
0.01 sec	<i>INF</i>	0/100	-	-	-	-
	<i>BID</i>	4/100	20.33	49.78	40.20	11.63
	<i>TT</i>	37/100	13.93	38.41	16.51	4.38
	<i>TTS</i>	34/100	13.93	24.90	16.75	3.29
	<i>BTTS</i>	98/100	13.93	15.04	14.13	0.22
0.1 sec	<i>INF</i>	0/100	-	-	-	-
	<i>BID</i>	82/100	35.84	83.00	51.42	8.80
	<i>TT</i>	96/100	13.93	73.76	25.90	17.58
	<i>TTS</i>	95/100	13.93	70.57	21.68	14.40
	<i>BTTS</i>	100/100	13.93	13.93	13.93	0.0
1.0 sec	<i>INF</i>	10/100	15.87	72.76	54.16	19.20
	<i>BID</i>	100/100	15.75	65.00	48.20	6.52
	<i>TT</i>	100/100	13.93	86.43	27.91	20.30
	<i>TTS</i>	100/100	13.93	75.84	16.73	9.51
	<i>BTTS</i>	100/100	13.93	13.93	13.93	0.0
10.0 sec	<i>INF</i>	16/100	19.80	69.21	48.10	16.24
	<i>BID</i>	100/100	25.74	60.75	45.85	4.76
	<i>TT</i>	100/100	13.93	72.46	25.31	16.51
	<i>TTS</i>	100/100	13.93	14.66	14.10	0.17
	<i>BTTS</i>	100/100	13.93	13.93	13.93	0.0

Table A.3: Result for Figure A.6(b)

		Success	Min.	Max.	Mean	STDEV
		Rate	[m]	[m]	[m]	[m]
0.01 sec	<i>INF</i>	2/100	16.25	20.78	18.51	2.26
	<i>BID</i>	64/100	17.30	50.68	28.80	6.97
	<i>TT</i>	73/100	14.00	33.88	19.20	5.58
	<i>TTS</i>	74/100	14.00	40.78	17.73	5.46
	<i>BTTS</i>	95/100	14.00	22.30	14.33	1.16
0.1 sec	<i>INF</i>	14/100	15.66	40.41	25.21	8.52
	<i>BID</i>	99/100	15.91	41.96	23.94	4.90
	<i>TT</i>	100/100	14.00	38.00	19.61	6.40
	<i>TTS</i>	100/100	14.00	30.70	15.47	2.87
	<i>BTTS</i>	100/100	14.00	14.14	14.00	0.20
1.0 sec	<i>INF</i>	32/100	16.50	91.39	28.40	14.19
	<i>BID</i>	100/100	16.28	37.14	23.30	3.52
	<i>TT</i>	100/100	14.00	47.61	20.57	7.36
	<i>TTS</i>	100/100	14.00	15.42	14.12	0.20
	<i>BTTS</i>	100/100	14.00	14.60	14.00	0.0
10.0 sec	<i>INF</i>	51/100	14.14	57.00	18.80	6.41
	<i>BID</i>	100/100	15.22	34.49	21.20	3.67
	<i>TT</i>	100/100	14.00	38.00	19.73	6.30
	<i>TTS</i>	100/100	14.00	14.14	14.10	0.30
	<i>BTTS</i>	100/100	14.00	14.60	14.00	0.0

Video Link: <https://youtu.be/q7yETLi7g88> Video Link: <https://youtu.be/unns1N4U80k>

and Table A.2, A.3, A.4, A.5. In all situations, the proposed algorithm ***BTTS*** had the highest path planning success rate regardless of the planning time. Especially at 0.01 sec planning time, ***BTTS*** had 86.9, 25.3, 2.45, and 2.3 times higher success rate than *INF*, *BID*, *TT*, and *TTS*, respectively. This value is the average of the success rates for 4 situations. Moreover, the minimum, maximum, mean and standard deviation path length of ***BTTS*** was shorter than these of other algorithms. Exceptionally in Figure A.6(d) parking situation, *TTS* planned shorter path, and lower mean and deviation than ***BTTS***, but the differences were small (see Table A.5).

BTTS had the highest probability of planning a path close to the shortest path. The shortest path was defined as the shortest path among 100 paths obtained by *TTS* with 100.0 sec. Figure A.7 is a graph of the probability of generating a path close to the shortest path in the 4 parking situations. This probability was defined as the ratio of the number of paths showing a difference from the shortest distance was less than 0.05 m among 100 paths. The reason for

Table A.4: Result for Figure A.6(c)

		Success Rate	Min. [m]	Max. [m]	Mean [m]	STDEV [m]
0.01 sec	<i>INF</i>	0/100	-	-	-	-
	<i>BID</i>	0/100	-	-	-	-
	<i>TT</i>	15/100	14.22	24.38	18.80	3.87
	<i>TTS</i>	19/100	14.22	24.38	19.55	3.69
	<i>BTTS</i>	74/100	14.22	14.47	2.15	
0.1 sec	<i>INF</i>	1/100	15.55	15.55	-	-
	<i>BID</i>	3/100	31.80	33.24	32.43	0.59
	<i>TT</i>	66/100	14.22	66.54	22.20	8.30
	<i>TTS</i>	68/100	14.22	36.56	20.73	6.12
	<i>BTTS</i>	96/100	14.22	21.67	14.35	0.93
1.0 sec	<i>INF</i>	1/100	25.50	25.50	25.50	-
	<i>BID</i>	4/100	24.74	68.68	41.72	16.59
	<i>TT</i>	92/100	14.22	72.40	24.18	10.48
	<i>TTS</i>	94/100	14.22	41.28	18.12	5.22
	<i>BTTS</i>	100/100	14.22	14.22	14.22	0.0
10.0 sec	<i>INF</i>	3/100	19.76	68.35	37.65	21.80
	<i>BID</i>	9/100	31.50	55.99	41.84	7.99
	<i>TT</i>	97/100	14.22	67.18	22.73	7.97
	<i>TTS</i>	99/100	14.22	88.96	16.50	7.61
	<i>BTTS</i>	100/100	14.22	14.22	14.22	0.0

Table A.5: Result for Figure A.6(d)

		Success Rate	Min. [m]	Max. [m]	Mean [m]	STDEV [m]
0.01 sec	<i>INF</i>	0/100	-	-	-	-
	<i>BID</i>	71/100	10.80	35.92	26.57	5.69
	<i>TT</i>	97/100	7.25	17.83	9.50	2.40
	<i>TTS</i>	98/100	7.21	20.67	8.49	1.87
	<i>BTTS</i>	100/100	7.22	9.73	8.52	1.50
0.1 sec	<i>INF</i>	9/100	10.75	25.12	14.72	3.92
	<i>BID</i>	100/100	10.32	59.52	24.20	6.53
	<i>TT</i>	100/100	7.25	20.67	9.58	2.62
	<i>TTS</i>	100/100	7.21	8.79	7.53	0.26
	<i>BTTS</i>	100/100	7.22	9.59	7.29	0.37
1.0 sec	<i>INF</i>	28/100	10.30	71.18	18.23	11.8
	<i>BID</i>	100/100	11.40	44.52	20.94	5.15
	<i>TT</i>	100/100	7.25	20.67	9.58	2.61
	<i>TTS</i>	100/100	7.21	7.65	7.30	0.80
	<i>BTTS</i>	100/100	7.22	7.22	7.22	0.0
10.0 sec	<i>INF</i>	54/100	9.66	62.39	15.63	11.74
	<i>BID</i>	100/100	10.67	41.95	19.29	4.91
	<i>TT</i>	100/100	7.25	33.16	9.83	3.51
	<i>TTS</i>	100/100	7.21	7.41	7.24	0.3
	<i>BTTS</i>	100/100	7.22	7.22	7.22	0.0

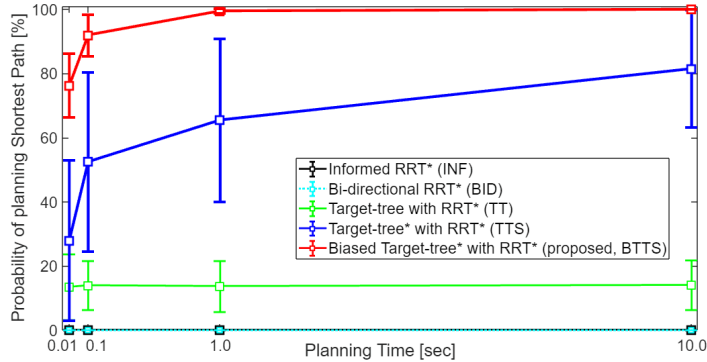
Video Link: <https://youtu.be/S46hWSfwCLQ>Video Link: <https://youtu.be/nJB8vaM23UO>

Figure A.7: Probability of Planning Shortest Path for Planning Time

this definition is that the shortest path is returned in multiple path generations, and the shortest path is more likely to be generated if a path close to the shortest length is frequently generated. In 0.01 sec planning time, the probability of *BTTS* was more than 70 % averagely. *BTTS* had 100 % probability over 1.0 sec, and had the highest probability with all given planning times.

Bibliography

- [1] NHTSA, “U.S. Department of Transportation the topic, the evolution of automated safety technologies,” <https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety>, May 30 2021.
- [2] M. Lu, K. Wevers, and R. Van Der Heijden, “Technical feasibility of advanced driver assistance systems (adas) for road traffic safety,” *Transportation Planning and Technology*, vol. 28, no. 3, pp. 167–187, 2005.
- [3] 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.
- [4] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, “Autonomous driving in urban environments: Boss and the urban challenge,” *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.
- [5] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, “Junior: The stanford entry in the urban challenge,” *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

- [6] A. Ibisch, S. Stümper, H. Altinger, M. Neuhausen, M. Tschentscher, M. Schlipfing, J. Salinen, and A. Knoll, “Towards autonomous driving in a parking garage: Vehicle localization and tracking using environment-embedded lidar sensors,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2013, pp. 829–834.
- [7] K.-W. Min and J.-D. Choi, “Design and implementation of autonomous vehicle valet parking system,” in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 2082–2087.
- [8] P. Furgale, U. Schwesinger, M. Rufli, W. Derendarz, H. Grimmer, P. Mühlfellner, S. Wonneberger, J. Timpner, S. Rottmann, B. Li, *et al.*, “Toward automated driving in cities using close-to-market sensors: An overview of the v-charge project,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2013, pp. 809–816.
- [9] D. H. Kang, C. M. Kang, J.-S. Kim, S. Kim, W.-Y. Kim, S.-H. Lee, and C. C. Chung, “Vision-based autonomous indoor valet parking system,” in *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2017, pp. 41–46.
- [10] R. Kummerle, D. Hahnel, D. Dolgov, S. Thrun, and W. Burgard, “Autonomous driving in a multi-level parking structure,” in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3395–3400.
- [11] S. Klemm, M. Essinger, J. Oberländer, M. R. Zofka, F. Kuhnt, M. Weber, R. Kohlhaas, A. Kohs, A. Roennau, T. Schamm, *et al.*, “Autonomous multi-story navigation for valet parking,” in *2016 IEEE 19th International*

- Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1126–1133.
- [12] U. Schwesinger, M. Bürki, J. Timpner, S. Rottmann, L. Wolf, L. M. Paz, H. Grimmett, I. Posner, P. Newman, C. Häne, *et al.*, “Automated valet parking and charging for e-mobility,” in *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 157–164.
- [13] D. Kohler, W. Hess, and H. Rapp, “Cartographer is a system that provides real-time simultaneous localization and mapping (slam) in 2d and 3d across multiple platforms and sensor configurations,” *Retrieved from github.com/googlecartographer/cartographer*, 2016.
- [14] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [15] S.-J. Han and J. Choi, “Real-time precision vehicle localization using numerical maps,” *ETRI Journal*, vol. 36, no. 6, pp. 968–978, 2014.
- [16] J. Ahn, Y. Lee, M. Kim, and J. Park, “Vision-based branch road detection for intersection navigation in unstructured environment using multi-task network,” *Journal of Advanced Transportation*, vol. 2022, 2022.
- [17] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.
- [18] M. Hentschel and B. Wagner, “Autonomous robot navigation based on openstreetmap geodata,” in *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE, 2010, pp. 1645–1650.
- [19] T. Ort, K. Murthy, R. Banerjee, S. K. Gottipati, D. Bhatt, I. Gilitschenski, L. Paull, and D. Rus, “Maplite: Autonomous intersection navigation

- without a detailed prior map,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 556–563, 2019.
- [20] M. Á. de Miguel, F. García, and J. M. Armingol, “Improved lidar probabilistic localization for autonomous vehicles using gnss,” *Sensors*, vol. 20, no. 11, p. 3145, 2020.
- [21] M. Osman, A. Hussein, and A. Al-Kaff, “Intelligent vehicles localization approaches between estimation and information: A review,” in *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 2019, pp. 1–8.
- [22] Z. Wang and A. Lambert, “A low-cost consistent vehicle localization based on interval constraint propagation,” *Journal of Advanced Transportation*, vol. 2018, 2018.
- [23] J. Ahn, S. Shin, and J. Park, “Development of localization using artificial and natural landmark for indoor mobile robots,” *The Journal of Korea Robotics Society*, vol. 11, no. 4, pp. 205–216, 2016.
- [24] F. Codevilla, M. Müller, 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)*. IEEE, 2018, pp. 4693–4700.
- [25] J. Ahn and J. Park, “End-to-end driving in unstructured environments using conditional imitation learning.”
- [26] K. Zhu, W. Chen, W. Zhang, R. Song, and Y. Li, “Autonomous robot navigation based on multi-camera perception,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5879–5885.

- [27] P. Cai, Y. Sun, H. Wang, and M. Liu, “Vtgnnet: A vision-based trajectory generation network for autonomous vehicles in urban environments,” *IEEE Transactions on Intelligent Vehicles*, 2020.
- [28] W. Fang, S. Zhang, H. Huang, S. Dang, Z. Huang, W. Li, Z. Wang, T. Sun, and H. Li, “Learn to make decision with small data for autonomous driving: deep gaussian process and feedback control,” *Journal of Advanced Transportation*, vol. 2020, 2020.
- [29] E. C. Pereira, D. A. Lima, and A. C. Victorino, “Autonomous vehicle global navigation approach associating sensor based control and digital maps,” in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE, 2014, pp. 2404–2409.
- [30] D. A. De Lima and A. C. Victorino, “Sensor-based control with digital maps association for global navigation: a real application for autonomous vehicles,” in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, 2015, pp. 1791–1796.
- [31] Q. Li, L. Chen, Q. Zhu, M. Li, Q. Zhang, and S. S. Ge, “Intersection detection and recognition for autonomous urban driving using a virtual cylindrical scanner,” *IET Intelligent Transport Systems*, vol. 8, no. 3, pp. 244–254, 2014.
- [32] Y. Yi, L. Hao, Z. Hao, S. Songtian, L. Ningyi, and S. Wenjie, “Intersection scan model and probability inference for vision based small-scale urban intersection detection,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1393–1398.
- [33] A. L. Ballardini, Á. Hernández Saz, S. Carrasco Limeros, J. Lorenzo, I. Parra Alonso, N. Hernández Parra, I. García Daza, and M. Á. Sotelo,

- “Urban intersection classification: A comparative analysis,” *Sensors*, vol. 21, no. 18, p. 6269, 2021.
- [34] L. Wang, J. Wang, X. Wang, and Y. Zhang, “3d-lidar based branch estimation and intersection location for autonomous vehicles,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1440–1445.
- [35] D. Bhatt, D. Sodhi, A. Pal, V. Balasubramanian, and M. Krishna, “Have i reached the intersection: A deep learning-based approach for intersection detection from monocular cameras,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4495–4500.
- [36] J. B. Chipka and M. Campbell, “Estimation and navigation methods with limited information for autonomous urban driving,” *Engineering Reports*, vol. 1, no. 4, p. e12054, 2019.
- [37] M. Kim, J. Ahn, and J. Park, “Global planning method for visiting roads with parking spaces in priority using rural postman problem,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 4184–4189.
- [38] D. Wu, M. Liao, W. Zhang, and X. Wang, “Yolop: You only look once for panoptic driving perception,” *arXiv preprint arXiv:2108.11250*, 2021.
- [39] Y. Qian, J. M. Dolan, and M. Yang, “Dlt-net: Joint detection of drivable areas, lane lines, and traffic objects,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4670–4679, 2019.
- [40] 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.

- [41] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.
- [43] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.
- [44] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.
- [45] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.
- [46] J. Redmon and A. Farhadi, “Yolo9000: better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.
- [47] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in *18th International Conference on Pattern Recognition (ICPR’06)*, vol. 3. IEEE, 2006, pp. 850–855.
- [48] 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.

- [49] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.
- [50] A. Bréhéret, “Pixel Annotation Tool,” <https://github.com/abreheret/PixelAnnotationTool>, 2017.
- [51] S. Qiu, “Object Bounding Box Labeling Tool,” <https://github.com/puzzledqs/BBox-Label-Tool>, 2017.
- [52] L. Adouane, *Autonomous vehicle navigation: from behavioral to hybrid multi-controller architectures*. United States: CRC Press, 2016.
- [53] 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.
- [54] C. Philippe, L. Adouane, B. Thuilot, A. Tsourdos, and H.-S. Shin, “Safe and online mpc for managing safety and comfort of autonomous vehicles in urban environment,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 300–306.
- [55] S. Zhu and B. Aksun-Guvenc, “Trajectory planning of autonomous vehicles based on parameterized control optimization in dynamic on-road environments,” *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1055–1067, 2020.
- [56] L. Li, X. Zuo, H. Peng, F. Yang, H. Zhu, D. Li, J. Liu, F. Su, Y. Liang, and G. Zhou, “Improving autonomous exploration using reduced approximated generalized voronoi graphs,” *Journal of Intelligent & Robotic Systems*, vol. 99, no. 1, pp. 91–113, 2020.

- [57] 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.
- [58] Y. Li, D. Li, C. Maple, Y. Yue, and J. Oyekan, “K-order surrounding roadmaps path planner for robot path planning,” *Journal of Intelligent & Robotic Systems*, vol. 75, no. 3, pp. 493–516, 2014.
- [59] S. Shin, J. Ahn, and J. Park, “Desired orientation rrt (do-rrt) for autonomous vehicle in narrow cluttered spaces,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4736–4741.
- [60] S. Yoon, C. Jung, and D. H. Shim, “Shape-aware and g2 continuous path planning based on bidirectional hybrid a for car-like vehicles,” *Journal of Intelligent & Robotic Systems*, vol. 103, no. 3, pp. 1–14, 2021.
- [61] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, “Any-time search in dynamic graphs,” *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.
- [62] M. Hoy, A. S. Matveev, and A. V. Savkin, “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey,” *Robotica*, vol. 33, no. 3, pp. 463–497, 2015.
- [63] 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.
- [64] 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.

- [65] H. Mouhagir, R. Talj, V. Cherfaoui, F. Aioun, and F. Guillemard, “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.
- [66] 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.
- [67] 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.
- [68] 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.
- [69] 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.
- [70] J. Ahn, M. Kim, and J. Park, “Autonomous driving using imitation learning with look ahead point for semi structured environments,” *Scientific Reports*, vol. 12, no. 1, pp. 1–17, 2022.
- [71] 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.

- [72] 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.
- [73] J. Zhang and K. Cho, “Query-efficient imitation learning for end-to-end autonomous driving,” *arXiv preprint arXiv:1605.06450*, 2016.
- [74] Y. Bicer, A. Alizadeh, N. K. Ure, A. Erdogan, and O. Kizilirmak, “Sample efficient interactive end-to-end deep learning for self-driving cars with selective multi-class safe dataset aggregation,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2629–2634.
- [75] W. Li, D. Wolinski, and M. C. Lin, “Adaps: Autonomous driving via principled simulations,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7625–7631.
- [76] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, “Exploring data aggregation in policy learning for vision-based urban autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 763–11 773.
- [77] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, “Hgdagger: Interactive imitation learning with human experts,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8077–8083.
- [78] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, “Imitation learning for agile autonomous driving,” *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

- [79] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, “Deep neural networks as gaussian processes,” *arXiv preprint arXiv:1711.00165*, 2017.
- [80] 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.
- [81] J. Zhang and K. Cho, “Query-efficient imitation learning for end-to-end simulated driving,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.
- [82] K. Judah, A. P. Fern, T. G. Dietterich, and P. Tadepalli, “Active imitation learning: Formal and practical reductions to iid learning,” *Journal of Machine Learning Research*, vol. 15, no. 120, pp. 4105–4143, 2014.
- [83] M. Monfort, M. Johnson, A. Oliva, and K. Hofmann, “Asynchronous data aggregation for training end to end visual control networks.” in *AAMAS*, 2017, pp. 530–537.
- [84] B. Kim and J. Pineau, “Maximum mean discrepancy imitation learning.” in *Robotics: Science and systems*, 2013.
- [85] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer, “Dropoutdagger: A bayesian approach to safe imitation learning,” *arXiv preprint arXiv:1709.06166*, 2017.
- [86] C. Cronrath, E. Jorge, J. Moberg, M. Jirstrand, and B. Lennartson, “Bagger: A bayesian algorithm for safe and query-efficient imitation learning,” in *Machine Learning in Robot Motion Planning–IROS 2018 Workshop*, 2018.

- [87] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, “Dart: Noise injection for robust imitation learning,” in *Conference on Robot Learning*, 2017, pp. 143–156.
- [88] Y. Cui, D. Isele, S. Niekum, and K. Fujimura, “Uncertainty-aware data aggregation for deep imitation learning,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 761–767.
- [89] 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.
- [90] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” *arXiv preprint:1711.03938*, 2017.
- [91] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [92] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.
- [93] A. Raffin, “Rl baselines3 zoo,” <https://github.com/DLR-RM/rl-baselines3-zoo>, 2020.
- [94] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora, “Model predictive contouring control for collision avoidance in unstructured dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4459–4466, 2019.
- [95] L. Tai, J. Zhang, M. Liu, and W. Burgard, “Socially compliant navigation through raw depth inputs with generative adversarial imitation

- learning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1111–1117.
- [96] H. Thomas, M. G. de Saint Aurin, J. Zhang, and T. D. Barfoot, “Learning spatiotemporal occupancy grid maps for lifelong navigation in dynamic scenes,” in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 484–490.
- [97] Z. Wang, Y. Zhuang, Q. Gu, D. Chen, H. Zhang, and W. Liu, “Reinforcement learning based negotiation-aware motion planning of autonomous vehicles,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4532–4537.
- [98] P. Cai, S. Wang, Y. Sun, and M. Liu, “Probabilistic end-to-end vehicle navigation in complex dynamic environments with multimodal sensor fusion,” *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4218–4224, 2020.
- [99] Z. Li, P. Zhao, C. Jiang, W. Huang, and H. Liang, “A learning-based model predictive trajectory planning controller for automated driving in unstructured dynamic environments,” *IEEE Transactions on Vehicular Technology*, 2022.
- [100] P. Cai, Y. Luo, A. Saxena, D. Hsu, and W. S. Lee, “Lets-drive: Driving in a crowd by learning from tree search,” *arXiv preprint arXiv:1905.12197*, 2019.
- [101] M. Everett, Y. F. Chen, and J. P. How, “Motion planning among dynamic, decision-making agents with deep reinforcement learning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

- [102] B. Brito, M. Everett, J. P. How, and J. Alonso-Mora, “Where to go next: learning a subgoal recommendation policy for navigation in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4616–4623, 2021.
- [103] X. Shen, E. L. Zhu, Y. R. Stürz, and F. Borrelli, “Collision avoidance in tightly-constrained environments without coordination: a hierarchical control approach,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 2674–2680.
- [104] L. Lei, R. Luo, R. Zheng, J. Wang, J. Zhang, C. Qiu, L. Ma, L. Jin, P. Zhang, and J. Chen, “Kb-tree: Learnable and continuous monte-carlo tree search for autonomous driving planning,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4493–4500.
- [105] J. Leu, Y. Wang, M. Tomizuka, and S. Di Cairano, “Autonomous vehicle parking in dynamic environments: An integrated system with prediction and motion planning,” *arXiv preprint arXiv:2204.10383*, 2022.
- [106] L. Qin, Z. Huang, C. Zhang, H. Guo, M. Ang, and D. Rus, “Deep imitation learning for autonomous navigation in dynamic pedestrian environments,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4108–4115.
- [107] Y. Pan, Q. Lin, H. Shah, and J. M. Dolan, “Safe planning for self-driving via adaptive constrained ilqr,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2377–2383.

- [108] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, “Adversarial policies: Attacking deep reinforcement learning,” *arXiv preprint arXiv:1905.10615*, 2019.
- [109] W. Guo, X. Wu, S. Huang, and X. Xing, “Adversarial policy learning in two-player competitive games,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 3910–3919.
- [110] S. Kuutti, S. Fallah, and R. Bowden, “Training adversarial agents to exploit weaknesses in deep control policies,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 108–114.
- [111] A. Sharif and D. Marijan, “Adversarial deep reinforcement learning for trustworthy autonomous driving policies,” *arXiv preprint arXiv:2112.11937*, 2021.
- [112] T. Hickling, N. Aouf, and P. Spencer, “Robust adversarial attacks detection based on explainable deep reinforcement learning for uav guidance and planning,” *arXiv preprint arXiv:2206.02670*, 2022.
- [113] T. Wu, F. Zhong, Y. Geng, H. Wang, Y. Zhu, Y. Wang, and H. Dong, “Grasparl: Dynamic grasping via adversarial reinforcement learning,” *arXiv preprint arXiv:2203.02119*, 2022.
- [114] K. X. Nguyen, D. Misra, R. Schapire, M. Dudík, and P. Shafto, “Interactive learning from activity description,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 8096–8108.
- [115] J. Song, H. Ren, D. Sadigh, and S. Ermon, “Multi-agent generative adversarial imitation learning,” *Advances in neural information processing systems*, vol. 31, 2018.

- [116] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.
- [117] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss, “Generalization, mayhems and limits in recurrent proximal policy optimization,” *arXiv preprint arXiv:2205.11104*, 2022.
- [118] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [119] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A toolbox for easily calibrating omnidirectional cameras,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 5695–5701.
- [120] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, “A stable tracking control method for an autonomous mobile robot,” in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 384–389.
- [121] Y. Park, J. Ahn, and J. Park, “Deep learning based parking slot detection and tracking: Psdt-net,” in *International Conference on Robot Intelligence Technology and Applications*. Springer, 2021, pp. 291–302.
- [122] P. Zips, M. Böck, and A. Kugi, “Optimisation based path planning for car parking in narrow environments,” *Robotics and Autonomous Systems*, vol. 79, pp. 1–11, 2016.

- [123] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path planning for autonomous vehicles in unknown semi-structured environments,” *The international journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.
- [124] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [125] L. Han, Q. H. Do, and S. Mita, “Unified path planner for parking an autonomous vehicle based on rrt,” in *IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 5622–5627.
- [126] Z. Feng, S. Chen, Y. Chen, and N. Zheng, “Model-based decision making with imagination for autonomous parking,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 2216–2223.
- [127] H. Banzhaf, P. Sanzenbacher, U. Baumann, and J. M. Zöllner, “Learning to predict ego-vehicle poses for sampling-based nonholonomic motion planning,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1053–1060, 2019.
- [128] Y. Dong, Y. Zhong, and J. Hong, “Knowledge-biased sampling-based path planning for automated vehicles parking,” *IEEE Access*, vol. 8, pp. 156 818–156 827, 2020.
- [129] M. Kim, J. Ahn, and J. Park, “Targettree-rrt*: Continuous-curvature path planning algorithm for autonomous parking in complex environments,” *IEEE Transactions on Automation Science and Engineering*, 2022.
- [130] H. Banzhaf, L. Palmieri, D. Nienhüser, T. Schamm, S. Knoop, and J. M. Zöllner, “Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments,” in *2017*

IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2017, pp. 1–8.

- [131] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2997–3004.
- [132] J.-H. Jhang, F.-L. Lian, and Y.-H. Hao, “Forward and backward motion planning for autonomous parking using smooth-feedback bidirectional rapidly-exploring random trees* with pattern cost penalty,” in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2020, pp. 260–265.

초 록

본 학위논문은 자율주행 차량이 주차장에서 위상지도와 비전 센서로 내비게이션을 수행하는 방법들을 제안합니다. 이 환경에서의 자율주행 기술은 완전 자율주행을 완성하는 데 필요하며, 편리하게 이용될 수 있습니다. 이 기술을 구현하기 위해, 경로를 생성하고 이를 현지화 데이터로 추종하는 방법이 일반적으로 연구되고 있습니다. 그러나, 주차장에서는 도로 간 간격이 좁고 장애물이 복잡하게 분포되어 있어 현지화 데이터를 정확하게 얻기 힘듭니다. 이는 실제 경로와 추종하는 경로 사이에 틀어짐을 발생시켜, 차량과 장애물 간 충돌 가능성을 높입니다. 따라서 현지화 데이터로 경로를 추종하는 대신, 낮은 비용을 가지는 비전 센서로 차량이 주행 가능 영역을 향해 주행하는 방법이 제안됩니다.

주차장에는 차선이 없고 다양한 정적/동적 장애물이 복잡하게 있어, 주행 가능/불가능한 영역을 구분하여 점유 격자 지도를 얻는 것이 필요합니다. 또한, 교차로를 내비게이션하기 위해, 전역 계획에 따른 하나의 갈래 도로만이 주행가능 영역으로 구분됩니다. 갈래 도로는 회전된 바운딩 박스 형태로 인식되며 주행가능 영역 인식과 함께 multi-task 네트워크를 통해 얻어집니다. 주행을 위해 모방학습이 사용되며, 이는 모델-기반 모션플래닝 방법보다 파라미터 튜닝 없이도 다양하고 복잡한 환경을 다룰 수 있고 부정확한 인식 결과에도 강인합니다. 아울러, 이미지에서 제어 명령을 구하는 기존 모방학습 방법과 달리, 점유 격자 지도에서 차량이 도달할 look-ahead point를 학습하는 새로운 모방학습 방법이 제안됩니다. 이 point를 사용함으로써, 모방 학습의 성능을 향상시키는 data aggregation

(DAgger) 알고리즘을 별도의 조이스틱 없이 자율주행에 적용할 수 있으며, 전문가 는 human-in-loop DAgger 훈련 과정에서도 최적의 행동을 잘 선택할 수 있습니다. 추가로, DAgger 변형 알고리즘들은 안전하지 않거나 충돌에 가까운 상황에 대한 데이터를 샘플링하여 DAgger 성능이 향상됩니다. 그러나, 전체 훈련 데이터셋에서 이 상황에 대한 데이터 비율이 적으면, 추가적인 DAgger 수행 및 사람의 노력이 요구됩니다. 이 문제를 다루기 위해, 가중 손실 함수를 사용하는 새로운 DAgger 훈련 방법인 WeightDAgger 알고리즘이 제안되며, 더 적은 DAgger 반복으로 앞서 언급 것과 유사한 상황에서 전문가의 행동을 더 정확하게 모방할 수 있습니다. DAgger 를 동적 상황까지 확장하기 위해, 에이전트와 경쟁하는 적대적 정책이 제안되고, 이 정책을 DAgger 알고리즘에 적용하기 위한 훈련 프레임워크가 제안됩니다. 에이전트는 이전 DAgger 훈련 단계에서 훈련되지 않은 다양한 상황에 대해 훈련될 수 있을 뿐만 아니라 쉬운 상황에서 어려운 상황까지 점진적으로 훈련될 수 있습니다.

실내외 주차장에서의 차량 내비게이션 실험을 통해, 모델-기반 모션 플래닝 알고리즘의 한계 및 이를 다룰 수 있는 제안하는 모방학습 방법의 효용성이 분석됩니다. 또한, 시뮬레이션 실험을 통해, 제안된 WeightDAgger가 기존 DAgger 알고리즘들 보다 더 적은 DAgger 수행 및 사람의 노력이 필요함을 보이며, 적대적 정책을 이용한 DAgger 훈련 방법으로 동적 장애물을 안전하게 회피할 수 있음을 보입니다. 추가적으로, 부록에서는 비전 기반 자율 주차 시스템 및 주차 경로를 빠르게 생성할 수 있는 방법이 소개되어, 비전기반 주행 및 주차를 수행하는 자율 발렛 파킹 시스템이 완성됩니다.

주요어: 비전-기반 내비게이션, 룩-어헤드 포인트, 다중-작업 네트워크, 모방 학습, 데이터 집계 알고리즘, 가중 데이터 집계 알고리즘, 적대적 에이전트 정책

학번: 2016-26039

ACKNOWLEDGEMENT

많은 분들의 도움으로 박사과정을 잘 마무리하며 박사학위 논문을 완성할 수 있었다고 생각하며, 매우 감사하게 생각합니다. 특히, 연구와 논문에 대해 많은 지도를 해주신 박재홍 교수님에게 깊은 감사 말씀을 올립니다. 교수님 덕분에 제가 좋아하는 자율주행 개발 및 연구를 마음껏 할 수 있었고, 행복한 연구실 생활을 할 수 있었습니다. 제가 개발하고 연구했던 결과물과 경험들에 대해 자부심을 가지며 뿌듯하게 생각합니다.

무엇보다도, 무인차팀원들에게 감사하다는 말을 하고 싶습니다. 연구실에 첫발을 디디면서부터 지금까지 개발 및 연구에 관해 정말 많은 것을 따듯하게 잘 알려주셨던 신세호 선배님에게 정말 감사하다는 말을 드립니다. 기쁘거나 힘들 때 옆에서 응원해 주시고 인간적으로 힘이 되어주신 김민성 선배님에게도 감사했습니다. 특히, 연구실 생활동안 가장 긴 시간을 함께 동고동락한 김민수 후배에게도 매우 고맙다는 말을 하고 싶습니다. 연구, 개발 관련된 것뿐만 아니라 개인적인 고민이나 냇두리와 같은 모든 얘기들을 잘 들어준 말동무였고, 많은 프로젝트를 훌륭하게 완료한 최고의 동료라고 생각하며, 항상 같이 일하고 싶은 동료로 기억될 것입니다. 어려운 부탁을 척척해내는 믿음직스러우면서 아주 건강한 이양우 후배가 있어 연구실 생활이 즐겁고 편했기에 많이 고맙다는 생각이 듭니다. 임규범 형님, 안찬우 후배, 차준혁 후배, Arthur 후배와도 같은 프로젝트를 진행하면서 많은 희로애락을 함께하며 좋은 경험들이 쌓인 것 같아 고맙다는 말 전합니다. 잠깐이었지만 여러 일을 도와주고 즐겁게 일했던 인턴들에게도 매우 고맙습니다.

연구 분야도 다르고 연구실도 달랐지만, 자주 술잔을 기울이며 많은 얘기를 하며 힘이 되어준 김상현, 김민곤, 심재훈 선배님, 그리고 장근우 동기 형님에게도 많이 감사드립니다. 덕분에 정말 많이 행복했습니다. 연구나 졸업 준비에 관해서 많은 조언을 해주신 최원제 선배님, 박현준 선배님, 그리고 박범영 선배님도 감사하였습니다. 마주칠 때마다 반갑게 맞아주시고 얘기를 잘 나누어주신 이이수 선배님, 김용태 선배님, 황순욱 선배님, 이지민 선배님, 박수민 선배님, 이호상 선배님, 김지섭 선배님, 조현범 선배님, 정재석 선배님, 김승연 선배님, 허성문 선배님, 김승훈 선배님에게도 감사의 말씀을 드립니다. 항상 저에게 따뜻하게 대해 주시고 좋은 인연을 만나게 해 주시신 김주완 선배님에게도 정말 감사합니다. 선배님들 모두 각자 계신 곳에서 좋은 결과 있으시고 행복하시길 바라겠습니다.

연구실에서 가끔 보더라도 편하게 잘 대해준 박수한 후배, 김준형 후배, 안준휘 후배, 김명주 후배, 이상엽 후배, 고종성 형님, 백지영 후배, 김명수 후배, 이해성 후배, 유승빈 형님, 성은호 후배, 이호균 후배, 김형철 후배, 박경재 후배, 신재용 후배, 그리고 김재현 후배에게도 고맙다는 말 전하고 싶습니다. 같은 연구실을 쓰면서 맛있는 것도 많이 먹고 연구 얘기도 잘 들어준 임대규 후배, 김동현 후배, 성동현 후배에게도 고맙다고 전합니다. 연구실에 계시는 모든 분들께서도 원하는 연구 하시면서 좋은 결과를 통해 행복한 연구실 생활을 하시면 좋겠습니다.

마지막으로 사랑하는 가족 및 친척들에게도 감사 말씀 전합니다. 어릴 때부터 항상 저를 믿고 응원해 주시고 존중해 주시고 자랑스러워해 주신 아버지, 어머니 그리고 동생이 있어 여기까지 올 수 있었습니다. 공학자의 길이 멋지고 존중받을 수 다는걸 보여주신 선배 공학자이신 할아버지에게도 감사드리고 존경드린다고 전하고 싶습니다. 대학원 생활 동안 열심히 도와주고 굳게 믿으며 응원해 준 사랑하는 아내 그리고 밝은 미소로 힘나게 해 주고 웃게 해 준 사랑하는 아들 리후가 있어 무사히 학위과정을 마칠 수 있었다고 생각합니다. 멀리서 저에게 격려와 응원을 아끼지 않아주신 삼촌, 숙모에게도 감사하다는 말씀 전합니다. 따뜻한 마음으로 학위과정에 대해 이해해 주시고 격려해 주신 장인어르신과 장모님에게도 정말 감사하다는 말씀 올립니다. 꼭 훌륭한 사람이 되어 보답하도록 하겠습니다.