Deep Perception-Action Coupling and Sensor Fusion for End-to-end Autonomous Driving

(End-to-end自動運転のための深層学習に基づく知覚-行動結合とセンサフュージョン)

July 2023

Doctor of Philosophy (Engineering)

Oskar Natan

オスカー ナタン

Toyohashi University of Technology

Abstract

In autonomous driving, how should we integrate perception and action components properly? What do we need to process and fuse multi-modal data from various sensors? In this study, we aim to answer these questions by conducting some experiments on end-to-end autonomous driving to achieve excellent drivability in complex environments under diverse conditions and scenarios.

End-to-end autonomous driving is a method that allows an autopilot model to directly use raw sensor data as the inputs and outputs the low-level control such as steering angle and throttle level. Since manual integration for joining perception and action parts is no longer needed, this method has become a preferred approach as the model can examine the necessity of information all by itself. Moreover, this method can be combined with imitation learning or behavior cloning which can be done easily with the supervised learning technique. By using the end-to-end behavior cloning strategy, we can create a single deep learning model to mimic an expert driver manipulation on vehicle control in handling complicated situations, which can be simulated using a simulator or derived from publicly available datasets to enrich the driving experiences. Although this method has plenty of benefits in performing human-like autonomous driving, there are two fundamental issues that remain and need to be addressed carefully. First, since the model has multiple outputs as it deals with perception and control tasks simultaneously at the same time, we must ensure that the solution for each task can be learned at an equal pace. This is necessary to prevent the model from tending to focus only on one specific task during the training process. Second, we need to design a complex network architecture using the correct layers and tune some hyperparameters accordingly, so that the model can handle multiple input data with different modalities. This challenge arises as we must create a compact unified model to avoid the burden of linking some independent task-specific modules, which may lead to information loss during the process of forming a connected modular system.

To tackle those issues, we propose an end-to-end multi-task model that can extract meaningful information from a set of observations retrieved by vehicle sensors and solve multiple tasks from the perception stage to the action stage in one forward pass. We also propose an adaptive loss weighting algorithm to balance the learning signal for each task equally. Then, we evaluate our proposed model by doing ablation and comparative studies with other models for clearer performance justification. The experimental results show that our model achieves superior performance in many criteria and aspects of driving. To better clarify the findings, this study is conducted gradually and step-by-step from the perception parts to the integration of perception and action parts, from predicting some driving records to performing automated driving, and from the utilization of simulation programs to real-world implementation on a robotic vehicle. Furthermore, as part of this thesis and to support future studies, we share the codes and data used in our publications publicly at https://github.com/oskarnatan. See the publication page for more details.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, autonomous driving technology has entered society advancing the transportation system in many sectors. This technology aims to automate vehicle operations to support human work, ranging from personal to corporation uses. Since this technology has been applied in diverse areas, it is important to ensure its safety by keeping up the drivability.

1.1 Research Background

Autonomous driving technology contains a lot of essential elements, either on the hardware side or software side [1]. On the software side, specifically based on the functional perspective, processing step, and information flow, a complete autonomous driving system is composed of four main parts: perception, planning, control, and system supervision [2]. The main objective of the perception part is to perceive the surrounding area around the ego vehicle by processing given data provided by various sensors that usually come with different modalities. Perception has always been a challenging task in developing the foundation of a complex autonomous driving system. The system needs to fully understand what kinds of objects are showing up on cameras and their relative distance from the ego vehicle [3]. Once clear information is available, the system is ready to receive commands like goals or missions for the planning part. Then, after the trajectory or navigation path is generated, any instruction related to the actuator can be executed in the controller part. Finally, the system supervision is responsible to monitor all aspects of the vehicle and ensure that everything works as planned [4].

There are two main techniques to integrate all of those parts into a unified system or model. The first technique works by joining each part where the outputs of a certain module are connected to another module for further information processing. This technique is often called classical autonomous driving where everything is integrated manually using a combination of hand-crafted methods that also incorporates the use of machine learning and rule-based algorithms [5]. Meanwhile, the other technique is referred to as end-to-end autonomous driving. True to its name, this technique allows a model to directly process multi-modal raw sensor data and output navigational controls to drive a vehicle [6]. In end-to-end autonomous driving, the model can almost be made entirely with deep multi-task learning that is capable of examining the necessity of information all by itself. Therefore, such a manual integration is not necessary. Both techniques still have limitations in many aspects such as handling unexpected situations, reliance on very accurate high-cost sensors, difficulty in inclement weather or low visibility, etc. However, researchers and engineers are actively working to address these challenges and improve the safety and reliability of autonomous driving technology.

1.1.1 Classical Autonomous Driving

Classical autonomous driving is a technique to make a complete driving system by integrating some independent or task-specific modules. It relies on a combination of sensors to gather information about the surroundings [7]. This information is then processed by a set of algorithms to make decisions about how the vehicle should navigate in the environment. In classical autonomous driving, these algorithms are typically designed to handle specific tasks, such as lane-keeping, path planning, and decision-making. Then, this method works by manually connecting the outputs of a specific module to another module for further processing [8]. In this section, we list some notable works in the field of classical autonomous driving where some of which are currently deployed on the street, ranging from lab projects to the products developed by many automobile companies in a chronological order as follows.

- Carnegie Mellon University's NavLab [9] [10] [11]: Initiated in 1984, this project was among the earliest efforts to develop autonomous driving technology that enabled vehicles to drive on public roads. This project laid the foundation for many subsequent projects and initiatives in this field.
- An Autonomous Land Vehicle in a Neural Network (ALVINN) [12]: Published in 1988, this project developed a neural network-based algorithm that could recognize road features, such as lane markings and road signs, and steer a vehicle along a predetermined path.
- DARPA Grand Challenge [13]: Held since 2004, this competition spurred significant advances in the field of autonomous driving as teams worked to develop vehicles capable of navigating difficult terrain and completing complex tasks without human intervention.
- Google's self-driving car: In 2009, Google began developing its self-driving car technology, which was based on a combination of sensors, cameras, and software algorithms, and was designed to enable fully autonomous driving. In late 2016, Waymo, which is the successor of Google's self-driving car technology, began testing fully self-driving cars on public roads without a human safety driver.
- Tesla's Autopilot: Initially introduced in 2014, Tesla's Autopilot system offered driver assistance features, such as lane keeping and adaptive cruise control, but required drivers to remain attentive and ready to take control of the vehicle at any time. This system will be replaced by Tesla's Full Self-Driving (FSD) Beta that is currently under testing.
- Toyota's Guardian and Chauffeur systems: In 2018, Toyota announced the development of two autonomous driving systems: Guardian and Chauffeur. The Guardian system assists human drivers with advanced driver assistance features, while the Chauffeur system is designed to enable fully autonomous driving.
- Mobileye's EyeQ Ultra: Introduced in 2022, this fifth-generation system-on-a-chip is designed to power advanced driver assistance systems and autonomous driving. It includes a range of sensors and software algorithms and is capable of processing massive amounts of data in real time to enable highly accurate and reliable autonomous driving.

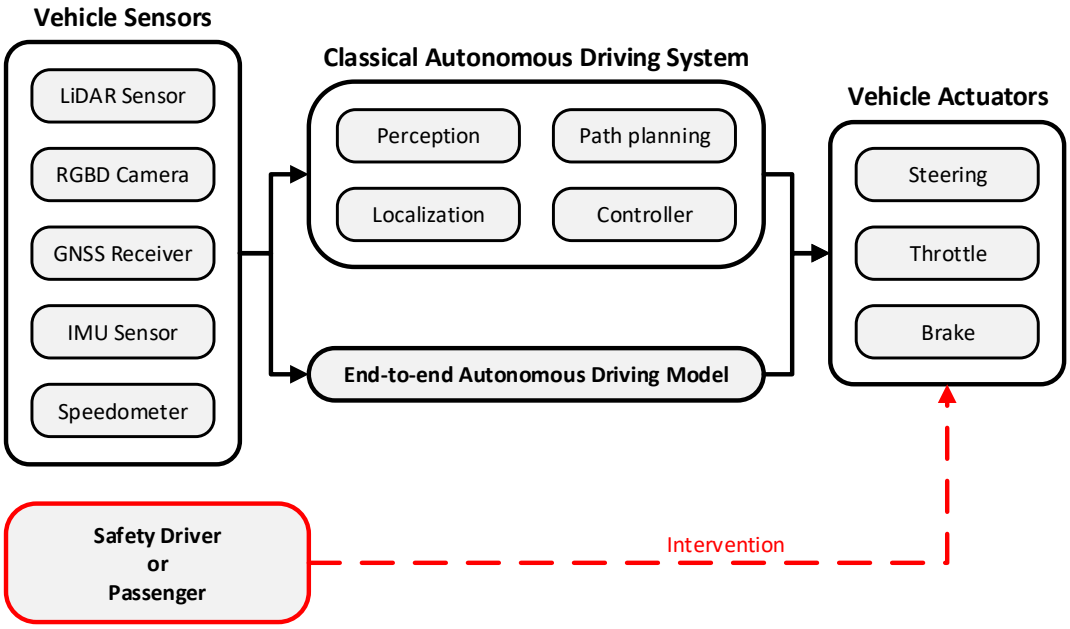


FIGURE 1.1: The difference between classical and end-to-end driving systems.

In classical autonomous driving, the system is composed of several task-specific modules that are integrated manually to perform automated driving. Meanwhile, in end-to-end autonomous driving, the system is represented with only one single model that can do equal jobs as the task-specific modules combined together. Usually, both systems are still covered with a kind of safety system that allows a driver or passengers to intervene for preventing any accidents.

1.1.2 End-to-end Autonomous Driving

End-to-end autonomous driving is a technique that allows a compact unified model to perceive the environment and drive a vehicle simultaneously at the same time. Unlike the classical autonomous driving system, an end-to-end autonomous driving model can use raw sensor data as its inputs and outputs the navigational controls directly in one forward pass [14]. Be noted that both systems still have to be covered with a kind of emergency or safety system that allows a driver or passenger to take over the vehicle actuators for safety reasons such as preventing any accidents due to system malfunction. The basic analogical difference between classical and end-to-end autonomous driving can be seen in Fig. 1.1. Compared to the classical approach, the end-to-end approach is way more sophisticated as a tedious manual integration is no longer needed. Moreover, we do not have to worry about information loss as the model can examine the necessity of data and extracted information all by itself [15]. Although end-to-end autonomous driving is not reliable enough to be deployed at this moment, this technique is said to be promising for future autonomous driving compared to existing classical systems.

In this section, we first explain perception as the core of reasoning in autonomous driving. Environmental perception and scene understanding can be said as fundamental keys in end-to-end autonomous driving as they play an important role that affects everything in the next stages. Then, we describe how perception and controller parts can be integrated or connected in an end-to-end manner in a single deep multi-task learning model.

Driving Perception

As the first stage in an autonomous driving system, perception holds an important role in scene understanding, which is the foundation before making any further decisions [16] [17]. To better understand the surrounding areas, a model may perceive the environment by performing many kinds of vision tasks such as object detection [18], semantic segmentation [19], and depth estimation [20]. When it comes to computer vision problems, deep learning algorithm, especially convolutional neural network (CNN) has been proven as state-of-the-art by plenty of research [21] [22]. However, employing a single-task deep learning model to handle each vision task can be costly and inefficient. Thus, a multi-task learning (MTL) model with a task-balancing algorithm that can handle multiple perception tasks simultaneously is preferable [23] [24]. Furthermore, perception with different perspectives of views is also important to improve scene understanding. This can be solved by mounting some sensors at several positions on the ego vehicle or projecting data into different perspectives of views [25]. Besides that, there are plenty of other challenges that must be tackled to achieve an excellent scene understanding [26]. For instance, the environmental condition can be varied such as the weather can be sunny, cloudy, foggy, or rainy. The situation on the road is also unpredictable as there are numerous vehicles and pedestrians along with their uncertain behavior. Therefore, the system must be supported with multiple kinds of sensors to provide various data and cover each other's weaknesses [27]. For example, a system cannot rely only on the RGB camera as it may fail in poor illumination conditions. To overcome this problem, another sensor such as dynamic vision sensor (DVS), radar, and LiDAR can serve as alternatives [28] [29] [30]. This leads to another issue on how to process multiple data with different modalities. To answer this issue, a sensor fusion technique that fuses different data such as combining RGB images with DVS images [31] [32] or with depth (RGBD) images [33] can be utilized to provide meaningful information in representing the environmental condition [34] [35]. With various data modalities taken as inputs, the model is expected to perform better in a dynamically changing environment.

Joined Perception and Control

Not only handling the vision-related tasks in the perception stage, but an autonomous driving system also deals with other tasks in the control stage [36] [37]. As a complex intelligent system, an autonomous driving system consists of several subsystems that handle multiple subtasks. The solution for each task can be done by simply employing a specific module [38]. However, this approach is costly and inefficient as a further configuration is needed to form an integrated modular system [39]. For example, we must assign the information provided by perception modules (e.g., semantic segmentation, object detection) as the input for the controller module. This integration process can be very tedious and may lead to information loss as a lot of parameters adjustment is done manually. With rapid deep learning research, plenty of works have been conducted to address this issue by training a single model in end-to-end manners [40] [41] [42]. The model can be trained to provide the final action solely based on the observation data captured with a set of sensors. As there is no manual tuning, the model leverages the extracted features all by itself [43]. End-to-end learning has become a preferable approach in autonomous driving as manual configuration to integrate task-specific modules is no longer needed. This technique

allows the model to share useful features directly from perception modules to controller modules. Moreover, the model can learn and receive extra supervision from a multi-task loss function that considers several performance criteria. All these benefits result in a better model performance even with a smaller model size due to its compactness [44]. To date, there have been a lot of works in the field of end-to-end autonomous driving, whether it is based on simulation [45], or offline real-world where the model predicts a set of driving records [46], or online real-world where the model is deployed for automated driving [47]. Besides dealing with diverse conditions, another challenge that remains in online real-world autonomous driving is that the model must deal with noise and inaccuracy of sensor measurement. This issue needs to be addressed as it affects model performance [48]. To address this issue, some works have been conducted with a focus on simulation-to-real adaptation. Although the models still suffer from performance losses due to sensor inaccuracies and diverse conditions, these approaches are said to be promising for future autonomous driving [49] [50]. Different approaches have been proposed with a focus on end-to-end imitation learning where the model is trained to mimic an expert in dealing with the issues [51] [52]. These approaches are preferable as they are easier and can be done with simple supervised learning. Moreover, plenty of publicly available datasets along with self-made datasets can be used for training the model to enrich its driving experiences.

1.2 Goal and Contribution

The goal of this study is to develop an end-to-end autonomous driving model that can handle both perception and control tasks simultaneously in one forward pass. The model acts as an autopilot agent that drives a robotic vehicle safely in diverse environments with various conditions. The model must have excellent drivability in the point-to-point navigation task determined by the capability of driving in the traversable area and following a set of route points in GNSS coordinates while avoiding any obstacles to prevent collisions as illustrated in Fig. 1.2. In order to accomplish this goal, we develop some essential components, which are also the contribution of this study as follows.

- A task-balancing algorithm called Modified Gradient Normalization (MGN) for ensuring a multi-task model can learn how to solve many tasks at an equal pace. We implement the algorithm to train some models that process multiple data modalities and perform multiple vision tasks simultaneously. Using the multi-task model, we also study how sensor fusion and different data representations can improve the overall performance [53] [54].

- An end-to-end autonomous driving model that takes multi-modal inputs provided by several sensors to perform both perception and control tasks in one forward pass. Evaluated in a simulated environment, the model achieves excellent drivability under different conditions and scenarios [55].

- A real-world autonomous driving demonstration by deploying the model to drive a robotic vehicle in real environments. This shows how a proof-of-concept study can be realized by addressing some implementation issues. Moreover, this also exhibits the usefulness of end-to-end imitation learning for multi-input multi-output models [56] [57].

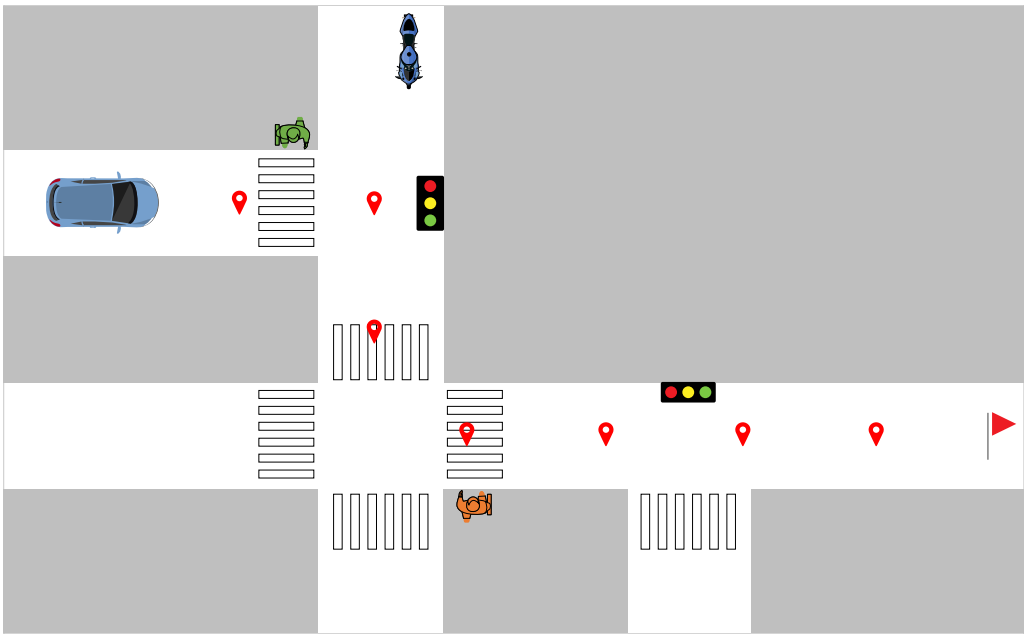


FIGURE 1.2: Point-to-point navigation task.

The model drives a vehicle safely following a set of route points in the form of GNSS coordinates. The model must obey traffic rules and avoid any obstacles to prevent collisions.

1.3 Thesis Structure

Following Chapter 1, which describes the background and purpose of this study, we provide some literature reviews in Chapter 2 that examine two critical components: perception-action coupling and sensor fusion in autonomous driving. Then, the rest of this thesis is arranged based on our publications that are organized as follows.

- Chapter 3: We propose an adaptive loss weighting algorithm to balance the learning signal of a compact model that solves multiple vision tasks simultaneously [53] [54]. This is important as we need to achieve excellent driving perception, which is the fundamental stage in autonomous driving technology.
- Chapter 4: We develop an end-to-end model that unifies the perception and control parts in its architecture to perform autonomous driving in one forward pass [55]. The model is evaluated in a CARLA-simulated environment to justify its drivability and clarify some other aspects of driving performance.
- Chapter 5: We conduct real-world experiments by deploying the model to drive a robotic vehicle in real environments [56]. This exhibits the usefulness of imitation learning for a complex model in the real world.
- Chapter 6: We improve the model architecture so that it can deal with more challenging conditions [57]. The model is employed to drive the robotic vehicle at night when the illumination is poor and everything is not clearly visible.

Based on the experimental results in each chapter, we provide analysis and discussion to disclose several findings. Finally, we summarize the conclusion and list some future research directions based on those findings in Chapter 7.

Chapter 2

Literature Review

In this Chapter, we provide some literature reviews of two critical components in our study, which are perception-action coupling and sensor fusion for end-to-end autonomous driving. Then, more detailed reviews explaining related works are also provided in the next chapters.

2.1 Perception-Action Coupling with Deep Learning

Perception is a crucial component of autonomous driving technology, and ongoing advancements in this field are critical for making self-driving cars safe and reliable for everyday use. Among various approaches to autonomous driving, perception has always been the first stage as it is important to understand the surrounding area before planning and action. Perception refers to the ability of the system to interpret data from various sensors, such as cameras and lidar, in order to understand the surrounding area and identify different objects such as pedestrians and other vehicles on the road. As a core of reasoning in autonomous driving, environmental perception can be achieved by performing various vision tasks such as semantic segmentation, depth estimation, and object detection [58] [59] [60]. In the field of autonomous driving research, Hahner et al. proposed a segmentation model that is made specifically to deal with foggy conditions [61]. Then, different work is proposed by Rajaram et al. [62] where a model called RefineNet is used to perform object detection. Besides completing a single vision task, the model can be pushed further to perform multiple vision tasks simultaneously to achieve a better scene understanding [63]. In deep learning, the process of learning to perform several tasks simultaneously is called multi-task learning (MTL). MTL aims to leverage shared feature maps during the training process to boost the performance of each task [64]. There are plenty of studies in the MTL area that are applied to autonomous driving vehicle problems. For instance, Teichmann et al. [65] proposed an MTL model called MultiNet that is used to perform several perception tasks such as road segmentation, vehicle detection, and street classification simultaneously. The model works well, however, it needs improvement in recognizing more crucial and various objects on the road. Kocic et al. [66] presented a network architecture called J-Net that processes RGB images to perform various control tasks such as controlling the steering wheel, speed, brake, etc. However, the simulation condition still needs to be improved to test the model generalization. These issues are solved by Cipolla et al. [67] where they develop a semantic segmentation model to recognize more various kinds of objects in the Cityscapes dataset [68]. It also performs instance segmentation and depth estimation by creating a branch of task-specific decoder for each task. However, it would be better if the model can take multiple inputs with multiple data modalities so that it can be applied to multi-view systems for a better scene understanding.

A model also needs to use the information from the perception parts to support the controller parts. In the field of end-to-end autonomous driving where perception and control are coupled together, better visual perception means better drivability as the controller gets better features directly from the perception module [69]. With the rapid deep learning research, perception and control parts can be coupled together in an end-to-end manner to avoid manual integration that prone to information loss. An end-to-end model is proven to have a better generalization as it can leverage the feature-sharing mechanism within its layers. Moreover, each neuron can receive extra supervision from a multi-task loss formula that considers multiple performance criteria. This results in a compact model that is relatively small but has a great performance which is preferable for real deployment [70]. Recent progress is made by Ishihara et al. [71] where an end-to-end model is deployed to perform multiple vision tasks and predict navigational controls. It is disclosed that performing vision tasks can improve drivability as the controller receives better perception features. Similar work is also proposed by Chitta et al. [72] where a camera-powered model called AIM-MT (auto-regressive image-based model with multi-task supervision) is deployed to perform automated driving in a simulated environment. This model completes perception and control tasks simultaneously to drive a vehicle. The RGB encoder of this model is guided by bird's eye-view (BEV) semantic prediction to provide better features to the controller decoder. Although it has a promising performance in poor illumination conditions, this model is practically hard to implement as it is difficult to provide the BEV semantic ground truth for the training process.

2.2 Leveraging Sensor Fusion Technique

Sensor fusion is a technique that leverages rich information provided by various sensors. Since each sensor comes with different data modalities, this technique must have the ability to handle all information as its input. In the field of autonomous driving, a model that is capable of processing multiple different inputs has been proposed by Hane et al. [73] where several cameras are placed in several different positions on the ego vehicle. Thus, the model will have a better capability in understanding the environmental condition. Although it has more views of RGB images, the model may still fail during nighttime or heavy rain due to poor illumination conditions. Processing one kind of data modality is not reliable for autonomous driving as it can be failed under certain conditions. Therefore, more heterogeneous data is needed to cover each other's weaknesses and produce more meaningful information through sensor fusion techniques [74]. A dynamic vision sensor (DVS) camera can be used as an alternative for providing information, especially in performing an active perception [75]. Hence, the idea of using the DVS camera can also be adopted in solving driving perception problems. To handle various data modalities, Nobis et al. [76] have demonstrated that a deep learning model can be used to process multiple sensor data by fusing extracted feature maps from each input modality. Nobis et al. proposed an object detection model called CameraRadarFusionNet (CRF-Net) that processes camera and radar data to get a better performance on a challenging autonomous driving dataset namely nuScene dataset [77]. CRF-Net provides a specific encoder for each input data and fuses extracted feature maps into the Feature Pyramid Network [78] to perform bounding boxes regression and classification in one forward pass. Then, another similar work is proposed by Niesen and Unnikrishnan [79] where camera and radar are fused to achieve accurate 3D depth reconstruction on the highway.

With using complex deep learning architectures, sensor fusion techniques can also be applied to end-to-end autonomous driving. Recent work is proposed by Prakash et al. [80] where a model that is supported with a camera, LiDAR, GPS receiver, and speedometer is used to perceive the environment and drive a vehicle at the same time. Inside its architecture, the model has two main modules, which are the perception module and the controller module. In the perception module, the camera is used to capture an RGB image in front of the vehicle, while LiDAR is used to capture point clouds around the vehicle. Then, the point clouds are projected into a 2-bin histogram over a 2D BEV grid with a fixed resolution [81]. Furthermore, a certain transformer-based module called TransFuser [82] is used to learn the relation between the RGB image and the projected point clouds to achieve a better perception. With this configuration, the model can perceive from the front and BEV perspectives. Meanwhile, in the controller module, a gated recurrent unit layer [83] is used to bias the extracted perception features with GPS coordinates and speed information. Then, the biased features are decoded into several waypoints that are translated into throttle and steering levels by a certain controller. Later similar work is proposed by Shao et al. [84] where a set of transformers is used to learn the contextual relationship between four RGB images (front, left, right, focus) and BEV-projected LiDAR point clouds. The model named InterFuser interprets its scene understanding and reasoning in the form of an object density map that shows the existence of any objects near the ego vehicle. Then, this map is used to constrain the final control action and maintain a safe distance from any objects. Both TransFuser and InterFuser show promising performance by combining multi-modal data to perceive the environment, however, other issues come as mounting two or more different sensors can cost more space, equipment, and extra budget. Therefore, using an equivalent sensor that is cheaper and can do a similar function may be preferable to tackle this problem. For example, a LiDAR can be replaced with a depth camera (merged with an RGB camera) to perceive the depth [85]. In the use of RGBD image for autonomous driving, Huang et al. [86] demonstrated how RGB image and depth map can be fused and extracted from the early perception stage to provide better features for the controller. Furthermore, the depth map can be also projected so that the model can perceive from a different perspective for a better perception.

Chapter 3

Multi-sensor Driving Perception with Balanced Learning

Our study begins with the experiment on driving perception which is the first stage of the autonomous driving system. In this Chapter, we present a novel compact deep multi-task learning model to handle various autonomous driving perception tasks in one forward pass. The model performs various computer vision tasks such as semantic segmentation, depth estimation, point cloud segmentation, and bird's eye view projection simultaneously in multiple views without being supported by other models. We also provide an adaptive loss weighting algorithm to tackle the imbalanced learning issue due to plenty of given tasks. Through data pre-processing and intermediate sensor fusion techniques, the model can process and combine multiple input modalities retrieved from RGB cameras, dynamic vision sensors (DVS), and a light detection and ranging (LiDAR) sensor placed at several positions on the ego vehicle. Therefore, a better understanding of a dynamically changing environment can be achieved. Based on the ablation study, the model variant trained with our proposed method achieves a better performance. Furthermore, a comparative study is also conducted to clarify its performance and effectiveness against the combination of some recent models. As a result, our model maintains better performance even with much fewer parameters. Hence, the model can infer faster with less GPU memory utilization. Moreover, the result tends to be consistent in three different CARLA simulation datasets and one real-world dataset namely nuScenes-lidarseg.

3.1 Motivation

To achieve a compact scene understanding and fulfill the needs in the perception stage, we conduct some experiments that focus on driving perception as shown in Fig. 3.1. Given a set of input data, we propose a model that performs various perception tasks with multiple perspectives of views: front (F), left (L), right (Ri), rear (R), and top (T). To be more specific, our model performs semantic segmentation (SS), depth estimation (DE), LiDAR segmentation (LS), and bird's eye view projection (BEVP) simultaneously. We use 4 RGB cameras, 4 DVS, and 1 LiDAR to provide rich information on a dynamically changing environment. Then, data preprocessing and sensor fusion techniques are used to handle multiple kinds of data modalities [87] [88]. Thus, a compact scene understanding, especially in the surrounding area of the ego vehicle can be achieved. We consider using the multi-task learning (MTL) approach since handling each task with a single-task model can be very costly and inefficient [89] [90]. However, for an MTL model, learning by combining several tasks is not always consistently better than in single-task learning. Different combined tasks may be conflicting with the gradient signals during the

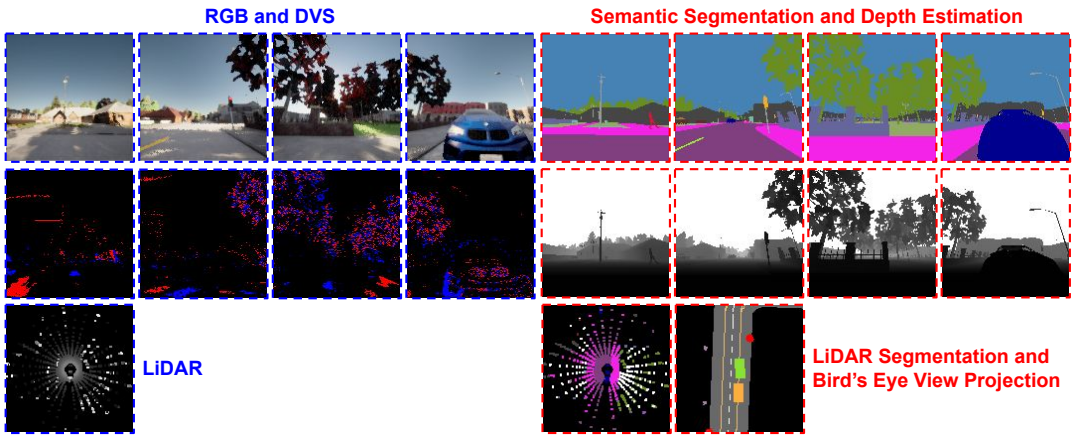


FIGURE 3.1: The inputs and outputs of the proposed model.

Given four views (L-F-Ri-R) of RGB and DVS images, and a top-view of pre-processed LiDAR point clouds as inputs (blue), the model performs four views of semantic segmentation and depth estimation along with a top-view of LiDAR segmentation and bird's eye view projection as outputs (red) simultaneously in one forward pass.

training process. If this issue is ignored, the outcome of the MTL approach cannot be optimal and cause performance degradation, or the training process may focus on one specific task only [91]. Hence, a proper strategy to balance the gradient and prevent imbalanced learning is a must. One of the possible answers is to give a set of loss weights to compensate for the imbalance. However, tuning a combination of loss weights can be tedious and computationally expensive. Therefore, rather than giving a fixed set of values [92], loss weights need to be tuned automatically [93] [94].

3.2 Related Work

In this section, we review several related works in the field of autonomous driving perception. We consider adopting and modifying some approaches to address challenges and issues in developing our model. In each subsection, we also summarize how these works are contributing to our study.

3.2.1 Handling Different Data Modalities

The idea of learning multiple tasks simultaneously is to leverage shared features during the training process. In the area of multi-task learning (MTL) for autonomous driving perception, Lv et al. [95] develop a model that takes a single RGB image to predict lane area and lane marking simultaneously. With a simple encoder-decoder style, the model is made with one RGB encoder and then branched into two task-specified decoders. A similar approach has been done by Chen et al. [96] where an MTL model called driving scene perception network is used to perform real-time joint detection, depth estimation, and semantic segmentation simultaneously. Moreover, Nakamura et al. [97] also conduct similar research to develop an MTL model that performs instance segmentation and depth estimation in one forward pass. Meanwhile, a different approach is presented by Yan et al. [98] where the model takes LiDAR point clouds to perform real-time occlusion-free road segmentation, dense road height estimation, and road topology recognition simultaneously. However, all of these approaches rely on one kind of input modality only which may

fail in unexpected environmental conditions. Due to the diversity of environmental conditions, an autonomous driving agent cannot rely on one kind of sensor only. For example, an RGB camera will be failed to capture the surrounding information during poor illumination conditions. To address this issue, another kind of sensor can be used as an alternative to retrieve the information. Therefore, a sensor fusion strategy may be needed to combine various data representations.

In the field of sensor fusion, Muresan and Nedevschi [99] combine LiDAR and RGB cameras to create affinity measurement and positional descriptor functions for an autonomous driving agent to perform multi-object tracking. Pre-trained models are used to process LiDAR point clouds and RGB images separately to obtain both LS and SS images. Then, a hand-crafted feature extractor and aggregator are used to perform the final calculation of object tracking. Another application of sensor fusion is presented by Dawar and Kehtarnavaz [100] where a depth camera and an inertial sensor are used for action detection and recognition in a continuous action stream. To extract depth images and inertial signal features, two separate deep learning-based encoders are used to process each input. A convolutional neural network (CNN) is used to handle the depth image, while a combination of CNN and a long short-term memory (LSTM) network is used to process inertial signals. Each encoder is performing detection and recognition, then a separate decision fusion model is used to make the final decision by leveraging extracted features from each encoder. However, this kind of late fusion strategy can lose potentially useful information as the extracted features are not shared among the encoders. To address this issue, Nie et al. [101] develop a multi-modality fusion framework called Integrated Multimodality Fusion Deep Neural Network (IMF-DNN) based on the intermediate fusion strategy where the extracted features are fused at some points in the network architecture. Their model takes multiple input modalities composed of LiDAR point clouds and RGB images, then fuses the extracted features several times. As a result, the IMF-DNN achieves higher performance in performing object detection and end-to-end driving policy in a diverse environment.

For our works, we imitate the architecture style presented by Lv et al. [95] that simply branches the decoder for each task. Hence, we have a task-specific decoder for each task on each view. Then, we adopt the intermediate fusion strategy proposed by Nie et al. [101] to combine multi-modal inputs retrieved from RGB cameras, DVS, and LiDAR by creating some fusion layers in the network architecture.

3.2.2 Bird's Eye View and LiDAR Representation

By having a bird's eye view projection (BEVP), an autonomous driving agent will have a better scene understanding whether in the form of a point-dot LS image or fully reconstructed BEVP image representations. In the field of BEVP, Reiher et al. [102] use four semantic segmentation images to construct BEVP. However, the model relies on other semantic segmentation models to provide four SS images. Thus, the entire process is not completed in one forward pass. With a similar concept, Palazzi et al. [103] develop a model that takes the front view RGB image along with its pre-predicted bounding boxes coordinate to estimate the bounding boxes on top-view perspective. Both approaches may fail due to poor illumination problems (night and heavy rain) since they only use a monocular camera to provide the information. Another similar approach is conducted by Mani et al. [104] where a single model is used to estimate BEVP without any help from other models. However, the model cannot estimate another view since it takes the front RGB image as the only input.

Besides using multiple RGB cameras, a 360^o LiDAR sensor can be used to collect point clouds that contain meaningful information about the surroundings. Moreover, LiDAR is more robust as it is not affected by light illumination conditions. Currently, there have been plenty of studies that conduct research on processing LiDAR point clouds to fit the input of a deep learning model. Point cloud-based models [105] [106] are known as the pioneers in taking the LiDAR point clouds directly, learning the feature, and predicting the label for each point. This mechanism is quite simple but the model tends to fail in capturing the local structure of an object. Then, in view-based models [107] [108], LiDAR point clouds are projected into several 2D frames with multiple perspectives of views, then a simple convolution layer is used to process each frame. However, the number of possibilities of views can be large and lead to an expensive computational cost. Thus, an effective way to pre-process LiDAR point clouds is needed to reduce the computational load while preserving useful information for the learning process.

In the BEVP task, any unnecessary projections can be eliminated since the surrounding objects are projected into the top-view perspective. Imad et al. [109] project raw LiDAR point clouds into a top-view RGB image that has three channels so that a transfer learning method from various pre-trained models can be applied to perform the BEVP task. Although the heatmap coloring technique is used to differentiate the data, a lot of information can be lost due to the limited scale. Then, an improved pre-processing approach is presented by Yang et al. [110] where LiDAR point clouds are stored in a 3D tensor with the height information of the point cloud kept as the third dimension like channels in an RGB image. Thus, a simple 2D convolution with a larger number of filters can be applied to process each channel. This data representation strategy is adopted by Zhang et al. to solve the LS task using the proposed model called PolarNet [111]. Finally, Chen et al. [112] develop a model that takes a top-view LS image to solve the BEVP task.

For our works, we also utilize all front, left, right, and rear images to support both LS and BEVP tasks as demonstrated by Reiher et al. [102]. Meanwhile, in preprocessing LiDAR point clouds, we combine two different techniques presented by Imad et al. [109] and Yang et al. [110]. Therefore, we will have a 3D tensor that stores all point clouds into two forms of representations that contain more useful information for the learning process.

3.2.3 Balancing Multiple Vision Tasks

A proper loss weighting strategy plays an important role in the training process of an MTL model, especially in tackling the imbalanced learning issue due to heterogeneous tasks with various loss functions. Cipolla et al. [67] conduct research in multi-loss weighting on an MTL model that performs scene understandings such as semantic segmentation, instance segmentation, and depth regression simultaneously. By conducting MTL experiments, they show that homoscedastic task uncertainty is an effective way to perform loss weighting on several tasks. Meanwhile, a different approach is presented by Chen et al. [113] where a loss weighting algorithm called Gradient Normalization (GradNorm) is proposed to control the training dynamics by manipulating the gradient during the training process. By adjusting the gradient signal, the learning conflict from different tasks can be minimized.

For our works, we adopt GradNorm [113] to deal with the imbalanced learning problem caused by plenty of tasks with different characteristics. We also do some modifications to the algorithm to meet our model needs.

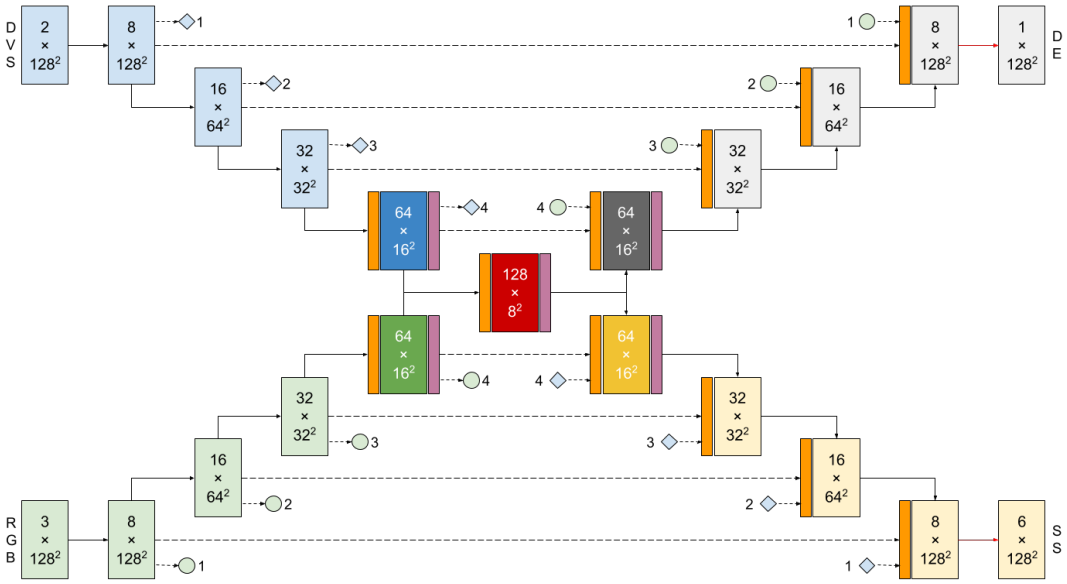


FIGURE 3.2: The architecture of the proposed model 1.

Blue and green boxes represent the inputs and feature maps for each view. Dark blue and dark green boxes are the concatenation across all views, while the dark red box is the concatenation of all feature maps. Then, grey and yellow boxes represent feature maps and outputs for each view of depth estimation (DE) and semantic segmentation (SS). 5 boxes in the center are considered as the bottlenecks where a dropout layer (purple) is applied for each. Dark grey and dark yellow boxes are the specific bottlenecks for each task. Solid lines represent the convolution block, while dashed lines represent the skip connection and concatenation (orange boxes). Each dashed line connects the feature map on the encoders with its symmetric feature map on the decoders. Denoted with numbered small blue squares and green circles, both encoders are used to support both tasks decoders. Finally, red lines represent the final pointwise convolution followed by an activation function.

3.3 Methodology

In this section, we explain the details of our proposed methods which are inspired by some related works reviewed in Section 3.2. First, we describe the model architecture and the proper loss and metric formulation. Then, we explain how we develop the adaptive loss weighting algorithm to tackle the imbalanced learning issue.

3.3.1 Proposed Model 1

To understand the contribution of the sensor fusion technique, we propose a model that takes RGB images and DVS images to perform semantic segmentation (SS) and depth estimation (DE) in multiple views. The visualization of the model architecture can be seen in Fig. 3.2. The model follows the encoder-decoder style [114] with additional skip connections inspired by U-Net paper [115]. Each feature map in both RGB and DVS encoders is connected to its symmetric feature maps in both DE and SS encoders. With this configuration, each encoder can act as a support for one other. For example, when the illumination is very poor and RGB cameras are failed to capture enough information about the surroundings, the network can learn how to leverage extracted information mainly from the DVS encoder. On the other hand, when the car is not moving (e.g. at the crossroads) and there is not enough information as the brightness change is very rare, the network can learn how to rely

more on the RGB encoder. In this architecture, each convolutional block on the encoder and decoder has $2 \times ((3 \times 3)$ convolutional layer + batch normalization [116] + ReLU activation [117]). Meanwhile, each convolutional block on the bottlenecks has 3 times more of them to extract more information from all views concatenation. Then, they are followed by (2×2) max-pooling on the encoder side and (2×2) bilinear upsampling on the decoder side. To deal with the overfitting issue, several dropout layers with $p = 0.5$ are placed on the bottleneck [118]. Finally, a pointwise (1×1) convolutional layer is used to reduce the channel number of feature maps to match the ground truth size. Then, it is followed by a sigmoid activation for SS and a ReLU activation [117] for DE.

In order to discover the advantage of sensor fusion on an MTL model, we perform an ablation study as follows. First, we remove the DVS input block (blue boxes) so that the model only processes RGB images to perform semantic segmentation and depth estimation. We refer to this model as A0 where only RGB images are fed into the network. Then, on the second model named A1, DVS inputs are added so that the model processes four pairs of RGB and DVS images. Feature maps from DVS encoders are concatenated to the semantic segmentation and depth estimation decoders as well as feature maps from RGB encoders. Furthermore, we also conduct a comparative study with another model to understand the usefulness of the feature-sharing mechanism in multi-task learning (MTL).

3.3.2 Proposed Model 2

Following the first model described in Subsection 3.3.1, we propose another model to study the contribution of adaptive loss weighting algorithm for a multi-input multi-output model. As shown in Fig. 3.3, we use a common encoder-decoder style with a specific encoder and decoder for each input and output as demonstrated by Lv et al. [95]. Then, we add several skip connections to connect the feature maps on the encoder side with their symmetric feature maps on the decoder side inspired by the famous U-Net architecture [115]. This mechanism aims to enhance the model performance by leveraging combined feature maps on the bottleneck with the specifically extracted features from each decoder. Each RGB encoder is connected and concatenated to each semantic segmentation (SS) decoder that has the same view and spatial dimension. Rich color, shape, and much more extra information contained in the RGB image can be helpful for segmentation problems. Meanwhile, the DVS input is specifically used to support the depth estimation (DE) task by connecting and concatenating each pair of symmetric encoder-decoders in a similar way to the RGB-SS pair. DVS image can be helpful for estimation problems, especially during poor illumination conditions since it contains contrast information even if there is only a small brightness change. Then, we connect the encoder of pre-processed LiDAR point clouds to the LiDAR segmentation (LS) decoder in line with the LS encoder to the bird's eye view projection (BEVP) decoder as they have the same top perspective of view. Similar to Chen et al. [112], our model performs BEVP by leveraging LS image. However, instead of taking the LS image directly as its input, we feed the model with the raw LiDAR point clouds that have been pre-processed to perform LS, then utilize the LS output to perform the BEVP task. Thus, there is no need to use another model to specifically support the LS task first. Our model also leverages 4 views of SS images as inspired by Reiher et al. [102] along with 4 views of DE images to support the LS encoder in performing BEVP. Finally, by following the intermediate fusion technique presented by Nie et al. [101], we create two bottlenecks in the form of convolution blocks to fuse and process multiple extracted

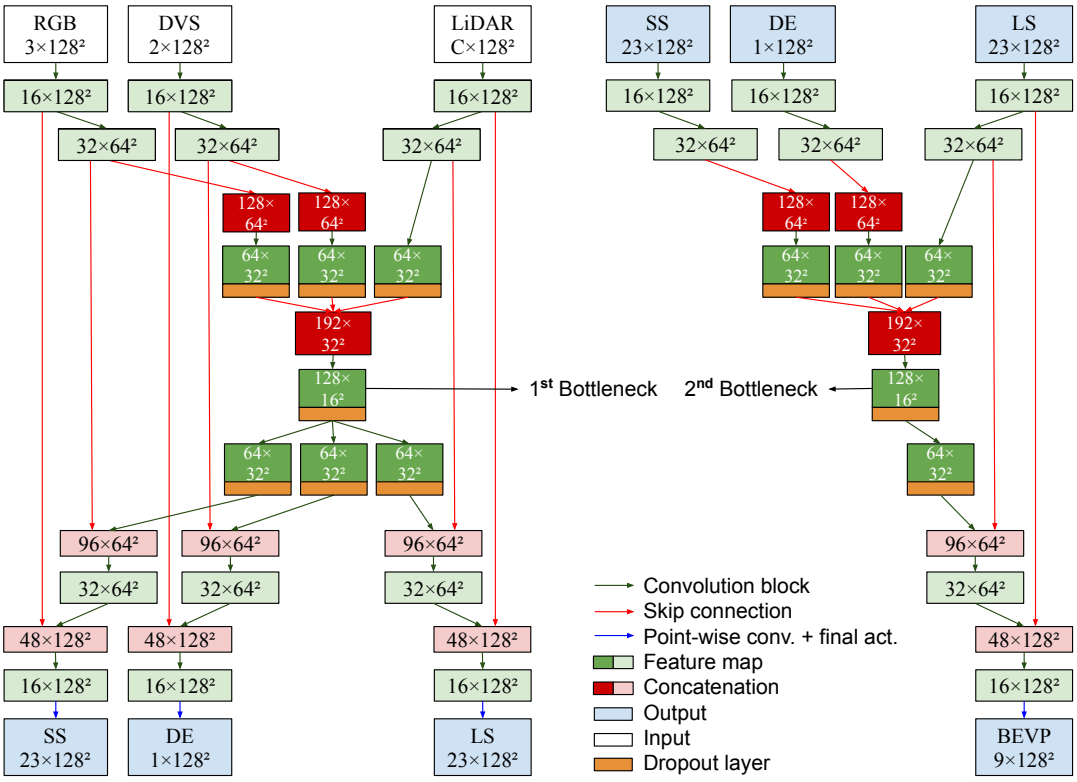


FIGURE 3.3: The architecture of the proposed model 2.

To be noted, each view (Front, Left, Right, and Rear) on RGB, DVS, SS, and DE have its own encoder, while LiDAR and LS only have one encoder as there is only one view (Top). The text written inside each box is the tensor size in $C \times S^2$, where C is the number of channels and S^2 is the spatial dimension (height $\times$ width). For the LiDAR encoder, the network can take 1 or 15 layers of pre-processed LiDAR point clouds (see Subsection 3.4.3).

feature maps from various input encoders. The 1^{st} bottleneck is used to store the extracted latent space from three inputs (RGB, DVS, and LiDAR) and used to perform SS, DE, and LS tasks. Meanwhile, the 2^{nd} bottleneck is used to store the extracted information from those tasks and perform BEVP as the final task. Hence, a compact architecture that performs multiple tasks in one forward pass can be obtained.

The detailed explanation of the network architecture shown in Fig. 3.3 is as follows. Each red line represents the skip connection that connects each pair of symmetric encoder-decoders followed by a concatenation process. The dark green line represents a common convolution block that consists of $2 \times$ (3×3 convolution layer + batch normalization [116] + ReLU activation [117]) and is followed by a 2×2 max-pooling layer for encoder path or 2×2 bilinear upsampling for decoder path. On the encoder path, the spatial dimension of the tensor is reduced by half while the number of feature maps in the channel axis is doubled each time it passes the convolution block. Meanwhile, on the decoder path, the spatial dimension is doubled while the number of feature maps is reduced by half gradually. Finally, the blue line represents a point-wise 1×1 convolution layer that reduces the channel so that the number of output elements will match the number of channels of the ground truth. Then, a sigmoid layer is used to perform point-wise classification for SS, LS, and BEVP tasks and a ReLU layer for point-wise regression in a positive normalized range of 0 to 1 for the DE task. To prevent overfitting, we add a dropout layer [118] with a drop rate of $p = 0.5$ on each convolution block in the bottlenecks.

3.3.3 Loss and Metric Formulation

To train the model and monitor its performance, loss and metric functions are needed to be formulated carefully. Loss functions are used to update the model weights while metric functions are used to monitor the model performance. Therefore, we use the metric scores to decide whether the training must be stopped or the learning rate should be reduced. To be noted, comparing loss values will not give a fair comparison due to different loss weights computed by the adaptive loss weighting algorithm. Hence, we use several metric scores as their calculation remains the same on each training epoch. As for the depth estimation (DE) loss ($\mathcal{L}_{DE}$), we calculate Huber loss as in (3.1).

$$\mathcal{L}_{DE} = \frac{1}{V}\sum_{i=1}^{V}\frac{1}{N}\sum_{j=1}^{N} z_{ij}, \tag{3.1}$$

where z_{ij} is given by (3.2).

$$z_{ij} = \begin{cases} 0.5(\hat{y}_{ij} - y_{ij})^2 & \text{if } |\hat{y}_{ij} - y_{ij}| < \delta \\ \delta(|\hat{y}_{ij} - y_{ij}| - 0.5\delta) & \text{otherwise} \end{cases} \tag{3.2}$$

We average the loss across all tensor elements N on all views $V = 4$. The number of N is also equal to the number of elements in pre-processed ground truth for DE task I_{DE} (see Subsection 3.4.3). Then, y_{ij} is the value of j^{th} element of the ground truth I_{DE} with view i, while $\hat{y}_{ij}$ is the predicted value of j^{th} element of the predicted depth output with view i after ReLU activation. Huber loss is widely used and suitable for the DE task as it takes the advantage of both mean squared error (MSE) and mean absolute error (MAE) based on the prediction results. We set $\delta = 0.5$ as the threshold for the Huber loss to start to curve like MSE if $|\hat{y}_{ij} - y_{ij}| < \delta$ or constantly have a large gradient which is the same as MAE if $|\hat{y}_{ij} - y_{ij}| \geq \delta$. Meanwhile, for the rest of segmentation-related tasks, we use the combination of standard binary cross entropy (BCE) and Dice loss as in (3.3) to calculate $\mathcal{L}_{SS}$, $\mathcal{L}_{LS}$, and $\mathcal{L}_{BEVP}$.

$$\mathcal{L}_{\{SS,LS,BEVP\}} = \frac{1}{V}\sum_{i=1}^{V}\left(\frac{1}{N}\sum_{j=1}^{N} y_{ij}log(\hat{y}_{ij}) + (1-y_{ij})log(1-\hat{y}_{ij})\right) + \left(1 - \frac{2|\hat{y}_i \cap y_i|}{|\hat{y}_i| + |y_i|}\right) \tag{3.3}$$

Similar to the Huber loss function, in the BCEDice loss function, the final loss calculation is also averaged across all tensor elements N and all output views $V = 4$ for semantic segmentation (SS) task and $V = 1$ for LiDAR segmentation (LS) and bird's eye view projection (BEVP) tasks. Then, y_i is the ground truth I_{SS} or I_{LS} or I_{BEVP} with view i and $\hat{y}_i$ is the predicted output of view i. Finally, the total loss can be calculated by multiplying each loss $\mathcal{L}_i$ with a loss weight w_i and summing all of them. In addition, to prevent overfitting, a weight decay [119] with $\lambda = 0.0001$ is used to penalize model complexity by multiplying the sum-squared of model weights $\mathcal{W}$ and added to the total loss as in (3.4).

$$\mathcal{L}_{total} = \lambda\Sigma\mathcal{W}^2 + \sum_{i=1}^{T} w_i\mathcal{L}_i \tag{3.4}$$

To be noted, $\mathcal{L}_i$ is an element in a set of losses $\{\mathcal{L}_{DE}, \mathcal{L}_{SS}, \mathcal{L}_{LS}, \mathcal{L}_{BEVP}\}$. Then, for the metric functions, we use MAE (3.5) for depth estimation (DE) task and intersection over union (IoU) (3.6) for segmentation-related tasks.

$$MAE_{DE} = \frac{1}{V}\sum_{i=1}^{V}\frac{1}{N}\sum_{i=1}^{N}|\hat{y}_{ij} - y_{ij}| \tag{3.5}$$

$$IoU_{\{SS,LS,BEVP\}} = \frac{1}{V}\sum_{i=1}^{V}\frac{|\hat{y}_i \cap y_i|}{|\hat{y}_i| \cup |y_i|} \tag{3.6}$$

Finally, the total metric (TM) is calculated by summing all metric scores as formulated in (3.7). To be noted, we use TM to determine the best model as it represents overall model performance in all tasks. The total loss (3.4) cannot be used for comparison as it is affected by the multiplication of loss weights and weight decay which are varied amongst models. Then, in order to know the discrepancy between tasks and show how balanced the performance is across all tasks, we calculate the metric variance (MV) within MAE_{DE}, $1 - IoU_{SS}$, $1 - IoU_{LS}$, and $1 - IoU_{BEVP}$ with (3.8).

$$TM = MAE_{DE} + (1 - IoU_{SS}) + (1 - IoU_{LS}) + (1 - IoU_{BEVP}) \tag{3.7}$$

$$MV = \frac{1}{T}\sum_{i=1}^{T}\left(M_i - \frac{TM}{T}\right)^2, \tag{3.8}$$

where M_i is the metric score of task-i and $\frac{TM}{T}$ is the mean of all metric scores with total tasks $T = 4$. Keep in mind that the lower TM and MV scores mean the better and more balanced the model performance.

3.3.4 Adaptive Loss Weighting

An adaptive loss weighting strategy can be used to deal with the imbalanced learning issue caused by plenty of given tasks with different characteristics. Hence, we adopt the GradNorm algorithm [113] and do some modifications to match our proposed model. The overall process of the modified GradNorm (MGN) algorithm can be seen in Fig. 3.4. Normally, the total loss for a multi-task model can be computed with (3.9).

$$\mathcal{L}(t) = \sum_{i=1}^{T} w_i(t)\mathcal{L}_i(t), \tag{3.9}$$

where $\mathcal{L}_i$ is the loss function of task-i from a T number of tasks and a static loss w_i is used to balance the learning process at training step t. Usually, the loss weights are tuned empirically which results in a huge computational cost to find the best set of loss weights. To address this issue, the GradNorm algorithm [113] is invented to learn the loss weight w_i by adjusting the gradient norms dynamically so that different tasks can be trained at similar rates. However, in their original paper, this algorithm is used to balance the learning process of the multi-task model of three tasks with only one input and one bottleneck of shared layers. Meanwhile, our model has four tasks with three data modalities, five different views as the input, and two bottlenecks of shared layers. Besides that, the loss weights are updated on each step and cause a huge computational load. We solve this issue by modifying the update process only at the last step $t = s$ for each epoch which means that there is only one update for one epoch. The number of maximum steps s is equal to the number of samples in the training set divided by the batch size. To be noted, the number of training samples on each dataset is different which means that the maximum step s can be varied depending on what dataset is currently used.

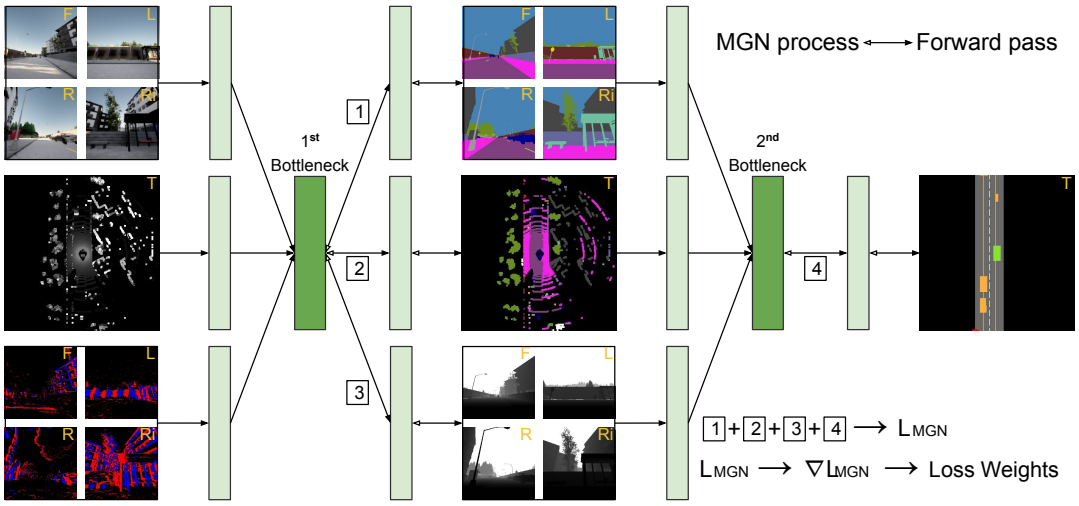


FIGURE 3.4: The process flow of the MGN algorithm.

The loss weight for each output on both SS and DE decoders is the same on each view since each SS loss and DE loss is averaged across all views.

Before the loss weights are updated on each epoch at step $t = s$, there are several quantities that need to be defined first as follows.

- Subsets of weights W from entire model weights $\mathcal{W}$ where the algorithm will be applied. We pick 2 subsets of weights as there are 2 bottlenecks in the network architecture shown in Fig. 3.3. $W(s)$ selection is expressed as (3.10).

$$W(s) = \{W_1(s), W_2(s)\} \subset \mathcal{W}(s) \tag{3.10}$$

  We pick $W_1(s)$ and $W_2(s)$ from the first layer of the 1^{st} and 2^{nd} bottleneck respectively. These layers are chosen since they have rich information on shared latent space from the concatenation of multiple feature maps.

- The L_2 norm of the gradient of the weighted single-task loss ($w_i(s)\mathcal{L}_i(s)$) with respect to the chosen subset of weights $W(s)$ that can be calculated with (3.11).

$$G_W^{(i)}(s) = \|\nabla_W(w_i(s)\mathcal{L}_i(s))\|_2 \tag{3.11}$$

  Based on the network architecture shown in Fig. 3.3, the gradient of the weighted single-task loss for BEVP is respected to $W(s) = W_2(s)$, while the others are respected to $W(s) = W_1(s)$. For further process, we need to compute $\overline{G}_W(s)$ which is the average of $G_W^{(i)}(s)$ across all tasks T with (3.12).

$$\overline{G}_W(s) = \frac{1}{T}\sum_{i=1}^{T} G_W^{(i)}(s) \tag{3.12}$$

- The ratio between $\mathcal{L}_i$ at the last step $t = s$ and first step $t = 0$ that can be computed with (3.13).

$$\bar{\mathcal{L}}_i(s) = \frac{\mathcal{L}_i(s)}{\mathcal{L}_i(0)} \tag{3.13}$$

  Concisely, the loss ratio $\bar{\mathcal{L}}_i(s)$ is also a measure of the inverse training rate of task-i where a lower ratio means a faster rate of learning task-i.

Algorithm 3.1: Training with the MGN algorithm

Initialize model weights $\mathcal{W}$ with kaiming init [120]
Set initial loss weights $w_i(0) = 1 \forall_i$
Set asymmetry alpha $\alpha = 1.5$

for *training step t = 0* **to** s **do**

- Input batch $x(t)$ and get prediction $\hat{y}(t)$
- Compute each single-task loss $\mathcal{L}_i(t)$
- Compute total loss $\mathcal{L}(t)$ with (3.9)

if $t = 0$ **then**
  Set initial task loss $\mathcal{L}_i(0) = \mathcal{L}_i(t)$
else if $t = s$ **then**
  Pick $W(s)$ with (3.10)
  Compute $G_W^{(i)}(s)$ with (3.11) for each task-i
  Compute $\overline{G}_W(s)$ with (3.12)
  Compute $\tilde{\mathcal{L}}_i(s)$ with (3.13) for each task-i
  Compute $r_i(s)$ with (3.14) for each task-i
  Compute $\mathcal{G}_W^{(i)}(s)$ with (3.16)
  Compute $\mathcal{L}_{MGN}(s)$ with (3.15)
  Compute MGN gradients $\nabla w_i \mathcal{L}_{MGN}(s)$
  Update each $w_i(s)$ using $\nabla w_i \mathcal{L}_{MGN}(s)$
  Normalize new $w_i(s)$ with (3.17)
end

- Compute gradients $\nabla_{\mathcal{W}} \mathcal{L}(t)$
- Update network weights $\mathcal{W}(t)$ using $\nabla_{\mathcal{W}} \mathcal{L}(t)$

end

Maximum training step s can be calculated by dividing the total training samples by batch size. It can be varied as the number of samples is different on each dataset

- The relative inverse training rate of task-i which can be calculated with (3.14). This variable is used to balance gradients during the training process.

$$r_i(s) = \frac{\tilde{\mathcal{L}}_i(s)}{\frac{1}{T}\sum_{i=1}^{T} \tilde{\mathcal{L}}_i(s)} \tag{3.14}$$

  The higher relative inverse training rate $r_i(s)$ means the higher gradient magnitude for task-i which results in the task being learned faster.

The detailed steps of the modified GradNorm (MGN) algorithm can be seen on Algorithm 3.1. To be noted, there are only two training steps in one epoch to be considered for MGN computation which are the first step $t = 0$ and the last step $t = s$. $\mathcal{L}_i(0)$ is very crucial, especially at the first epoch of the training process. Thus, proper model weights $\mathcal{W}$ initialization and task loss $\mathcal{L}_i$ formulation need to be considered carefully. Furthermore, both depth estimation (DE) and semantic segmentation (SS) tasks have multiple inputs from 4 different views while LiDAR segmentation (LS)

and bird's eye view projection (BEVP) tasks only have one input from a top perspective. Thus, the $\mathcal{L}_i$ for both DE and SS tasks are averaged across all views first before computing $\mathcal{L}_i(0)$ and $\mathcal{L}_i(s)$. This means that the loss weight w_i for each DE task and SS task will be the same on any view. The MGN algorithm is deployed as a loss function that computes the MAE between the target and actual gradient norms as in (3.15) for each task in every last step $t = s$ on each epoch.

$$\mathcal{L}_{MGN}(s) = \sum_{i=1}^{T} \left| \mathcal{G}_W^{(i)}(s) - G_W^{(i)}(s) \right|, \tag{3.15}$$

where the loss is summed across all tasks T with the target gradient $\mathcal{G}_W^{(i)}(s)$ is given by (3.16).

$$\mathcal{G}_W^{(i)}(s) = \overline{G}_W(s) r_i(s)^{\alpha} \tag{3.16}$$

We set the asymmetry $\alpha = 1.5$ as an additional parameter to control the balancing rate. The higher α value means stronger balancing enforcement which is usually used if tasks are significantly different [113]. Then, we use stochastic gradient descent (SGD) algorithm [121] to compute the gradients $\nabla w_i \mathcal{L}_{MGN}(s)$ and update the loss weights $w_i(s)$. We set the initial update rate $\eta_{MGN_0} = 0.1$ and reduce it by half until a minimum value of $\eta_{MGN_{min}} = 0.0001$ if there is no drop in total metric score in validation dataset in 4 epochs in a row. Finally, each loss weight $w_i(s)$ is normalized with (3.17) so that the sum of all loss weights will always equal T.

$$w_{i_{new}}(s) = \frac{w_i(s)}{\sum_{i=1}^{T} w_i(s)} T \tag{3.17}$$

3.3.5 Training Configuration

We use two GPUs, the NVIDIA RTX 2080 super and GTX 1080 Ti separately to train the model along with its variation described in Section 3.5. We develop the model entirely from scratch using PyTorch [122]. We do not use any pre-trained network to perform transfer learning and fine-tuning. As mentioned in Subsection 3.3.4, weights initialization can be crucial as it affects $\mathcal{L}_i(0)$, especially at the early epoch of the training process. Therefore, the Kaiming initialization strategy [120] is used to initialize the entire model weights $\mathcal{W}$. Then, a small batch size of 6 is enough since the model already takes multiple views of inputs. Similar to the loss weight updates, we use SGD [121] with momentum $\mu = 0.9$ to update the model weights during the training process. We set the initial learning rate $\eta_0 = 0.1$ and reduce it by half gradually until $\eta_{min} = 0.00001$ if there is no drop on the validation total metric (TM) score in 4 epochs in a row. We also stop the training process automatically if there is no drop in the validation TM score in 25 epochs in a row.

3.4 Experiment Setup

To strengthen our findings, we conduct experiments on four different datasets composed of three simulation datasets gathered using CARLA simulator [123] and one real-world dataset called nuScenes-lidarseg [77], which is also used to illustrate the implementation of the proposed model in a real-world scenario. Then, the preprocessing steps are explained to understand the data representation. We also provide a brief explanation of the training configuration.

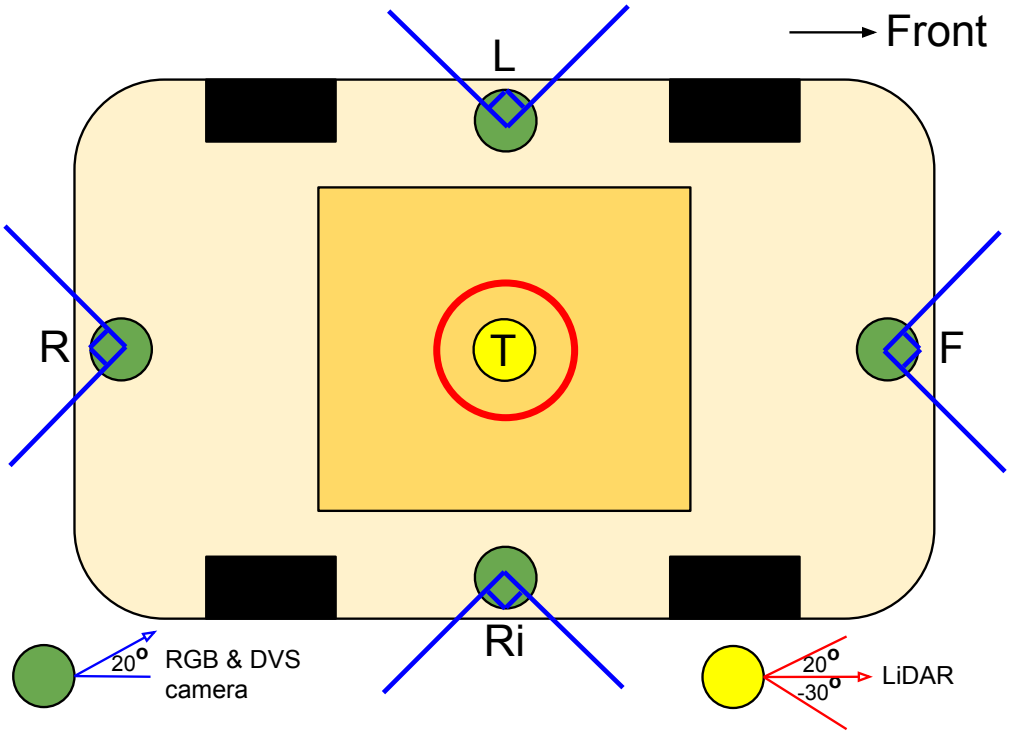


FIGURE 3.5: Sensors placement on a car.

Both RGB and DVS cameras are placed at four different positions, while the 360^o LiDAR sensor is placed at the top of the ego vehicle.

3.4.1 Simulated Environment

We use CARLA simulator [123] to generate simulation datasets to train our model. We collect a large amount of data composed of RGB images, DVS arrays, and LiDAR point clouds as the inputs and semantic segmentation (SS) images, depth estimation (DE) images, LiDAR segmentation (LS) images, and bird's eye view projection (BEVP) images as the outputs. We create three different datasets namely set A, B, and C for the experiment and strengthen our justification. In dataset A, we gather the simulation data from map Town01 as the training set and Town02 for both validation and test sets. Then, in dataset B, we collect the training set from Town02 and the rest validation and test sets from Town01. Meanwhile, in dataset C, we generate all simulation data from all maps (Town01 to Town05) for training, validation, and testing sets. Each map has different characteristics and contains various objects.

To retrieve more information, several sensors are mounted on the ego vehicle as shown in Fig. 3.5. We place RGB and DVS cameras at four positions, which are front (F), left (L), right (Ri), and rear (R). Each camera has a 90^o horizontal and vertical field of view, 20^o upward rotation, and resolution of $H \times W = 128 \times 128$. Then, a 360^o LiDAR sensor with 64 lasers and 32 meters of the maximum range is placed at the top (T). The LiDAR lasers are vertically spread between the range of -30^o to 20^o from the horizontal line. The same configuration is also applied to get the ground truth data for each task. During the data gathering process, we create a realistic condition by changing the weather dynamically where the environment can be sunny, rainy, foggy, morning, noon, evening, and night. Each condition also varies on a scale of 0 to 100% and can be combined. In addition, we also spawn non-player characters such as pedestrians and other vehicles to mimic the real situation on the road. The detailed data generation setting can be seen in Table 3.1.

TABLE 3.1: Data Generation Setting

Train : Val : Test ratio	3 : 1 : 1
Total data	2000 (set A and B), 10000 (set C)
Maps used	2 (set A and B), 5 (set C)
Simulation time	Morning, noon, evening, and night
Weather	Sunny, rainy, cloudy, and foggy
Non-player characters	Other vehicles (truck, car, bicycle, motorbike) and pedestrians
Object classes for SS and LS	Unlabeled, building, fence, other, pedestrian, pole, road lane, road, side walk, vegetation, other vehicles, wall, traffic sign, sky, ground, bridge, rail track, guard rail, traffic light, static object, dynamic object, water, terrain
Object classes for BEVP	Road, road lane, road centerline, other vehicles, ego vehicle, green traffic light, yellow traffic light, red traffic light, pedestrian
CARLA version	0.9.10.1

3.4.2 Real Environment

To illustrate how our model can be deployed in a real-world scenario, we also use nuScenes-lidarseg dataset [77] as the fourth dataset in our experiment. However, this dataset has a different sensor configuration and is not providing DVS images and ground truth for both semantic segmentation (SS) and depth estimation (DE) tasks. Thus, we consider modifying and pre-processing the dataset to meet the model needs. First, we use front-left and front-right images as the replacement for left and right images which are not provided in nuScenes-lidarseg. As there are no DVS images for the model inputs, we remove all DVS encoders in the network architecture and branch each RGB encoder to support both SS and DE decoders. Therefore, the model will take RGB inputs only to perform DE and SS tasks. Moreover, this dataset does not come with the ground truth for SS and DE tasks. Thus, we use the provided LiDAR point clouds associated with class and distance information to create ground truths for SS and DE tasks. We create SS and DE ground truths using the point clouds that are shown on each camera's perspective of view. Concisely, we plot each point cloud's associated class data as the SS ground truth and distance data as the DE ground truth. Then, to fill the gap between plotted frame's pixels, we give neighboring pixels the same class or value as the filled pixel. With this mechanism, we can obtain nearly similar ground truths as retrieved from the CARLA simulator. To be noted, we also resize all images to have a spatial dimension of $H \times W = 128 \times 128$ which is the same as in datasets A, B, and C. Thus, there is no need to make any further modifications to the model input size. Finally, since nuScenes-lidarseg has 32 possible object classes to be recognized, therefore, the number of channels of semantic segmentation (SS), LiDAR segmentation (LS), and bird's eye view projection (BEVP) outputs become $C = 32$. A set of nuScenes-lidarseg samples can be seen in Fig. 3.6.

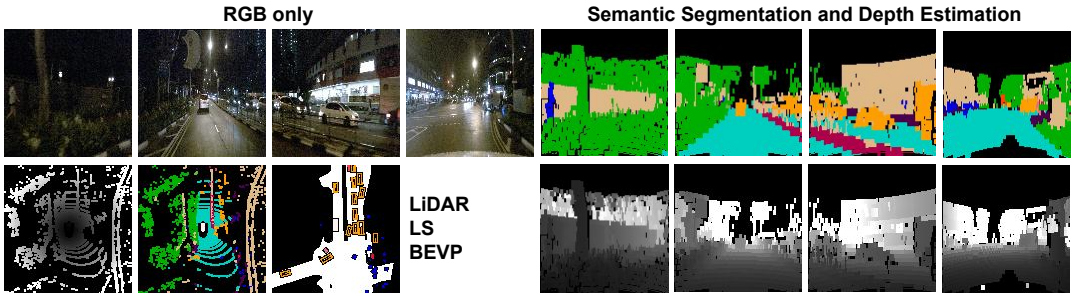


FIGURE 3.6: A set of pre-processed samples in nuScenes-lidarseg.

From left to right views: L-F-Ri-R while LiDAR, LS, and BEVP only have a top-view. There are no DVS images for model inputs so all DE decoders will retrieve extracted RGB feature maps only. Since there are no ground truths given for DE and SS tasks, the information of the LiDAR point cloud's associated class and distance are used to create the ground truth for both tasks.

Originally, the nuScenes-lidarseg dataset has 1000 driving scenes obtained from Boston and Singapore which have dense traffic and challenging situations. However, we only use the original train and validation (train-val) set (850 scenes) for our experiment. Meanwhile, the original test set (150 scenes) is excluded since the ground truth is not publicly available. Hence, we cannot measure the performance of the model. The nuScenes-lidarseg's train-val set has a total sample of 34149 from 850 scenes that are different from one another. We divide the original train-val set into the train, validation, and test sets with the ratio of 3:1:1 (the same as in datasets A, B, and C) based on the number of scenes so that the sample will be completely different on each set. Thus, there are 510 scenes (20418 samples) for training, 170 scenes (6873 samples) for validation, and 170 scenes (6858 samples) for testing.

3.4.3 Data Representation

An image is retrieved as $I_{RGB} \in \{0, ..., 255\}^{3\times128\times128}$ representing a set of 8-bit pixel values in the form of RGB channel (C) $\times$ height (H) $\times$ width (W). Different from I_{RGB}, the DVS array is retrieved as $A_{DVS} \in \mathbb{R}^{N\times4}$ where N is the total number of pixels that are considered to have a brightness change in one simulation step and $\mathbb{R}$ is an array with four elements consisted of timestamp, pixel's x-coordinate, pixel's y-coordinate, and pixel's polarization. The pixel's polarization can be positive or negative depending on the brightness change. Meanwhile, a set of LiDAR point clouds is retrieved as $A_{LID} \in \mathbb{R}^{M\times4}$ where M is the total number of point clouds retrieved in one simulation step and $\mathbb{R}$ is an array with four elements composed of point's x,y,z-coordinate and cos of incident angle ($\cos(\theta)$).

We normalize RGB images on a scale of 0 to 1 as $I_{RGB} \in \{0, ..., 1\}^{3\times128\times128}$. Since the model already takes multiple inputs of views, there is no need to feed the model with a bigger input size. Thus, it can reduce the computational load during the training and validation processes. To meet the input shape of the model, both A_{DVS} and A_{LID} need to be pre-processed first due to the possibility of numerous N and M which are affected by the simulation condition at a time. Thus, to deal with this issue, we pre-process both A_{DVS} and A_{LID} to become a 3D tensor with a fixed shape. With a maximum x,y coordinate range of $(127, 127)$, we project A_{DVS} into $I_{DVS} \in \{0, 1\}^{2\times128\times128}$ with the x,y-coordinate takes place on the spatial dimension ($H \times W = 128 \times 128$) and the polarization takes place on the channel dimension $C = 2$. In the channel axis, the positive polarization takes place on the first channel while

the negative polarization takes place on the second channel of I_{DVS}. We convert the polarization status into a value of 0 or 1 with (3.18). Meanwhile, the timestamp attribute in A_{DVS} is used to synchronize with other sensor data.

$$I_{DVS_{ij}} = \begin{cases} 1 & \text{if there is polarization record in } A_{DVS_{xy}} \\ 0 & \text{otherwise,} \end{cases} \tag{3.18}$$

where $I_{DVS_{ij}}$ is the i,j-coordinate derived from A_{DVS}'s x,y-coordinate. Meanwhile, for the LiDAR point clouds, we propose two different techniques to pre-process the LiDAR point clouds and perform an ablation study to understand their influence. The first method ignores A_{LID}'s z-coordinate and projects the LiDAR point clouds into a tensor with top-view perspective $I_{LID} \in \{0, ..., 1\}^{1\times128\times128}$ with (3.19). This kind of input representation is similar to Imad et al. [109], however, instead of using heatmap color (in 3-channel RGB), we only use one channel to store the cosine of the incident angle (cos(θ)) of each point cloud. Therefore, a newly formed tensor I_{LID} looks like a heat map-like gray image.

$$I_{LID_{ij}} = \begin{cases} cos(\theta) & \text{if there is a point record in } A_{LID_{xy}} \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

The I_{LID}'s i,j-coordinate has been shifted and scaled from the original A_{LID}'s x,y-coordinate. The center $(x, y) = (0, 0)$ of A_{LID} is at the top of the ego vehicle and the range of the x and y-axis are -32 to 32 meters (the maximum range of LiDAR). Therefore, we need to shift and scale the I_{LID}'s i,j-coordinate so that the center is at $(i, j) = (64, 64)$ and the minimum and maximum coordinate are at $(i, j) = (0, 0)$ and $(i, j) = (127, 127)$ respectively. The shifting and scaling process of the i,j-coordinate from the original x,y-coordinate can be done with (3.20) and (3.21) respectively.

$$I_{LID_i} = \left\lfloor \frac{A_{LID_x} - (-32)}{32 - (-32)} \times 127 \right\rceil \tag{3.20}$$

$$I_{LID_j} = \left\lfloor \frac{A_{LID_y} - (-32)}{32 - (-32)} \times 127 \right\rceil \tag{3.21}$$

Both I_{LID_i} and I_{LID_j} represent the I_{LID}'s i,j-coordinate while A_{LID_x} and A_{LID_y} represent A_{LID}'s x,y-coordinate. Meanwhile, 127 is set to be the highest point of I_{LID}'s i,j-coordinate. Then, we also give a value to the nearest pixels from $I_{LID_{ij}}$ as the same as the pixel's value of $I_{LID_{ij}}$ itself. Thus, the pre-processed I_{LID} will have a better area coverage from the top perspective of view. However, the first method can lose points that have the same A_{LID}'s x,y-coordinate but with a lower A_{LID}'s z-coordinate since the method only stores one point with the highest A_{LID}'s z-coordinate.

In the second LiDAR pre-processing method, we adopt the LiDAR pre-processing technique presented by Yang et al. [110] that takes the A_{LID}'s z-coordinate into account. Then, we stack the pre-processed point clouds with the data from the first method to provide more rich information. The visualization of this method can be seen in Fig. 3.7. Concisely, the second method projects the LiDAR point clouds into a 3D tensor $I_{LID} \in \{0, ..., 1\}^{15\times128\times128}$. Here, we set the number of channels $n(k) = 15$ where $k \in \{0, ..., 14\}$ based on the vertical field of view and the maximum range of the LiDAR sensor. As can be seen in Fig. 3.5, the sensor has a 30^o view below and 20^o view above the horizon line. Since the sensor is placed on the top of the ego vehicle which is 2 meters from the ground, then the lowest point of the point

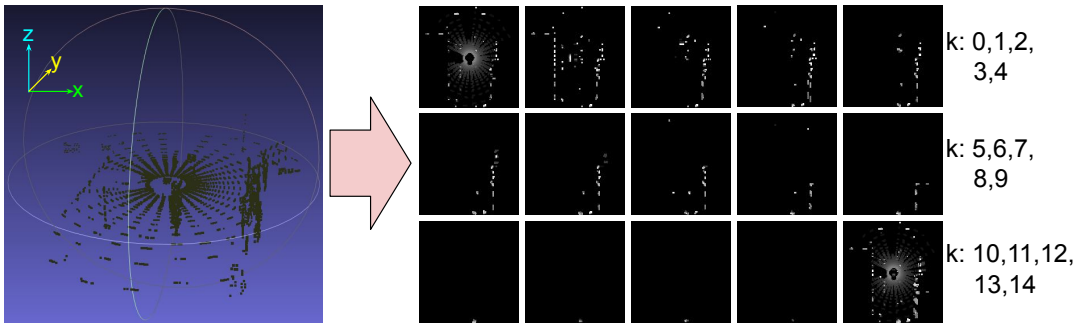


FIGURE 3.7: Point clouds pre-processing.

All point clouds are mapped into a tensor $I_{LID} \in \{0, ..., 1\}^{15\times128\times128}$ where each layer holds the height information that spreads from -2 meter (lowest point) to 11 meters (highest point).

cloud is equal to -2. Meanwhile, the highest point of the point cloud can be calculated with $\lceil sin(20) \times 32 \rceil = 11$. Therefore, we set $n(k) = 15$ where the first 14 channels ($k \in \{0, ..., 13\}$) are used to store point clouds based on their height defined by A_{LID}'s z-coordinate which are spreading from the lowest point of -2 meter to the highest point of 11 meters. Then, the last channel ($k = 14$) is used to store all flattened point clouds from the first method. We map the A_{LID}'s z-coordinate into k channels with (3.22).

$$I_{LID_k} = \left\lfloor \frac{A_{LID_z} - (-2)}{11 - (-2)} \times 13 \right\rfloor, \tag{3.22}$$

where I_{LID_k} is the I_{LID}'s k-coordinate (channels) and A_{LID_z} is the original A_{LID}'s z-coordinate. The multiplier of 13 is used to ensure that there are no point clouds stored in the last channel ($k = 14$) as it has been reserved to store all flattened point clouds from the first method. The maximum and minimum values of A_{LID_z} are set to 11 and -2 for all recorded point clouds so that the scale for all pre-processed I_{LID} will be the same. Thus, the network can easily learn to segment objects based on their height even if the maximum point of each recorded point cloud is different. Finally, similar to the first method, we also give the same value to the nearest pixels around $I_{LID_{ijk}}$. To be noted, if there are 2 points or more with different A_{LID}'s x,y,z-coordinate but have the same I_{LID}'s i,j,k-coordinate after pre-processing, only the point that has the highest A_{LID}'s z-coordinate that can take place on the I_{LID}'s i,j,k-coordinate to prevent multiple data points stored in one coordinate. Therefore, having a large number of k channels would be better since there is more space to store point clouds. However, processing a larger input would also cost more computational time.

On the output side, we read the depth estimation (DE) ground truth as a tensor $I_{DE} \in \{0, ..., 1\}^{1\times128\times128}$. Thus, the output layer of the model for the DE task will only have 1 channel with the spatial dimension of (128×128) that predicts normalized depth values within the range of 0 to 1. With this mechanism, we can use simple ReLU activation for the final output layer of the DE decoder. Meanwhile, the original semantic segmentation (SS), LiDAR segmentation (LS), and bird's eye view projection (BEVP) ground truths are retrieved as $I_{\{SS,LS,BEVP\}} \in \{0, ..., 255\}^{3\times128\times128}$ which are following the color palette in Cityscapes dataset [68]. To meet the needs of the network architecture, especially on its output layers for segmentation-related tasks (SS, LS, and BEVP), we perform one hot encoding process to convert the 8-bit RGB representation. As a result, each ground truth become $I_{SS} \in \{0, 1\}^{23\times128\times128}$, $I_{LS} \in \{0, 1\}^{23\times128\times128}$, and $I_{BEVP} \in \{0, 1\}^{9\times128\times128}$. Therefore, a sigmoid activation

can be used at the output layer of SS, LS, and BEVP decoders. The number of classes in the CARLA simulation dataset is 9 for the BEVP task and 23 for both SS and LS tasks as mentioned in Table 3.1. Meanwhile, in the real-world nuScenes-lidarseg dataset, the number of classes is 32 for all SS, LS, and BEVP tasks as mentioned in Subsection 3.4.2. Hence, the number of channels (axis C) of $I_{\{SS,LS,BEVP\}}$ is also become 32 as well.

3.5 Result and Discussion

To evaluate our proposed methods, some ablation and comparative studies are conducted by comparing all model variants along with the combination of single-task and multi-task models for all given tasks. For sensor fusion experiments, we conduct a comparative study against another MTL model named W-Net [124] that performs the same tasks. We create four W-Net models as there are four different views around the ego vehicle. Meanwhile, for adaptive loss weighting experiments, we configure the comparative study as follows. For depth estimation (DE) and semantic segmentation (SS) tasks, we compare our models with the multi-task GradNorm model [113]. In the LiDAR segmentation (LS) task, we compare our model with PolarNet [111] which performs the same top-view LiDAR segmentation. Finally, for the bird's eye view projection (BEVP) task, we replicate the works by Chen et al. [112] and perform some modifications in their model's final output layer to be a point-wise convolution layer for one-hot encoded prediction so that we can calculate the IoU and perform a fair comparison. The best model is defined by the lowest total metric score as formulated with (3.7). Moreover, as mentioned in Section 3.4, we compare all models on three simulation datasets generated by CARLA simulator [123] and one real-world dataset from nuScenes-lidarseg [77]. Concisely, there are four points that will be disclosed in this Chapter as follows.

- The influence of adding DVS data into the model. As described in Subsection 3.3.1, we create two model variants namely A0 and A1 that are deployed to perform inference on test sets, A and B.
- The influence of providing 15-layer LiDAR data into the model. We compare a model that takes 1 layer of LiDAR data (1L) and a model that takes 15 layers of LiDAR data (15L). Then, we observe its influence based on the TM score.
- The influence of using the MGN algorithm during the training process. By using the MGN algorithm, the model is expected to have better performance as the imbalanced learning problem will be solved by giving appropriate weight to each loss function. Therefore, to understand its effectiveness, a comparative study is conducted on the model with adaptive loss weights (15L+MGN) and the model with static loss weights (15L with $w_i = 1 \forall_i$). We also provide a separate subsection to discuss the behavior of this algorithm.
- A comparative study with the combination of some recent models. We compare all of our model variants as follows. For multi-task SS and DE, we compare A0 and A1 with W-Net [124]. Meanwhile, for multi-task SS, DE, LS, and BEVP, we compare 1L, 15L, and 15L+MGN with the combination of two single-task models and one multi-task model which are PolarNet [111] for LS, Chen et al. [112] for BEVP, and GradNorm model [113] for multi-task DE and SS. Furthermore, we also compute the number of model parameters, model size, GPU memory usage, and inference speed to justify the model efficiency.

TABLE 3.2: Multi-task Performance Score for Comparative Study 1

Dataset	Model	$TM \downarrow$	$MAE_{DE} \downarrow$	$IoU_{SS} \uparrow$	FPS↑
Test A	A0	$0.196 \pm <0.001$	$\mathbf{0.056} \pm <\mathbf{0.001}$	$0.860 \pm <0.001$	**100**
	A1	$0.188 \pm <0.001$	$0.059 \pm <0.001$	$0.871 \pm <0.001$	83
	W-Net [124]	$\mathbf{0.166} \pm <\mathbf{0.001}$	$0.057 \pm <0.001$	$\mathbf{0.891} \pm <\mathbf{0.001}$	58
Test B	A0	$0.193 \pm <0.001$	$0.038 \pm <0.001$	$0.845 \pm <0.001$	**104**
	A1	$\mathbf{0.186} \pm <\mathbf{0.001}$	$\mathbf{0.035} \pm <\mathbf{0.001}$	$\mathbf{0.849} \pm <\mathbf{0.001}$	84
	W-Net [124]	0.220 ± 0.001	$0.038 \pm <0.001$	$0.818 \pm <0.001$	57

The uncertainty on each prediction score is measured by calculating the variance over all inference results. Meanwhile, the speed test is conducted on the same NVIDIA RTX 3090 GPU with batch size = 1 and calculated in frame per second (FPS). The FPS difference on both datasets is caused by the fluctuating GPU condition.

3.5.1 Performance Gain by Feature Fusion

RGB images are usually used as the only input when dealing with semantic segmentation (SS) and depth estimation (DE) tasks. In this Chapter, we study the influence of providing DVS images as the input and fused together with RGB images in the network architecture to leverage the extracted information. As explained in Subsection 3.3.1, we first remove the DVS input block on Fig. 3.2 and named the model as A0 model, then compare its performance with the A1 model that has both RGB and DVS input blocks on its architecture. To clarify the influence of DVS images, we use 2 different datasets, A and B.

Based on the inference result on testing sets shown in Table 3.2, the A1 model also has lower TM scores of 0.188 (set A) and 0.186 (set B). Considering that the A1 model has a better score in all validation and testing sets, it can be said that DVS is giving a positive influence on the model performance. However, as a result of having more encoders to process DVS data, the A1 model inference is slower than the A0 model with an FPS rate of around 83 to 84. Meanwhile, the qualitative result can be seen in Fig. 3.8 where both A0 and A1 models are deployed at night (test set B: Town01, left) and on a cloudy day (test set A: Town02, right). The A1 model seems to have a better result compared to the A0 model. The A1 model is more stable in segmenting rare objects such as poles and the small appearance of surrounding vehicles, especially during poor illumination conditions as it can leverage the information provided by the DVS camera. Meanwhile, both models have comparable performance in solving the depth estimation task.

A further comparative study against another MTL model is conducted to clarify the performance of our proposed model. We compare our model with W-Net [124] which is composed of two serially connected U-Net models [115]. W-Net uses its first U-Net block to perform semantic image segmentation. Then, the prediction is concatenated with the RGB image as the input for the second U-Net block to perform depth estimation. Therefore, W-Net is able to perform both semantic segmentation and depth estimation simultaneously in one forward pass. For a fair comparison, we follow the training configuration described in the W-Net paper to train the model using our datasets. The performance comparison result can be seen in Table 3.2. To be noted, each metric score is averaged across all views as there are four independent W-Net models.

Based on Table 3.2, both A1 and W-Net models can be said to be comparable to each other. On test set A, W-Net performs better than the A1 model with a lower

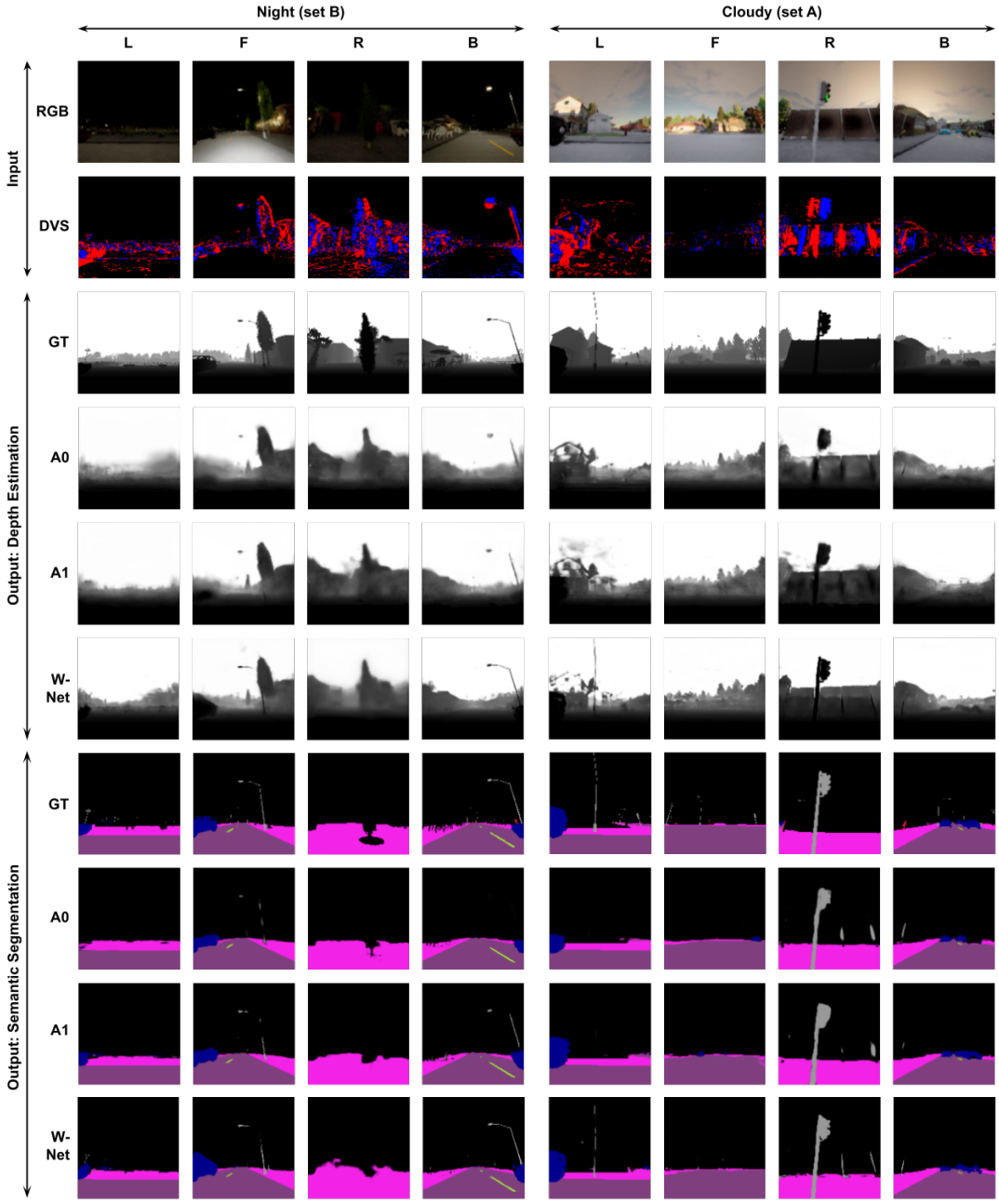


FIGURE 3.8: Inference results on the test images in set A and set B.

Note: F (front); L (left); R (right); B (back); GT (ground truth).

TM score of 0.166. Meanwhile, on test set B, the A1 model surpasses W-Net with a lower TM score of 0.186. Then, as shown in Fig. 3.8, W-Net seems to have a better result, especially when the illumination is enough (set A) as it has much more layers compared to the A1 model. W-Net can estimate and segment very thin objects such as light poles on both left images of depth estimation and semantic segmentation. However, in terms of inference speed, the A1 model is still better with an FPS rate of more than 80 on both datasets. Meanwhile, W-Net only achieves an FPS rate of below 60 when tested with the same device. Therefore, even though the TM score is comparable, it can be said that a single A1 model is preferable as it can perform faster inference compared to the combination of four W-Net models.

TABLE 3.3: Multi-task Performance Score for Comparative Study 2

Dataset	Model	$MAE_{DE} \downarrow$	$IoU_{SS} \uparrow$	$IoU_{LS} \uparrow$	$IoU_{BEVP} \uparrow$	$TM \downarrow$	$MV \downarrow$
	Chen et al. [112]	-	-	-	**0.654 ± <0.001**		
	PolarNet [111]	-	-	**0.483 ± <0.001**	-	1.430	**0.024**
Set	GradNorm† [113]	0.113 ± <0.001	0.546 ± 0.002	-	-		
A	1L	0.090 ± <0.001	0.619 ± 0.001	0.406 ± <0.001	0.613 ± <0.001	1.452	0.032
	15L	**0.083 ± <0.001**	**0.636 ± 0.002**	0.424 ± <0.001	0.575 ± <0.001	1.448	0.032
	15L+MGN	0.084 ± <0.001	0.627 ± 0.002	0.470 ± <0.001	0.594 ± <0.001	**1.393**	0.027
	Chen et al. [112]	-	-	-	0.567 ± 0.003		
	PolarNet [111]	-	-	**0.723 ± <0.001**	-	1.160	0.016
Set	GradNorm† [113]	**0.095 ± <0.001**	0.645 ± 0.003	-	-		
B	1L	0.096 ± <0.001	0.675 ± <0.001	0.621 ± 0.001	0.589 ± 0.002	1.211	0.015
	15L	**0.095 ± <0.001**	**0.682 ± <0.001**	0.680 ± <0.001	0.603 ± 0.001	1.131	0.013
	15L+MGN	0.099 ± <0.001	0.679 ± <0.001	0.704 ± <0.001	**0.630 ± 0.002**	**1.086**	**0.011**
	Chen et al. [112]	-	-	-	**0.637 ± <0.001**		
	PolarNet [111]	-	-	**0.735 ± 0.001**	-	**0.976**	**0.013**
Set	**GradNorm† [113]**	**0.055 ± <0.001**	0.706 ± <0.001	-	-		
C	1L	0.069 ± <0.001	0.751 ± <0.001	0.573 ± 0.001	0.606 ± <0.001	1.138	0.020
	15L	0.062 ± <0.001	**0.765 ± <0.001**	0.645 ± 0.001	0.602 ± <0.001	1.050	0.017
	15L+MGN	0.063 ± <0.001	0.756 ± <0.001	0.678 ± <0.001	0.630 ± <0.001	0.999	0.014
	Chen et al. [112]	-	-	-	0.788 ± <0.001		
	PolarNet [111]	-	-	**0.696 ± <0.001**	-	1.124	0.020
nuScenes	GradNorm† [113]	**0.112 ± <0.001**	0.504 ± 0.005	-	-		
-lidarseg	1L*	0.119 ± <0.001	0.527 ± 0.009	0.597 ± 0.001	0.804 ± <0.001	1.191	0.021
	15L*	0.123 ± <0.001	**0.538 ± 0.007**	0.682 ± <0.001	0.824 ± <0.001	1.079	**0.017**
	15L+MGN*	0.123 ± <0.001	0.536 ± 0.008	0.685 ± <0.001	**0.833 ± <0.001**	**1.069**	**0.017**

†Scores are averaged across all views performed by 4 independent GradNorm models.
*The model only takes extracted feature maps from RGB encoders to perform DE and SS as there is no DVS data in nuScenes-lidarseg.
The higher IoU and the lower MAE, total metric (TM), and metric variance (MV) scores mean the better the model. Be noted, the TM score is used to determine the best model as it represents overall performance on all tasks. Meanwhile, the uncertainty on each metric score is calculated by computing the variance across all inference results on each test set.

3.5.2 1 Layer vs 15 Layers of LiDAR Representation

LiDAR point clouds contain a z-coordinate that represents the height of the object captured by the LiDAR lasers. The idea of our second LiDAR pre-processing method is to differentiate the object based on height data so that the model can leverage this useful information during the training process. As shown in Fig. 3.7, each layer contains a specific object based on its height. For example, the lower layer holds objects which are mostly on the ground such as roads, sidewalks, etc. Then, the middle layer holds other vehicles, pedestrians, etc. Then, the upper layer holds tall objects such as buildings, trees, etc. Finally, stacking all point clouds pre-processed by the first method into the last layer will provide more information.

As shown in Table 3.3, the model that takes 15-layer LiDAR data (15L) has a better performance compared to the model that takes 1 layer only (1L). The comparison between both models is consistent where the 15L model has a lower total metric (TM) score than the 1L model on all test sets. The TM score gets lowered from 1.452 to 1.448 (set A), 1.211 to 1.131 (set B), 1.138 to 1.050 (set C), and 1.191 to 1.079 (nuScenes-lidarseg). Intuitively, adding more layers of information will boost the LS performance as it has inline skip connections from the LiDAR encoder to the LS decoder. This is proven by comparing the IoU_{LS} score where the 15L model has a higher score than the 1L model on all test sets. However, in the BEVP task, both model variants are comparable to each other as the 15L model has higher IoU_{BEVP} scores on dataset B and nuScenes-lidarseg but has lower scores on dataset A and

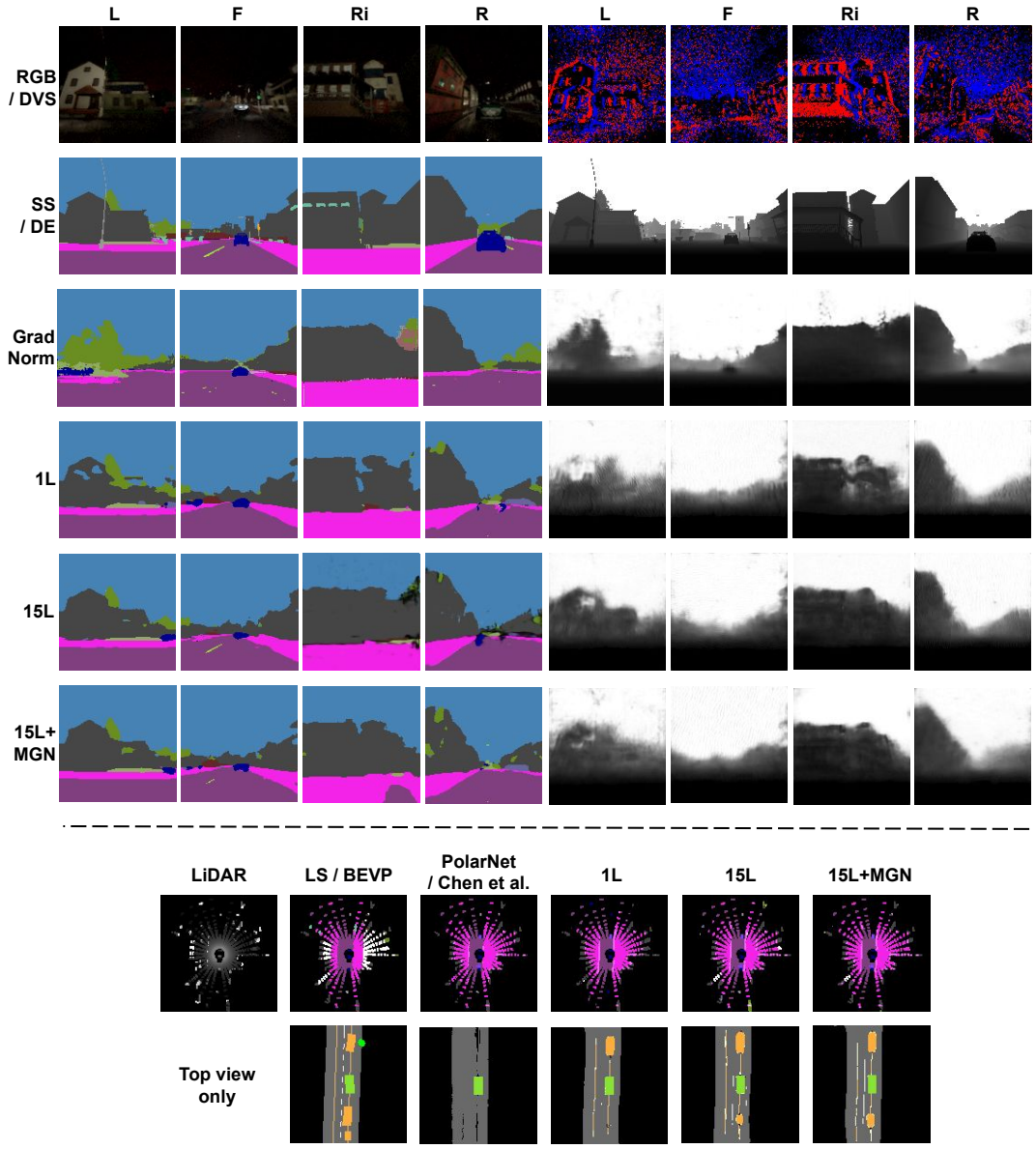


FIGURE 3.9: Inference results on the test images in set C (rainy night).

A qualitative comparison between our model variants (1L, 15L, and 15L+MGN) and combination of STL (single-task learning) and MTL models by Chen et al.'s [112] (BEVP), PolarNet [111] (LS), and GradNorm [113] (DE and SS).

set C. Then, the other interesting thing is the result of DE and SS tasks. Based on MAE_{DE} and IoU_{SS} scores, we found that adding more LiDAR layers is somehow improving DE and SS performance. Consistently, the 15L model has higher IoU_{SS} and lower MAE_{DE} than the 1L model on all simulation datasets, and only the DE performance is degraded on nuScenes-lidarseg. As the pre-processed LiDAR keeps the vertical information (A_{LID_z} to I_{LID_k}) and both RGB and DVS images are naturally at the LiDAR's z-axis, the performance on DE and SS are getting improved. Although there is no specific transformation applied to the network architecture, the 15L model can learn the relationship between shared feature maps. This means that the LiDAR also plays an important role in DE and SS tasks and shows that the 15L model successfully leverages shared feature maps through intermediate fusion.

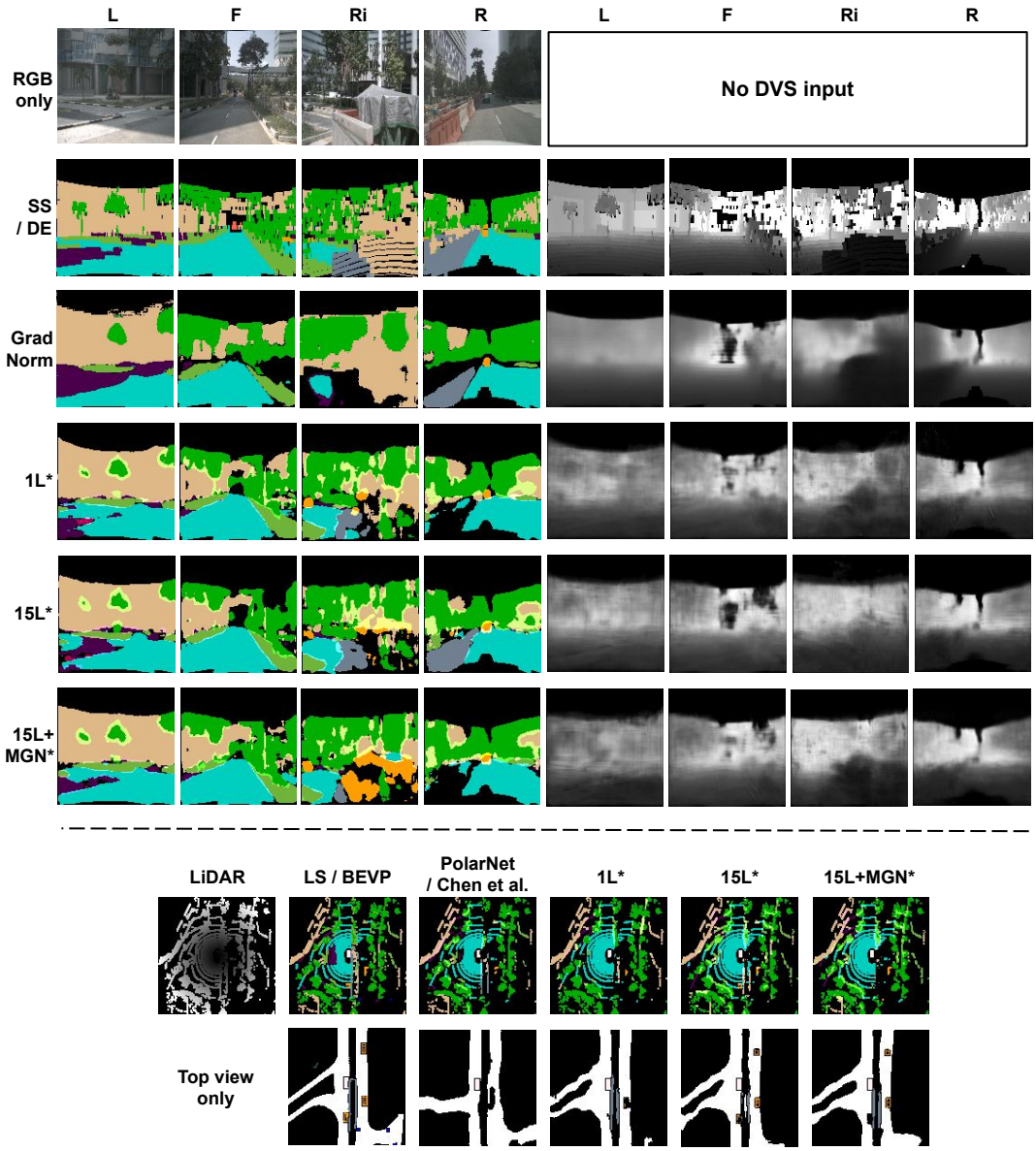


FIGURE 3.10: Inference results on the test images in nuScenes-lidarseg (sunny day).

A qualitative comparison between our model variants without DVS inputs (1L*, 15L*, and 15L+MGN*), and a combination of STL (single-task learning) and MTL models by Chen et al.'s [112] (BEVP), PolarNet [111] (LS), and GradNorm [113] (DE and SS).

Furthermore, based on the qualitative results shown in Fig. 3.9 (rainy night) and Fig. 3.10 (sunny day), the image quality of both 1L and 15L models are comparable on both DE and LS tasks. To be more specific on the model inference on sunny day images (samples from nuScenes-lidarseg), the 15L model performance is quite similar to the 1L model. However, if we take a close look at the rear SS image, the 15L model can segment temporary road barriers successfully while the 1L model cannot. Besides that, on the model inference on rainy night images (samples from set C), the 15L model performs better as it can segment the road lane on the front view SS image. Furthermore, based on the quality of the BEVP images, it is also better at recognizing surrounding vehicles.

3.5.3 Static vs Adaptive Loss Weighting

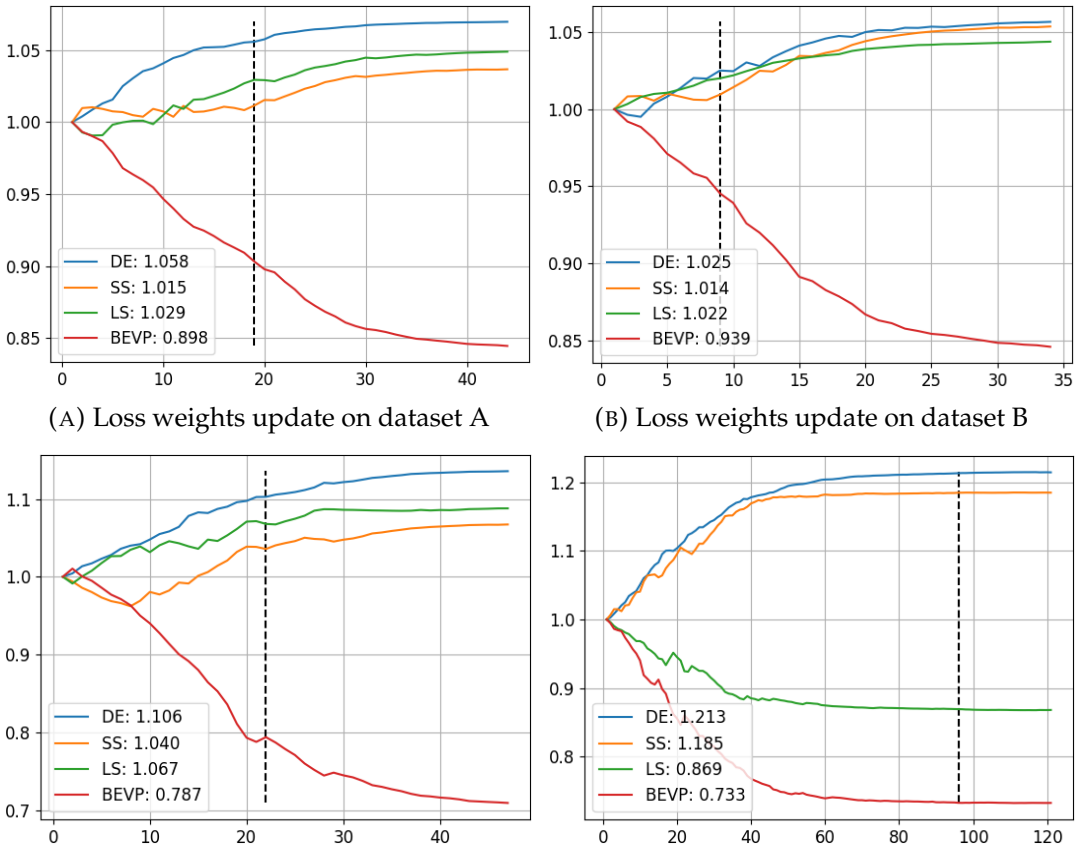
Plenty of tasks can lead to an uneven loss value which depends on what kind of loss function is used. Even if multiple related tasks are handled with the same loss function, it still can lead to imbalanced learning due to the different number of elements and characteristics at the output layer. For instance, the elements in I_{LS} are much larger than in I_{BEVP} and significantly different from I_{SS}. Therefore, a proper set of loss weights is needed to balance the task-learning process. Moreover, it has to be tuned automatically to avoid an expensive computational cost in finding the best combination. Therefore, we propose the MGN algorithm to balance the rate of task learning by tuning each task's loss weight adaptively.

Based on Table 3.3, the model trained with the modified GradNorm (MGN) algorithm (15L+MGN) has a better performance compared to the previous best model with static loss weights (15L) on all test sets. With a consistent result, the total metric (TM) score gets lowered from 1.448 to 1.393 (set A), 1.131 to 1.086 (set B), 1.050 to 0.999 (set C), and 1.079 to 1.069 (nuScenes-lidarseg). However, even with lower TM scores, not all tasks are getting improved by the model. The 15L+MGN variant may have better performance on LS and BEVP tasks where IoU_{LS} and IoU_{BEVP} scores are higher than the 15L variant. However, the 15L model still performs better than the 15L+MGN by achieving lower MAE_{DE} and higher IoU_{SS} on DE and SS tasks respectively. Be noted, the goal of the MGN algorithm is to improve the overall model performance by balancing the rate of learning on each task. Instead of improving the performance of each task, the MGN algorithm is focused on balancing the gradient signal among the tasks. Therefore, the 15L model may still have a better performance on some tasks. In this case, there is a performance trade-off, especially between DE-SS tasks and LS-BEVP tasks. Besides the TM score (3.7), the metric variance (MV) (3.8) can be used to determine the model performance based on the rate of discrepancy between tasks. As can be seen in Table 3.3, the MV of the 15L+MGN model is smaller than the 15L model on all simulation datasets and has the same MV on nuScenes-lidarseg. A lower MV indicates that the model performance on overall tasks is getting balanced with just a little discrepancy.

Based on the qualitative results shown in Fig. 3.9 (a rainy night) and Fig. 3.10 (sunny day), we can see that the 15L+MGN model has a better BEVP performance where it has a more clear projection of a car behind the ego vehicle (sample of a rainy night in dataset C) and a better projection of the roadmap (sample of a sunny day in nuScenes-lidarseg). Meanwhile, the 15L model has a better SS performance on overall views. To be more specific, the 15L model can segment the sidewalk in the right SS image (set C) and the temporary road barriers (nuScenes-lidarseg).

3.5.4 Loss Weighting Behavior

The loss weights update process of the 15L+MGN model during the training phase can be seen in Fig. 3.11. In all simulation datasets, at the time when the model convergence, the modified GradNorm (MGN) algorithm tends to have similar behavior where it gives the highest loss weight to the depth estimation (DE) task followed by LiDAR segmentation (LS) at the second, semantic segmentation (SS) at the third, and bird's eye view projection (BEVP) at the last. Meanwhile, the order between LS and SS is swapped in nuScenes-lidarseg. From the loss weights change behavior, the MGN is penalizing less on the BEVP task so that it will not cause a high imbalance. As shown in the network architecture in Fig. 3.3, the BEVP decoder is placed after the 2^{nd} bottleneck meaning that the network has more focus during training.



(A) Loss weights update on dataset A

(B) Loss weights update on dataset B

(C) Loss weights update on dataset C

(D) Loss weights update on nuScenes-lidarseg

FIGURE 3.11: Loss weights update log.

The vertical black dashed line shows the exact epoch where the model convergence. The vertical axis on each figure is the loss weight while the horizontal axis is the epoch.

The gradient produced from BEVP also affects the decoder of SS, DE, and LS tasks on the previous layer. Thus, it makes sense that the MGN penalizes the BEVP task less than the other tasks. For tasks that are placed before the 2^{nd} bottleneck, the DE task loss calculated with the Huber loss function (3.1) produces a smaller loss value compared to LS and SS losses which are calculated by BCEDice loss function (3.3). Hence, the MGN algorithm gives a higher loss weight to the DE task to compensate for the imbalance. With this mechanism, the network will not lose its focus on the DE task learning while still maintaining progress on learning the other tasks.

Moreover, even if the number of elements in I_{LS} is equal to I_{SS}, both tasks have a significant difference in characteristics. In I_{LS}, many elements are filled with 0 since there are plenty of vacant points caused by LiDAR lasers limitation. Meanwhile, in I_{SS}, many elements are filled with 1 representing the one-hot object class on each tensor channel and have a strong correlation as all points are captured by the RGB camera. As a result, $\mathcal{L}_{SS}$ tends to be bigger than $\mathcal{L}_{LS}$ even when computed with the same loss function. Therefore, to balance the task learning, MGN gives higher weight to the LS task than to the SS task. However, the loss weighting behavior on SS and LS tasks is not consistent in the real-world dataset. The weight order is swapped between those tasks as the characteristic nuScenes-lidarseg is different from the simulation dataset. To be noted, the sum of all loss weights will always be equal to $T = 4$ as they are normalized with (3.17) at the end of each epoch.

3.5.5 Single-task vs Multi-task Models

We also conduct further model testing by comparing our models with some recent models for each task. In the multi-task depth estimation (DE) and semantic segmentation (SS) comparison, we train the original GradNorm model [113] for each view so that there are four models in total. We use the GradNorm SegNet [125] version with VGG16 encoder [126] and symmetric task decoders for comparative study as the GradNrom authors only use this model for in-depth analysis. The training setup is configured to be the same as described in the GradNorm paper where Adam optimizer [127] along with pixel-wise cross-entropy and squared losses are used to train the model. Then, in the LiDAR segmentation (LS) comparison, we train PolarNet [111] to take our pre-processed LiDAR point clouds. We use the same training configuration written in the provided code as the author did not mention the detail in their paper. Concisely, Adam optimizer [127] along with cross-entropy loss is used to train the model until convergence. Finally, in the bird's eye view projection (BEVP) comparison, we replicate Chen et al.'s model [112] that takes front RGB and top-view LS images as the input. The model has a BEVP decoder to perform BEVP and input reconstruction modules that reconstruct the front RGB and LS images. However, LS input and BEVP output are represented in RGB image representation $\{0, ..., 255\}^{3\times128\times128}$. This kind of representation is not suitable for segmentation-related tasks. Thus, we change it into a one-hot encoded image, so that each of them is represented as $\{0, 1\}^{C\times128\times128}$ where C is the number of possible classes. Then, we put 1 extra point-wise (1×1) convolution layer and a sigmoid activation at the last layer of the LS input reconstruction module and BEVP decoder. With this modification, the metric function IoU (3.6) can be calculated for comparison purposes. Furthermore, besides comparing all metric scores, we also compare the number of model parameters, model size, GPU memory utilization, and inference speed to measure how efficient the model is.

Based on Table 3.3, our best model variant (15L+MGN) is better than the combination of Chen et al., PolarNet, and GradNorm models. In small and large datasets, the 15L+MGN model has lower total metric (TM) scores of 1.393 (set A), 1.086 (set B), and 1.069 (nuScenes-lidarseg). Meanwhile, the other variants still maintain a comparable performance with a small gap. However, in the medium dataset (set C), the combination has a better performance with a TM score of 0.976. Independently, PolarNet consistently gives a better LS performance by achieving the highest IoU_{LS} in all datasets. Meanwhile, Chen et al. and GradNorm models are still comparable in BEVP, DE, and SS tasks. However, based on the qualitative results shown in Fig. 3.9 (a rainy night) and Fig. 3.10 (sunny day), both Chen et al. and GradNorm models are missing the surrounding vehicles. On BEVP images, Chen et al.'s model is able to locate the occupied area by the surrounding vehicles, but cannot segment the vehicle correctly (set C). It also cannot project the local roadmap as well as our models (nuScenes-lidarseg). Then, as shown on SS images, GradNorm is facing difficulties in segmenting vehicles on a rainy night. On the other hand, our model faces the same difficulties, but it can locate the occupied region properly. During rainy conditions, the DVS sensor is distracted by the raindrops. As a result, the DE performance of our model is getting degraded. Then, as shown on LS images, both PolarNet and our models have a similar performance where both models can segment all objects from the top-view perspective and locate the corresponding pixel class nearly the same as in the ground truth. Then, based on Table 3.4, our models have much fewer parameters where they only have less than 2% of the total parameters owned by the combination. Even with that small number of parameters, the 15L+MGN model

TABLE 3.4: Model Specification

Dataset	Model	Parameters↓	Total Parameters↓	Size↓	Total Size↓	GPU Usage↓	FPS↑
Simulation (A, B, C)	Chen et al. [112]	10365211		83.061			
	PolarNet [111]	13403701	136728752	107.342	1094.867	1987	44.472
	4×GradNorm† [113]	4×28239960		4×226.116			
	1L	2519488	2519488	20.549	20.549	1049	57.117
	15L	2521504	2521504	20.565	20.565	1049	56.018
	15L+MGN	2521504	2521504	20.565	20.565	1049	54.428
nuScenes -lidarseg	Chen et al. [112]	10372539		83.120			
	PolarNet [111]	13404286	136757437	107.347	1095.099	2025	48.236
	4×GradNorm† [113]	4×28245153		4×226.158			
	1L*	2275604	2275604	18.557	18.557	1015	65.969
	15L*	2277620	2277620	18.574	18.574	1017	65.426
	15L+MGN*	2277620	2277620	18.574	18.574	1017	65.192

†The number of parameters and model size of the GradNorm model are multiplied by 4 as there are 4 models in total.
*These model variants do not have DVS encoders as there are no DVS images recorded in the dataset.
For a fair comparison, we use the same GPU device (NVIDIA GTX 1080 Ti) to run all models with batch size = 1. However, the inference speed measured in frames per second (FPS) is slightly different on each dataset due to the fluctuating GPU performance. Therefore, we average the FPS over A, B, and C datasets for the inference on simulation data. We separate the measurement on nuScenes-lidarseg as it has different characteristics. Thus, there is a small change in the number of parameters, model size (in MB), and GPU usage (in MB).

maintains a better performance with less GPU memory utilization. Hence, it has a smaller size and can infer faster with a speed of around 54 frames per second (FPS) on simulation datasets and 65 FPS on a real-world dataset. Considering the performance result shown in Table 3.3 and Table 3.4 along with the qualitative result shown in Fig. 3.9 (a rainy night) and Fig. 3.10 (sunny day), it can be said that our model is better and more efficient than the combination. Moreover, our model is preferable due to its compactness and smaller size.

The reason why our model can outperform the combination even with fewer parameters is that it can leverage feature sharing on its encoders that process multiple views of input to efficiently learn the features. The network architecture makes it possible for each decoder to take the advantage of the extracted features from each encoder. Besides that, our proposed techniques are also playing an important key in boosting the model performance and keeping the performance balanced. Based on Table 3.3, without using these methods, our model cannot be better than the combination. Our 15L+MGN model may win on datasets A, B, and nuScenes-lidarseg but lose on dataset C with a TM score gap of 0.023. However, if we take a close look at the scores on dataset C, the lowest TM score obtained by the combination is mostly influenced only by the outstanding performance of PolarNet which achieves IoU_{LS} of 0.735. In fact, PolarNet has always maintained to be the best on the LS task in all datasets. In dataset C, PolarNet outperforms our model with an IoU_{LS} gap of 0.057 which is the largest among all experiments. If the gap on IoU_{LS} is similar to the gap in other datasets, our model might have won on dataset C. PolarNet has what is called "ring-connected CNN" that is specifically used to process LiDAR data. Therefore, with a larger number of learnable parameters, it is more capable of capturing more useful features in varying areas. Be noted, based on the number of towns used on each dataset, it can be said that dataset C is more varied as it contains five different maps while datasets A, B, and nuScenes-lidarseg only have two maps, and both are similar to each other.

3.6 Findings

In this Chapter, we develop a compact deep multi-task learning (MTL) model to perform various driving perception tasks simultaneously in one forward pass. Through data pre-processing and multi-sensor fusion techniques, the model can process and combine multiple input modalities. In addition, we propose an adaptive loss weighting algorithm to tackle the imbalanced learning issue and boost overall performance. To understand the influence and behavior of our proposed methods, an ablation experiment is conducted by creating several variants. Finally, a comparative study against the combination of some recent models is conducted to clarify performance and efficiency. Based on the ablation and comparative results on both simulation and real-world datasets, we disclosed several findings as follows.

- Fusing both RGB and DVS images will boost the overall model performance since the model can take more distinctive information from both RGB and DVS encoders. This is supported by the comparison result between A0 and A1, where A1 outperforms A0 in semantic segmentation (SS) and depth estimation (DE) tasks.

- Keeping the height information of the LiDAR point clouds A_{LID}'s z-coordinate, the overall model performance is improved, especially in the LS task that has direct skip connections from the LiDAR encoder. This is proven by the 15L model which has a better performance compared to the 1L model. Moreover, with rich vertical features given from the LiDAR encoder through intermediate fusion at the first bottleneck, the 15L model gains better performance on DE and SS tasks.

- The MTL process is successfully balanced and results in a better model by using the modified GradNorm (MGN) algorithm to update the loss weights adaptively based on the gradient signal. Based on the comparison result, the 15L+MGN model performs better than the 15L model where it has lower total metric (TM) and metric variance (MV) scores.

- The MGN algorithm makes a better trade-off between DE-SS tasks with LS-BEVP tasks. Based on the loss weighting behavior, the MGN algorithm tends to penalize less on the task that has a higher focus by default such as the bird's eye view projection (BEVP) task that is placed at the end of the network. This algorithm is also capable of compensating for small or large losses produced by different loss functions with varying output elements. As evidence, the depth estimation (DE) loss computed with the Huber loss function has a bigger loss weight than LiDAR segmentation (LS) and semantic segmentation (SS) loss computed with the BCEDice loss function.

- Based on the comparative study with the combination of some recent models, our best model variant (15L+MGN) maintains better performance even with much fewer parameters. Hence, it can infer faster and consume less GPU memory which is preferred for further deployment.

Chapter 4

Simulation-based End-to-end Autonomous Driving

Continuing our work described in Chapter 3, we propose a novel deep learning model called DeepIPC (Deeply Integrated Perception and Control) that is trained with end-to-end and multi-task learning manners to perform both perception and control tasks simultaneously. Focusing on the task of point-to-point navigation for autonomous driving, the model is deployed to drive an ego vehicle safely by following a sequence of routes defined by the global planner. The perception part of the model is used to encode high-dimensional observation data provided by an RGBD camera while performing semantic segmentation, semantic depth cloud (SDC) mapping, and traffic light state and stop sign prediction. Then, the control part decodes the encoded features along with additional information provided by the GNSS receiver and speedometer to predict waypoints that come with a latent feature space. Furthermore, two agents are employed to process these outputs and make a decision that determines the level of steering, throttle, and brake as the final action. DeepIPC is evaluated with CARLA simulator that simulates various scenarios made of normal-adversarial situations and different weathers to mimic real-world conditions. In addition, we conduct a comparative study with some recent models to justify the performance in multiple aspects of driving. Moreover, we also conduct an ablation study on SDC mapping and multi-agent to understand their impact on the model. As a result, DeepIPC achieves the highest driving score even with fewer parameters and computation load.

4.1 Motivation

Currently, the challenge that remains for an end-to-end model is how to encode or extract useful features so that the controller module can decode them into proper navigational controls. Obviously, the perception module needs to be supported with many kinds of information that represent the detailed condition around the ego vehicle. In this case, sensor fusion-based models have been proven to achieve better performance as they use various kinds of sensors to gather more detailed information [128] [129]. However, a huge computation load is inevitable as bigger models are needed to process large data. Moreover, a data pre-processing technique is also necessary as varying sensors often come with different data modalities [130] [131]. Even with a good set of encoded features, there is still another challenge that remains for an end-to-end model that is the controller module can be stuck with a certain behavior due to its learning experience from limited driving records. Therefore, more decision-makers may be needed to translate extracted information into a

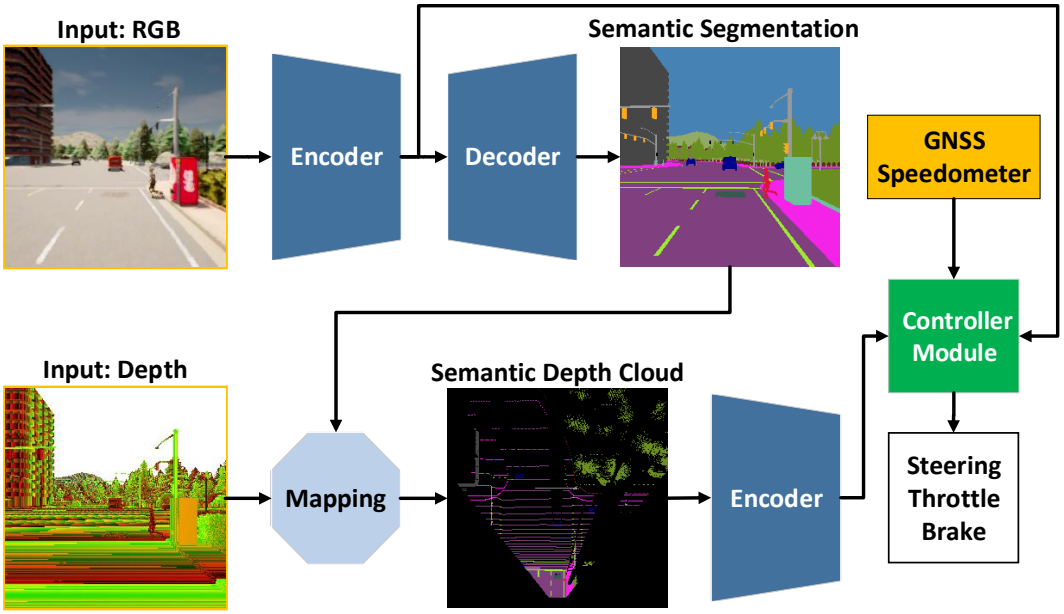


FIGURE 4.1: The process flow inside DeepIPC.

DeepIPC takes a set of input data provided by RGBD camera, GNSS receiver, and speedometer to perceive the environment and drive the ego vehicle. A more detailed network architecture can be seen on Fig. 4.2.

different aspect of driving control [132] [133]. Furthermore, the imbalance of learning during the training process could be another issue since the solution for both perception and control tasks is learned simultaneously. Hence, a method to balance the training signal is also necessary to ensure that all tasks are learned at the same pace [134] [135].

To answer those challenges, we propose an end-to-end deep multi-task learning model namely DeepIPC, which stands for Deeply Integrated Perception and Control as illustrated in Fig. 4.1. This model is made of two main modules, the perception module (blue) and the controller module (green) that are deeply connected inside the network architecture. As the main input for its perception module, DeepIPC takes RGB images and depth maps of the front view. Thanks to the rapid development of sensor devices, both observation data can be provided with a single RGBD camera so that there is no need to mount more sensors on the ego vehicle [136] [137] [138] [139]. Besides that, RGB images and depth maps also have similar dimensions and representations so that both data can be processed easily. Meanwhile, the controller module is responsible for decoding the extracted features from the perception module along with additional information on route location and measured speed provided by the GNSS receiver and speedometer. By using two different agents, more varied navigational controls can be made considering multiple aspects of driving [140] [141] [142]. Furthermore, we design the model to only have a small number of neurons or trainable parameters to reduce the computation load footprint. We consider imitation learning, especially the behavior cloning technique as it can leverage the use of large-scale datasets to train the model to near-human standard [143]. Finally, we use an adaptive loss weighting algorithm namely modified gradient normalization (MGN) to tackle the imbalance learning problem by weighting the training signal [54]. Therefore, the model can be prevented from tending to focus only on a single task during the training process.

4.2 Related Work

We review some related works in the area of end-to-end autonomous driving with the imitation learning approach. Then, we explain how they inspire our work.

4.2.1 End-to-end Multi-task Model

There are two main advantages of a model to be trained in end-to-end and multi-task learning manners. With an end-to-end learning fashion, there are no additional settings needed to integrate all submodules so that such kinds of information loss and human error can be avoided. Then, with a multi-task learning (MTL) strategy, the model can leverage shared features to speed up the training process. In the field of autonomous driving research, Ishihara et al. [71] have demonstrated the usefulness of training a model in end-to-end and multi-task manners. Similar to CILRS [144] (conditional imitation learning with ResNet [145] and speed input) model, their model takes front RGB image, speed measurement, and discrete high-level navigational command to predict the level of steering, throttle, and brake used to drive the vehicle. In addition, the MTL approach to depth estimation, semantic segmentation, and traffic light state prediction is used to improve the quality of extracted features in the perception module. With better features, the controller module is expected to be better at determining navigational controls. A similar imitation learning-based approach has been studied by Chen et al. [146] where the same set of inputs is used to drive the vehicle. However, instead of using discrete high-level navigational commands directly, the model produces a set of waypoints used by two PID controllers to drive the vehicle.

For our work, we take the idea of performing multiple perception tasks of semantic segmentation and traffic light state prediction as extra supervision demonstrated by Ishihara et al. [71] to guide the perception module in producing better features for the controller module. However, instead of performing depth estimation, we use a depth map provided by the RGBD camera as an input to the model which opens the possibility of sensor fusion strategy in performing better scene understanding [48] [131] [53]. Another issue that needs to be addressed in the multi-task learning approach is the imbalanced learning problem where the model may tend to focus on a certain task only [94] [134] [135]. To address this issue, we use an adaptive learning algorithm called modified gradient normalization to ensure that all tasks are learned at the same pace [54].

4.2.2 Sensor Fusion Strategy

By using the sensor fusion technique, a model can have a better scene understanding as it opens plenty of possibilities to perceive the environment in multiple kinds of representation. In the field of autonomous driving research, Huang et al. [86] have proposed a sensor fusion-based model that takes an RGB image and depth map to capture a deeper global context in front of the vehicle. Both inputs are fused at an early stage to form low-dimensional latent features. The extracted features are processed by the controller module to determine a set of actions. Similar to Ishihara et al. [71] and Chen et al. [146], this approach also uses navigational commands to drive the vehicle. The sensor fusion technique also opens the possibility of perceiving the environment from a different perspective. Prakash et al. [80] has developed a sensor fusion-based model that takes an RGB image and pre-processed LiDAR point clouds. The RGB image contains information on the front-view perspective

while the LiDAR contains information on the top-view or bird's eye view (BEV) perspective. By fusing both features, the model can perform a better scene understanding [81] [147]. Moreover, unlike most current works, this model does not use high-level navigational commands to drive, instead, it takes sparse GNSS location of predefined routes provided by a global planner.

For our work, we also use a combination of an RGB image and a depth map provided by a single RGBD camera. However, instead of extracting depth features at the early stage, we project the depth map and perform semantic depth cloud (SDC) mapping with a BEV perspective. Therefore, the model can take advantage of perceiving the environment from the top-view perspective [80] [81] [147]. Moreover, since the SDC map stores semantic information, the model will have a better understanding as the occupied or traversable regions become clearer than pre-processed LiDAR point clouds which only contain height information. We also consider using a sequence of routes instead of high-level navigational commands as it makes more sense for driving an autonomous vehicle in real-world conditions [148] [149] [150].

4.3 Methodology

In this section, we describe the details of the network architecture of DeepIPC that acts as a pilot for driving an ego vehicle autonomously. Then, we explain the data generation process and data representation. Furthermore, we also define the training configuration including the formulation of the loss function.

4.3.1 Proposed Model

As shown in Fig. 4.2, DeepIPC is composed of two main modules, the perception module (blue) and the controller module (green). Concisely, the perception module is responsible for complex scene understanding and providing useful information to the control module. Specifically, the perception module performs semantic segmentation, semantic depth cloud (SDC) mapping in a bird's eye view (BEV) perspective, traffic light state prediction, and stop sign prediction. Then, the controller module leverages the given information in the form of encoded features together with additional inputs of current speed measurement and the GNSS location of the route. This module provides waypoints and navigational controls as the final outputs.

Perception Module

The perception module takes an RGB image and depth map provided by a single RGBD camera as the main inputs. We consider a region of interest (ROI) of 256×256 at the center for a fair comparison with other models (see Section 4.5 for more details). Another purpose is to eliminate the distortion at the corner that causes RGB images and depth maps to have a different appearance. As shown in Fig. 4.2, we begin the perception phase with ImageNet normalization on the RGB image since we use EfficientNet version B3 [151] pre-trained on ImageNet [152] as the RGB encoder. EfficientNet is chosen as it can perform excellently on many vision-related tasks with a small number of parameters. Then, the extracted feature maps are learned by the segmentation decoder to perform semantic segmentation in 23 different classes mentioned in Table 4.1. The decoder is made of multiple convolution blocks ($2\times$(3×3 convolution + batch normalization [116] + ReLU [117]) + bilinear interpolation) and a final pointwise 1×1 convolution with sigmoid activation. It is also enhanced with different scales of extracted feature maps from the encoder

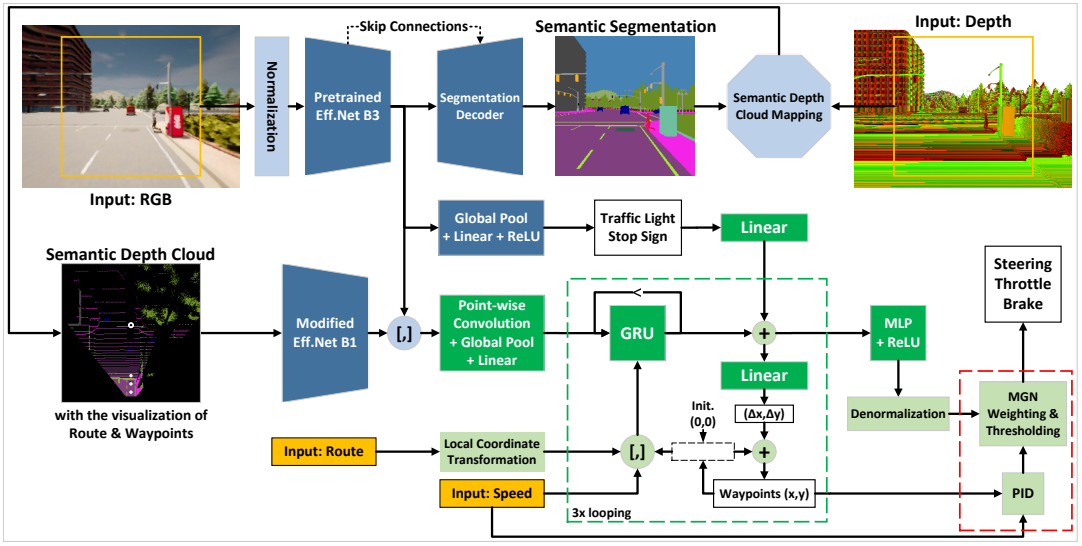


FIGURE 4.2: The architecture of DeepIPC.

The light-colored items are not trainable. Blue-colored items are considered as the perception module while green-colored items are the controller module. The process inside the dashed green line box is looped over three times. Meanwhile, items inside the dashed red line box are only used for driving. Therefore, the model predicts waypoints and estimates the level of steering, throttle, and brake separately during the training process. Inside the semantic depth cloud, the route is represented with a white hollow circle, while the waypoints are represented with small white circles.

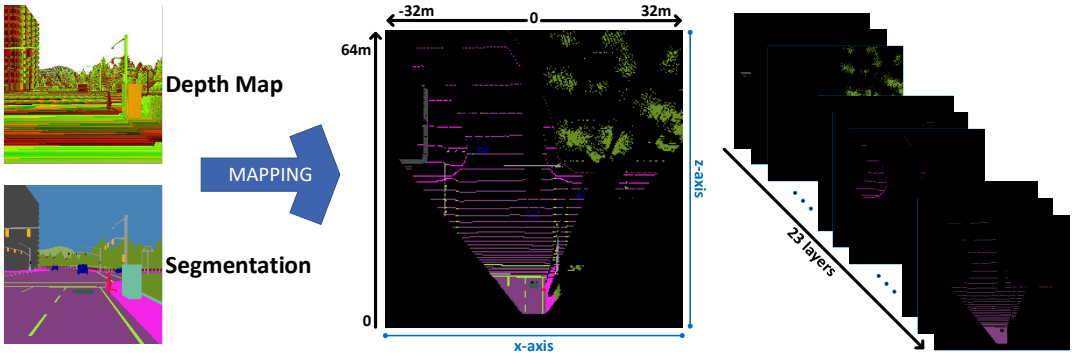


FIGURE 4.3: Semantic depth cloud mapping.

Each layer holds one unique class mentioned in Table 4.1.

connected with some skip connections [53] [153]. In addition, we also create a separate module consisting of a global average pooling layer, linear layer, and ReLU to specifically predict the traffic light state and stop sign. Although both tasks can be considered classification problems, we choose ReLU rather than sigmoid since we want to avoid information loss as the prediction outputs will be encoded later in the controller module for extra supervision.

In addition to the front-view perspective, the scene understanding capability can be improved further by providing more information from the BEV perspective. Thus, the agent can have a better capability in estimating the traversable regions. Hence, we perform semantic depth cloud (SDC) mapping using segmentation prediction and depth map as shown in Fig. 4.3. However, we ignore the height information (y-axis) as we consider the BEV perspective which relies on the x-axis and z-axis

(given by depth map). Therefore, if there are multiple object classes corresponding to one point on the 2D plane, then the object that has the highest location is selected. The SDC mapping process can be summarized as follows.

- Define the distance range of 64 meters to the front and 32 meters to the left and right forming a coverage area of 64×64 square meters. This means that the ego vehicle is always positioned at the bottom center. Then, define the spatial dimension of the SDC tensor $H \times W = 256 \times 256$, so that one element is equal to an area of 25×25 square centimeters.
- Get segmentation tensor $\mathcal{S}$, depth tensor $\mathcal{D}$, transformation matrix for x-axis $\mathcal{T}_x$ (formed with camera parameter f_x and ROI size of 256×256).
- Compute x-axis ($\mathcal{P}_x$) and z-axis ($\mathcal{P}_z$) coordinates and normalize them to match the spatial dimension of the SDC tensor with (4.1) and (4.2).

$$\mathcal{P}_x = \left\lfloor \frac{(\mathcal{D} \times \mathcal{T}_x + 32)}{64} \times 255 \right\rceil, \tag{4.1}$$

$$\mathcal{P}_z = \left\lfloor \left(1 - \frac{\mathcal{D}}{64}\right) \times 255 \right\rceil, \tag{4.2}$$

  Keep in mind that the tensor index starts from 0, hence, we use 255 instead of 256. Unlike the standard cartesian coordinate system, any of (x,0) coordinates are located at the top of the 2D plane. Thus, we do mirroring and shifting $(1 - \frac{\mathcal{D}}{64})$ on $\mathcal{P}_z$ computation.
- Project every element containing a certain semantic class in $\mathcal{S}$ to a 256×256 matrix based on $\mathcal{P}_x$ and $\mathcal{P}_z$.
- Finally, apply one-hot encoding to obtain SDC tensor $\mathbb{R} \in \{0,1\}^{23 \times 256 \times 256}$, where 23 is the channel representing the number of classes and 256×256 is the spatial dimension.

With this representation, DeepIPC can perceive better to drive the ego vehicle safely in the environment. We use a smaller variant of EfficientNet named B1 [151] for the SDC encoder. There is no need to use the same or even a bigger variant than the RGB encoder as the SDC already contains distinctive information. Since the SDC contains 23 layers of semantic segmentation classes, we modify the first convolutional layer to receive a tensor with 23 channels. Then, the entire encoder is initialized with Kaiming initialization [120] to catch up with the RGB encoder during the training process. Finally, both RGB and SDC features are concatenated to form a tensor with the size of $C \times H \times W = (1536 + 1280) \times 8 \times 8$, where 1536 and 1280 are the number of channels in RGB and SDC features, respectively. To be noted, the RGB features have more channels than SDC features as it is extracted with a bigger EfficientNet variant. Meanwhile, the spatial dimension of 8×8 is a result of multiple downsampling through the EfficientNet architecture.

Controller Module

The controller module is used to decode concatenated RGB and SDC features given by the perception module. We begin the control phase by employing a fusion block composed of a point-wise (1×1) convolution layer, global average pooling layer, and linear layer to process the features. The point-wise convolution layer is used

to fuse and learn the relation of each feature element along the channel axis and results in a smaller feature tensor with a size of $C \times H \times W = 384 \times 8 \times 8$. Then, the global average pooling layer is used to obtain the global context by averaging 8×8 array on each channel. Meanwhile, the linear layer is used to reduce the number of feature elements from 384 to 232, so that the upcoming layers do not need to process an enormous number of neurons to save computational load.

We use a gated recurrent unit (GRU) introduced by Cho et al. [83] to further process the features from the fusion block. In the network architecture, GRU is chosen as the model needs to keep relevant information for the waypoints prediction that relies on the previous prediction stored in the GRU memory. Moreover, GRU has been proven to be able to eliminate the vanishing gradient problem in a standard recurrent neural network (RNN) using so-called update and reset gates. Besides that, GRU has a better performance-cost ratio than the other RNN layers as it can be trained faster [154]. Since there are three waypoints that will be predicted, the process inside the dashed green line box on Fig. 4.2 is looped over three times during the forward pass. In the first loop, GRU takes the features as an initial hidden state and uses the current waypoint coordinate in BEV space (local vehicle coordinate), route location coordinate transformed to BEV space, and current speed (measured in m/s) as the inputs. To be noted, the initial value for the current waypoint coordinate is the vehicle's local coordinate which is always at (0,0) positioned at the bottom-center of the semantic depth cloud (SDC) map. Then, global (x_g,y_g) to local (x_l,y_l) coordinate transformation can be done with (4.3).

$$\begin{bmatrix} x_l \\ y_l \end{bmatrix} = \begin{bmatrix} \cos(90+\theta_v) & -\sin(90+\theta_v) \\ \sin(90+\theta_v) & \cos(90+\theta_v) \end{bmatrix}^T \begin{bmatrix} x_g - x_{vg} \\ y_g - y_{vg} \end{bmatrix} \tag{4.3}$$

The relative distance is the gap between the route's global coordinate (x_g,y_g) and the vehicle's global coordinate (x_{vg},y_{vg}). We use $90 + \theta_v$ (vehicle rotation degree) in the rotation matrix since the GNSS compass is oriented to the north. Then, the next hidden state coming from the GRU is biased by the prediction of the traffic light state and stop sign which is encoded by a linear layer and added through element-wise summation. For waypoints prediction, we use a linear layer to decode the biased hidden state into Δx and Δy. Then, the next waypoint can be obtained with (4.4).

$$x_{i+1}, y_{i+1} = (x_i + \Delta x), (y_i + \Delta y), \tag{4.4}$$

where i is the current step in the loop process (dashed green line box). On the next loop, the current hidden state (before biased by traffic light state and stop sign) is taken by the GRU to predict the next hidden state with the first waypoint replacing the vehicle's local coordinate (0,0) as the input (along with the same route location and speed measurement). At the end of the looping process, there will be three predicted waypoints and a latent space biased by traffic light state and stop sign prediction. As shown in Fig. 4.2, we use two agents that act as the final decision-makers. The first agent is a multi-layer perceptron (MLP) network composed of two linear layers and a ReLU that decodes the latent space into a set of navigational controls in a normalized range of 0 to 1 (see Subsection 4.3.2 for more details). The second agent is two PID controllers (lateral and longitudinal) that compute the predicted waypoints along with the current speed into a set of navigational controls as summarized in Algorithm 4.1. We set Kp, Ki, Kd parameters for each PID controller as the same as Chen et al. [146] and Prakash et al. [80]. Each agent computes diverse levels of steering, throttle, and brake. Then, the final control actions that actually drive the ego vehicle are made by a control policy defined in Algorithm 4.2.

Algorithm 4.1: PID agent

$\Theta = \frac{\omega\rho_1 + \omega\rho_2}{2}$; $\theta = \tan^{-1}\left(\frac{\Theta[1]}{\Theta[0]}\right)$; $\gamma = 2 \times ||\omega\rho_1 - \omega\rho_2||_F$
PID steering = Lateral PID(θ)
PID throttle = Longitudinal PID($\gamma - \nu$)

θ: heading angle based on first and second waypoints, $\omega\rho_1$ $\omega\rho_2$
γ: desired speed, 2 $\times$ Frobenius norm between $\omega\rho_1$ and $\omega\rho_2$
ν: current speed measured by speedometer

Algorithm 4.2: Control Policy

if *MLP throttle* $\geq$ *0.2 and PID throttle* $\geq$ *0.2* **then**
  steering = β_{00}MLP steering + β_{10}PID steering
  throttle = β_{01}MLP throttle + β_{11}PID throttle
  brake = 0
else if *MLP throttle* $\geq$ *0.2 and PID throttle* $<$ *0.2* **then**
  steering = MLP steering; throttle = MLP throttle; brake = 0
else if *MLP throttle* $<$ *0.2 and PID throttle* $\geq$ *0.2* **then**
  steering = PID steering; throttle = PID throttle; brake = 0
else
  steering = 0; throttle = 0; PID brake = 1
  brake = β_{02}MLP brake + β_{12}PID brake

$\beta \in \{0, ..., 1\}^{2\times3}$ is a set of control weights initialized with:
$\beta_{00} = \frac{\alpha_4}{\alpha_4+\alpha_7}$; $\beta_{10} = 1 - \beta_{00}$; $\beta_{01} = \frac{\alpha_5}{\alpha_5+\alpha_7}$; $\beta_{11} = 1 - \beta_{01}$; $\beta_{02} = \frac{\alpha_6}{\alpha_6+\alpha_7}$; $\beta_{12} = 1 - \beta_{02}$
where $\alpha_4, \alpha_5, \alpha_6, \alpha_7$ are loss weights for steering, throttle, brake, and waypoints which are tuned by the MGN algorithm [54] (see Subsection 4.3.3)

There are two reasons why we predict three waypoints even though only two of them are used by the PID agent. First, the last waypoint prediction ensures that the final hidden state contains the information of the second waypoint which is being used by the GRU as its input. Therefore, it makes the MLP agent act based on the same information as provided to the PID agent. Second, by predicting an extra waypoint, neurons in the GRU and the waypoint prediction layer can have more learning experiences. As described in Algorithm 4.2, to make DeepIPC becomes more cautious in driving the ego vehicle, we only consider full brake and set the minimum active threshold for the throttle level to 0.2 for each agent.

4.3.2 Behavior Cloning

We consider imitation learning, especially behavior cloning where the goal is to learn a policy π by mimicking the behavior of an expert with a policy π^* [44] [143]. We define the policy as a mapping function that maps inputs to waypoints, steering, throttle, and brake levels which can be approximated with a supervised learning paradigm. Therefore, we use CARLA simulator [123] to generate a dataset for training and validation. As described in Table 4.1, we use all available maps and weather presets to create a more varying simulation environment. We also spawn non-player characters (NPC) to mimic real-world conditions. In generating the dataset, an ego vehicle driven by an expert with privileged information is rolled out to retrieve a set of data for every 500ms. One set of data consists of an RGB image and depth map along with semantic segmentation ground truth and the corresponding expert

TABLE 4.1: Data Generation Setting

Maps	All CARLA Towns
Route sets*	Long (1000-2000m), short (100-500m), and tiny (one turn or one go-straight)
Weather presets	Clear noon, clear sunset, cloudy noon, cloudy sunset, wet noon, wet sunset, mid rainy noon, mid rainy sunset, wet cloudy noon, wet cloudy sunset, hard rain noon, hard rain sunset, soft rain noon, soft rain sunset
Non-player characters	Other vehicles and pedestrians
Object classes	Unlabeled, building, fence, other, pedestrian, pole, road lane, road, sidewalk, vegetation, other vehicles, wall, traffic sign, sky, ground, bridge, rail track, guard rail, traffic light, static object, dynamic object, water, terrain
CARLA version	0.9.10.1

*The number of routes in long, short, and tiny route sets is different in each town due to varying map complexity and characteristic.

trajectory, speed measurement, and navigational controls. The trajectory is defined by a set of 2D waypoints transformed in the local vehicle coordinate, while the navigational control is the record of the level of steering, throttle, and brake at the time. For comparison purposes, we also gather LiDAR point clouds which are needed by other models. Following the configuration used by Prakash et al. [80], we give the expert a set of predefined routes to follow in driving the ego vehicle. Each route is defined with a sequence of GNSS coordinates provided by the global planner and high-level navigational command (e.g., turn left, turn right, follow the lane, etc.). There are three kinds of route sets namely long, short, and tiny. In the long routes set, the expert must drive for 1000-2000 meters comprising around 10 intersections for each. In the short routes set, the expert must drive for 100-500 meters comprising three intersections for each. In the tiny routes set, the expert must complete one turn or one go-straight in an intersection or turn. To be noted, the number of routes for each kind of route set is different in each CARLA town depending on the map topography, road length, and other characteristics. Town01 to Town06 have all kinds of route sets, while Town07 and Town10 only have short and tiny route sets. We create two datasets, one for clear noon-only evaluation and one for all-weather evaluation (see Subsection 4.4.1 for more details).

Each generated dataset is expressed as $\mathbb{D} = \{(\mathbb{X}^i, \mathbb{Y}^i)\}_{i=1}^{J}$ where J is the size of the dataset. $\mathbb{X}$ is considered as a set of inputs composed of RGB image, depth map, LiDAR point clouds, speed measurement, GNSS locations, and high-level navigational command at a time. To be noted, DeepIPC does not take LiDAR point clouds and high-level navigational command to drive the ego vehicle. Meanwhile, $\mathbb{Y}$ is considered as a set of outputs composed of semantic segmentation ground truth, waypoints, and the record of navigational controls at the time together with the state of traffic light and stop sign appearance for additional supervision. Originally, RGB image and depth map are retrieved at a resolution of 300×400 then cropped to 256×256 for some reasons described in Subsection 4.3.1. Thus, both RGB image

and depth map are represented as $\mathbb{R} \in \{0, ..., 255\}^{3\times256\times256}$ which is the set of 8-bit values in a form of RGB channel (C) $\times$ height (H) $\times$ width (W). Then, the true depth value for each pixel i in the depth map can be decoded with (4.5).

$$\mathbb{R}_i^{dec} = \frac{R_i + 256G_i + 256^2B_i}{256^3 - 1} \times 1000, \tag{4.5}$$

where $\mathbb{R}_i^{dec}$ is the decoded true depth of pixel i, (R_i, G_i, B_i) are stored 8-bit value of pixel i, 256 is the highest decimal value of 8-bit, and 1000 is the actual depth range of RGBD camera in meters. Meanwhile, LiDAR point clouds are converted into a 2-bin histogram over a 2D BEV image $\mathbb{R}^{2\times256\times256}$ representing the point above and on/below the ground plane [80] [81] [147]. Then, segmentation ground truth is represented as $\mathbb{R} \in \{0, 1\}^{23\times256\times256}$ where 23 is the number of classes mentioned in Table 4.1 with 0 if the pixel does not belong to any class and 1 if the pixel belongs to a class. Then, the waypoints are represented in BEV space with $\{\omega\rho_i = (x_i, y_i)\}_{i=1}^{3}$. Keep in mind that the center (0,0) of the BEV space (local vehicle coordinate) is on the ego vehicle itself positioned at the bottom center. The model estimates the navigational controls in a normalized range of 0 to 1, then they will be denormalized to their original value with steering $\in \{-1, ..., 1\}$, throttle $\in \{0, ..., 0.75\}$, and brake $\in \{0, 1\}$. For the traffic light state and stop sign prediction, we set 1 if a red light/stop sign appeared, otherwise, they are 0. Meanwhile, speed measurement (in m/s) and GNSS locations are sparse, and high-level navigational commands are one-hot encoded.

4.3.3 Training Configuration

To learn multiple tasks simultaneously, several loss functions need to be defined first. For the semantic segmentation loss function ($\mathcal{L}_{SEG}$), we use a combination of binary cross-entropy and dice loss that can be calculated with (4.6). With this formulation, we can obtain the advantage of distribution-based and region-based losses at the same time [54] [153]. Giving extra loss criteria to the semantic segmentation task is necessary as the rest of the network architecture depends on it.

$$\mathcal{L}_{SEG} = \left(\frac{1}{N}\sum_{i=1}^{N} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right) + \left(1 - \frac{2|\hat{y} \cap y|}{|\hat{y}| + |y|}\right), \tag{4.6}$$

where N is the number of pixel elements at the output layer of the semantic segmentation decoder. Then, y_i and $\hat{y}_i$ are the value of i^{th} element of the ground truth y and prediction $\hat{y}$ respectively. Meanwhile, we use a simple L1 loss as formulated in (4.7) for the other tasks: traffic light state loss ($\mathcal{L}_{TL}$), stop sign loss ($\mathcal{L}_{SS}$), steering loss ($\mathcal{L}_{ST}$), throttle loss ($\mathcal{L}_{TH}$), brake loss ($\mathcal{L}_{BR}$), and waypoints loss ($\mathcal{L}_{WP}$).

$$\mathcal{L}_{\{TL,SS,ST,TH,BR,WP\}} = |\hat{y} - y| \tag{4.7}$$

To be noted, only $\mathcal{L}_{WP}$ needs to be averaged as there are three predicted waypoints. As explained in Subsection 4.3.1, DeepIPC outputs Δx and Δy instead the exact x,y-coordinate location. Thus, the waypoints need to be calculated first with (4.4) before computing the loss. Meanwhile, the prediction of navigational controls (steering, throttle, brake) needs to be denormalized first as described in Subsection 4.3.2. Finally, the total loss that covers all penalization can be calculated with (4.8).

$$\mathcal{L}_{TOTAL} = \alpha_1\mathcal{L}_{SEG} + \alpha_2\mathcal{L}_{TL} + \alpha_3\mathcal{L}_{SS} + \alpha_4\mathcal{L}_{ST} + \alpha_5\mathcal{L}_{TH} + \alpha_6\mathcal{L}_{BR} + \alpha_7\mathcal{L}_{WP}, \tag{4.8}$$

where $\alpha_{1,..,7}$ is the loss weight for each task. We use an adaptive loss weighting algorithm called modified gradient normalization (MGN) to tune the loss weights adaptively for each training epoch [54]. For a multi-task model, balancing task learning by modifying the gradient signal is necessary to prevent imbalance problems where the model tends to focus on a certain task only. We use Adam optimizer with decoupled weight decay of 0.001 to train DeepIPC until convergence [155] [127]. The initial learning rate is set to 0.0001 and reduced gradually by half if there is no drop on validation $\mathcal{L}_{TOTAL}$ in 3 epochs in a row. Additionally, the training will be stopped if there is no improvement in 15 epochs in a row to prevent unnecessary computational costs. We implement DeepIPC with PyTorch framework [122] and train it on NVIDIA GeForce RTX 3090 with a batch size of 20. The training log and the loss weighting behavior are explained in Appendix A.1.

4.4 Experiment Setup

In this section, we define the task and explain several scenarios for evaluations. Then, we explain some metrics used to measure model performance including other models and their variants for conducting ablation and comparative studies.

4.4.1 Task and Scenario

We consider the point-to-point navigation task where an autopilot model is obligated to drive an ego vehicle following a set of predefined routes and traffic regulations. Following standard CARLA protocol, the routes are defined in the form of sparse GNSS locations given by the global planner. The model drivability is evaluated in a variety of areas with different characteristics (e.g., urban, rural, highway, etc.) and various kinds of weather conditions. The main goal is to complete the routes while safely reacting to any events whether in normal or adversarial situations. For example, the model must avoid a collision with a pedestrian that suddenly crosses the street, or with another vehicle when there is a double green light error at an intersection. To achieve convincing evidence, we consider several scenarios for the experiments as follows.

- 1W-N: Clear noon-only with normal situations. In this scenario, we train the model on all available maps and route sets excluding Town05 short and tiny sets which are used for validation, and leave the Town05 long set (10 long routes) for evaluation purposes. Town05 is chosen as it is large and has complex characteristics. During the evaluation, all Non-Player Characters (NPC) behave normally following the traffic rules. We run the experiment three times and calculate the average performance and its standard deviation. The model is expected to be able to drive the ego vehicle properly by not violating traffic regulations or any other kind of infraction.
- 1W-A: Clear noon-only with adversarial situations. This scenario is similar to 1W-N, however, the NPC is behaving abnormally which can cause collisions (e.g., the pedestrian or bicyclist is crossing the street suddenly). Intentionally, we also make the traffic light manager create a state where double green lights appear at an intersection. Thus, the ego vehicle may collide with another vehicle coming from a different path. The purpose of this condition is to mimic the event of an ambulance or firefighter truck skipping the traffic light due to emergency situations. Besides driving the ego vehicle properly, the model is expected to be able to safely react and avoid collisions.

- AW-N: All weathers with normal situations. This scenario is similar to 1W-N, however, the model is trained and validated with the dataset in which the weather is changed dynamically. Then, the model is evaluated on Town05 long set one time for each weather preset mentioned in Table 4.1. Thus, we calculate the average and standard deviation of model performance over fourteen times running as there are fourteen weather presets. The model is expected to be able to adapt to various conditions.

- AW-A: All weathers with adversarial situations. This scenario is similar to AW-N but with adversarial situations as described in 1W-A. This is the hardest scenario in our experiments where model performance is justified based on the robustness against various weather conditions and the ability to respond to adversarial situations.

4.4.2 Performance Evaluation

In the evaluation process, there are several metrics used to justify model performance in some aspects of driving. Following the CARLA leaderboard evaluation setting (https://leaderboard.carla.org), we use the driving score (DS) as the main metric where the higher the driving score means the better the model. The driving score can be computed with (4.9).

$$DS = \frac{1}{N_r} \sum_{i=1}^{N_r} RC_i IP_i \tag{4.9}$$

The DS for the i^{th} route (DS_i) is a simple multiplication between the percentage of route completion of route i (RC_i) and the infraction penalty of route i (IP_i). Then, the final driving score can be calculated by averaging over N_r, the number of routes in the route set. RC_i can be simply obtained by dividing the completed distance of route i by the total length of route i. However, if the ego vehicle drives offroad (e.g., drives on the sidewalk), then the path where the ego vehicle drives offroad is not counted and yields a reduced RC_i. Meanwhile, IP_i can be computed with (4.10).

$$IP_i = \prod_{j}^{M} (p_i^j)^{\#\text{infractions}_j}, \tag{4.10}$$

where M is the set of infraction types considered for the evaluation process. The IP_i for each model starts with an ideal base score of 1.0 and is reduced if an infraction is committed. Ordered by its severity, we consider the following types of infraction M and penalty values p^j as described in the CARLA leaderboard website.

- Collision with pedestrians: 0.50
- Collision with other vehicles: 0.60
- Collision with others (static elements): 0.65
- Red light violation: 0.70
- Stop sign violation: 0.80

The final RC and IP scores can be obtained by averaging over N_r similar to the final DS calculation. To save computation costs, the evaluation process on route i will be stopped if the ego vehicle deviates more than 30 meters from the assigned route

TABLE 4.2: Model Specification

Model	Total Parameters↓	GPU Usage ↓	Model Size ↓	Input/Sensor
CILRS	12693331	2143 MB	50.871 MB	RGB camera, speedometer, high-level command
AIM	21470722	2217 MB	86.033 MB	RGB camera, GNSS, speedometer
LF	32644098	2303 MB	130.808 MB	RGB camera, LiDAR, GNSS, speedometer
GF	40919554	2367 MB	163.944 MB	RGB camera, LiDAR, GNSS, speedometer
TF	66293634	2553 MB	265.617 MB	RGB camera, LiDAR, GNSS, speedometer
DeepIPC	20985934	2197 MB	84.984 MB	RGBD camera, GNSS, speedometer

GPU memory usage is measured by NVIDIA GeForce RTX 3090 driver while the model size is measured based on Ubuntu 20 system. We assume that models with fewer trainable parameters (number of neurons) and less GPU utilization will inference faster. We cannot compute the inference speed fairly since we run multiple parallel experiments at the same time so the GPU computation performance becomes very fluctuating. CILRS: Conditional Imitation Learning-based model [144] (R: using ResNet [145], S: with Speed input), AIM: Auto-regressive IMage-based model [80], LF: Late Fusion-based model [80], GF: Geometric Fusion-based model [156] [157] [158], TF: TransFuser model [80].

or does not take any actions for 180 seconds. Thus, it will affect the performance calculation, yielding a low RC_i. Then, the evaluation process continues to the next route for further performance calculation.

As mentioned in Subsection 4.4.1, we evaluate the model in several scenarios to understand multiple aspects of driving. Furthermore, for a comparative study, we pick CILRS [144] (Conditional Imitation Learning using ResNet [145] and Speed input) as the representative of a model that needs high-level navigational commands to drive the ego vehicle. Then, we replicate several models developed by Prakash et al. [80] as the representative of a model that does not need high-level navigational commands to drive the ego vehicle. To be more detailed, we replicate four models namely AIM (Auto-regressive IMage), LF (Late Fusion), GF (Geometric Fusion), and TF (TransFuser). These models have the same module that is responsible for determining navigational controls, however, their perception modules are completely different from one another. AIM only uses an RGB camera as the main source of information for its perception module. Meanwhile, the other models combine a front RGB camera and LiDAR sensor but with different fusion strategies. TF uses transformers to learn the relationship between two unique features. GF uses geometric transformation inspired by Liang et al. [156] [157] [158] for fusing the extracted features. Meanwhile, LF only uses element summation to fuse both features and let the next layer learn its correlation. To ensure that the comparison is conducted fairly, we use the same camera settings for all models and consider an ROI of 256×256 described in Prakash et al.'s works [80]. The model specification details can be seen in Table 4.2 in which DeepIPC has the second smallest number of parameters. Furthermore, we conduct an ablation study by modifying the architecture of DeepIPC and changing the control policy to understand their contribution.

To reflect the intuitive performance on each task independently, we also conduct an inference test on the expert driving data and do a comparative study with some recent task-specific models. We compare DeepIPC with CILRS for the navigational controls estimation task. Then, we compare DeepIPC against AIM, LF, GF, and TF for the waypoints prediction task. Meanwhile, for semantic segmentation comparison, we train and evaluate the DeepLabV3+ segmentation model [159] with ResNet101 backbone [145] pre-trained on ImageNet [152]. Finally, for the performance comparison of traffic light state and stop sign prediction, we train and evaluate a classifier model based on Efficient Net B7 [151] pre-trained on ImageNet [152].

TABLE 4.3: Driving Performance Score for Comparative Study

Scenario	Model	DS↑	RC↑	IP↑	Collision↓			Violation↓		Offroad↓
					Pedestrian	Vehicle	Others	Red	Stop	
Normal Clear Noon (1W-N)†	CILRS	8.291 ± 0.571	11.149	0.819	0.000	1.060	0.019	0.207	0.000	1.042
	AIM	41.191 ± 1.047	92.794	0.456	0.000	0.043	0.004	0.108	0.088	0.014
	LF	42.770 ± 2.334	73.836	0.457	0.000	0.032	0.004	0.043	0.072	0.055
	GF	37.320 ± 7.150	82.592	0.480	0.000	0.056	0.000	0.089	0.123	0.052
	TF	35.273 ± 1.115	57.680	0.710	0.000	0.020	0.000	0.097	0.037	0.085
	DeepIPC	**48.428 ± 1.467**	84.270	0.625	0.000	0.038	0.000	0.048	0.075	0.011
	Expert	73.553 ± 5.086	100.000	0.735	0.000	0.051	0.000	0.019	0.000	0.000
Adversarial Clear Noon (1W-A)†	CILRS	8.261 ± 0.924	13.237	0.657	0.119	0.512	0.095	0.258	0.000	1.012
	AIM	25.541 ± 2.257	79.141	0.412	0.018	0.113	0.005	0.175	0.041	0.081
	LF	35.310 ± 2.510	58.744	0.658	0.012	0.049	0.000	0.102	0.018	0.031
	GF	32.423 ± 1.042	65.291	0.602	0.012	0.148	0.000	0.049	0.023	0.094
	TF	24.740 ± 0.948	37.921	0.788	0.016	0.074	0.008	0.121	0.008	0.101
	DeepIPC	**35.982 ± 2.105**	76.189	0.476	0.034	0.062	0.000	0.214	0.020	0.014
	Expert	41.579 ± 1.576	68.225	0.696	0.000	0.086	0.000	0.081	0.000	0.000
Normal All Weathers (AW-N)*	CILRS	7.376 ± 0.996	9.569	0.869	0.000	0.849	0.020	0.071	0.000	0.964
	AIM	39.613 ± 3.644	84.467	0.542	0.000	0.022	0.018	0.090	0.076	0.030
	LF	36.890 ± 7.067	48.747	0.805	0.000	0.063	0.001	0.023	0.050	0.041
	GF	20.446 ± 5.281	29.616	0.831	0.000	0.051	0.000	0.079	0.022	0.063
	TF	16.672 ± 4.175	26.425	0.843	0.000	0.011	0.014	0.082	0.010	0.427
	DeepIPC	**47.133 ± 5.276**	77.427	0.653	0.000	0.069	0.021	0.031	0.056	0.080
	Expert	75.815 ± 5.045	100.000	0.758	0.000	0.058	0.000	0.012	0.000	0.000
Adversarial All Weathers (AW-A)*	CILRS	5.234 ± 0.869	8.663	0.778	0.119	0.039	0.148	0.357	0.000	1.049
	AIM	30.207 ± 5.857	76.399	0.471	0.013	0.066	0.028	0.147	0.041	0.047
	LF	22.592 ± 7.039	32.525	0.808	0.371	0.088	0.010	0.050	0.021	0.069
	GF	14.799 ± 3.860	20.113	0.865	0.007	0.120	0.007	0.036	0.008	0.069
	TF	11.167 ± 2.434	20.965	0.808	0.003	0.054	0.033	0.124	0.011	0.273
	DeepIPC	**31.055 ± 2.700**	64.132	0.531	0.031	0.128	0.049	0.106	0.022	0.134
	Expert	39.475 ± 6.792	71.474	0.643	0.002	0.108	0.001	0.078	0.000	0.000

The best performance is defined by the highest driving score (DS) in each scenario. CILRS: Conditional Imitation Learning-based model [144] (R: using ResNet [145], S: with Speed input), AIM: Auto-regressive IMage-based model [80], LF: Late Fusion-based model [80], GF: Geometric Fusion-based model [156] [157] [158], TF: TransFuser model [80].
†The result is averaged over three experiment runs.
*The result is averaged over fourteen times of experiment run with one time running for each weather preset mentioned in Table 4.1.

4.5 Result and Discussion

Be noted, the main evaluation metric used to determine the best model is the driving score (DS) which is the multiplication of route completion (RC) and infraction penalty (IP). Keep in mind that having a higher RC or IP does not mean that the model is better. The model may have a higher RC by disobeying the traffic rules so it can keep going to achieve further route distance but drive terribly awful. The model may also have small infraction counts (resulting in higher IP) due to the low percentage of the completed route where any collisions or traffic violations can happen if it travels further. Therefore, the most appropriate metric is DS as it combines both aspects of driving in RC and IP: drive as far as possible with small infractions as little as possible. As mentioned in Subsection 4.4.1, we calculate the average and the standard deviation of each model performance over three times of experiments run in clear noon-only evaluation (1W-N and 1W-A) and over fourteen times experiments run in all weathers evaluation (AW-N and AW-A). The evaluation score can be seen in Table 4.3. In addition, we add several driving footage on various weather conditions as shown in Fig. 4.4.

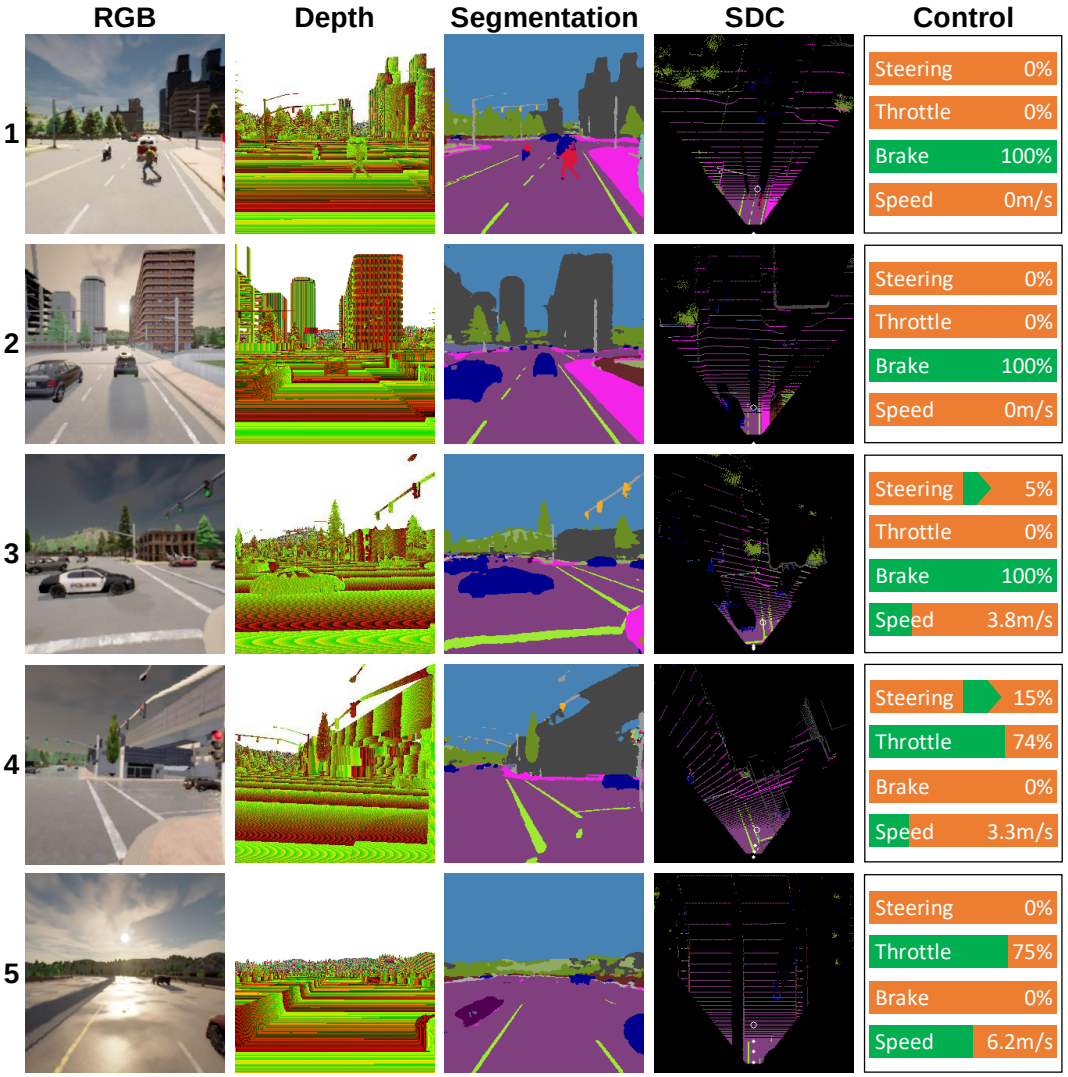


FIGURE 4.4: Driving footage.

1. Clear noon: DeepIPC manages to stop the ego vehicle as a pedestrian crosses suddenly. Therefore, the pedestrian does not get hit by the vehicle.
2. Cloudy sunset: Even though there are some vacant spaces in front of the ego vehicle, DeepIPC tends to stop behind any vehicles shown on the front camera at the intersection as a result of behavior cloning from the expert driver.
3. Mid rainy noon: DeepIPC reacts properly and avoids collision by doing instant braking when another vehicle driving ahead due to a double green light error at the intersection (simulating an ambulance or a firefighter truck rushing for emergency situations).
4. Hard rain sunset: After the traffic light turns green, DeepIPC drives to the right getting close to the given route (white hollow circle inside semantic depth cloud (SDC) map). However, it drives offroad as the sidewalk is wrongly classified as a road.
5. Wet sunset: DeepIPC drives fast on a highway. Although it cannot segment the road barrier properly, DeepIPC manages to stay on the lane thanks to the well-segmented road lane and the information on vacant regions in the SDC map.

4.5.1 Drivability in Normal and Adversarial Situations

Normally, all road users, including pedestrians and other drivers, must obey traffic regulations so that any kind of accident can be prevented. However, this condition is nearly impossible in the real world, especially in a crowded urban area where

plenty of vehicles and pedestrians are moving around. No one can guarantee that the pedestrian will always walk only on the sidewalk or not cross the street suddenly. Moreover, there is also a possibility that the traffic light can err (e.g., double green light) due to a certain cause. Therefore, we evaluate all models in both normal situations (where everything is working as it should be) and adversarial situations (where unexpected events can occur at any time). Specifically, in adversarial situations, we make abnormal events by spawning some pedestrians to cross the street suddenly and making the traffic light error when the ego vehicle is approaching. These events are used to mess up the traffic on purpose to simulate an ambulance or a firefighter truck rushing for emergency situations. Besides driving the vehicle properly, the model is expected to be able to react safely and avoid collisions with pedestrians or other vehicles in these situations.

Based on Table 4.3, DeepIPC has the best drivability by achieving the highest driving scores (DS) in all scenarios. Moreover, DeepIPC can be said to be more stable compared to the second-best model, LF, as it achieves a smaller standard deviation of DS in the clear noon-only evaluation (1W-N and 1W-A). Meanwhile, in all weathers evaluation, DeepIPC stability is comparable to AIM as the standard deviation is larger in AW-N but smaller in AW-A. This is caused by the variety of weather conditions that challenge the capability of the perception module in extracting stable features for the controller module. Based on the comparison of DS in normal situations and adversarial situations, all models suffer from such unexpected events. The driving scores achieved by each model in 1W-A and AW-A scenarios are lower than in the 1W-N and AW-N scenarios. The performance drop is as expected due to abnormalities that occurred during driving. However, even with these adversarial situations, DeepIPC still manages to achieve the best performance meaning that it can react properly to prevent such kinds of infractions.

4.5.2 Adaptability to Various Weather Conditions

To evaluate all models further, we also change the weather condition in fourteen weather presets provided in the CARLA simulator. The purpose of this evaluation is to check the adaptability of the model when deployed in various conditions. In the real world, it is obvious that we cannot expect that the weather will always be sunny where everything is clearly visible. Therefore, conducting adaptability tests in various weather conditions is necessary. Unlike in the clear noon-only test where each model runs three times, we run each model only once for each weather. Then, the average and standard deviation are calculated over fourteen sets of scores as described in Subsection 4.4.1.

As shown in Table 4.3, all models get their driving score dropped in AW-N and AW-A scenarios compared to their achievement in 1W-N and 1W-A scenarios. Besides that, most of the models have a larger standard deviation on their scores. However, DeepIPC still manages to achieve the best performance. The phenomena of performance drop and less stability mean that performing scene understanding to perceive the surrounding is much harder in various weather conditions. The only outlier of this result is the fact that the AIM performance on the adversarial situation in all weathers evaluation is better than in clear noon-only evaluation where its DS is getting improved from 25.541 to 30.207. An answer to this phenomenon lies in the network architecture of AIM. As mentioned in Table 4.2, AIM uses a front RGB image as the only input for its perception module. Therefore, in various weather conditions, its capability in extracting informative and stable features can be drastically improved as the rest of the architecture only relies on it.

4.5.3 Models Behavior

The CILRS model which takes high-level navigational commands is completely far behind the other models. The problem with the navigational command-based model is the imbalance distribution of discrete navigational commands. It is obvious that the command *follow lane* or *go straight* are more than *turn left* or *turn right*. As a result, the model tends to fail to turn properly at the intersection. Judging from its low percentage of RC, the model cannot drive any further due to a collision with a pedestrian, another vehicle, or other objects. The model may have failed at the first or second intersection in the given route.

The AIM model has the highest RC, however, it cannot achieve the highest DS as it makes many infractions resulting in the lowest IP in all scenarios. The problem with AIM is it only uses the front RGB image as the only input for its perception module. The reason AIM can travel further compared to the other models in all scenarios is due to its less awareness caused by limited information about the surrounding. AIM tends to just keep going and makes a lot of infractions since the only thing it knows about is the information in front of the ego vehicle. This kind of driving behavior is resulting in the highest RC but with the smallest IP (many infractions) in each scenario.

The RGB-LiDAR fusion-based models (LF, GF, and TF) are too careful which results in smaller RC and higher IP (fewer infractions). During the evaluation process, these models tend to wait for another vehicle to make the first move in an intersection so that they can follow those vehicles. Unlike AIM, these models focus on capturing the surrounding vehicle situation rather than recognizing the traffic light state to decide whether to drive the vehicle or not. Therefore, if there are no other vehicles around, then these models are not going to drive the vehicle any further. In this case, these models may have a better understanding of the surrounding area, however, the extracted features provided by the LiDAR encoder are biasing too much and affecting its ability to recognize the state of the traffic light which cannot be captured by LiDAR.

In DeepIPC, the information provided by LiDAR is replaced by the semantic depth cloud (SDC) mapping function. In addition, we use a specific module to recognize the traffic light state and the appearance of stop signs that may disappear during GRU looping. Furthermore, we also use two agents as the decision makers to create more varying driving options. The first agent is the MLP network that decodes the final hidden state given by the GRU. Meanwhile, the second agent is two PID controllers that translate predicted waypoints into navigational controls. As a result, DeepIPC can maintain the trade-off between RC and IP which results in the highest driving score as shown in Table 4.3. Moreover, DeepIPC is more efficient compared to LF and AIM as it has fewer parameters and smaller GPU memory usage as described in Table 4.2.

4.5.4 The Importance of SDC and Multi-agent

We conduct an ablation study on the clear noon-only evaluation by removing the semantic depth cloud (SDC) mapping function and changing the control policy to understand how the SDC map and multi-agent can improve DeepIPC performance.

To understand how important SDC mapping is, we train and evaluate a variant of DeepIPC that does not have this function with the same experiment settings for a fair comparison. As shown in Table 4.4, the *No SDC* variant achieves a lower driving score with a higher standard deviation. This result is as expected since the

TABLE 4.4: Driving Performance Score for Ablation Study

Scenario	Variant	DS↑	RC↑	IP↑	Collision↓			Violation↓		Offroad↓
					Pedestrian	Vehicle	Others	Red	Stop	
Normal (1W-N)	*No SDC*	39.579 ± 10.251	63.276	0.715	0.000	0.028	0.022	0.062	0.068	0.029
	Proposed	**48.428 ± 1.467**	84.270	0.625	0.000	0.038	0.000	0.048	0.075	0.011
	MLP	43.210 ± 3.689	80.794	0.589	0.000	0.047	0.000	0.078	0.054	0.020
	PID	46.351 ± 7.879	65.673	0.761	0.000	0.022	0.000	0.081	0.044	0.034
	Both	42.927 ± 9.392	67.373	0.705	0.000	0.026	0.000	0.072	0.037	0.010
	Expert	71.440 ± 5.855	100.000	0.714	0.000	0.061	0.000	0.011	0.000	0.000
Adversarial (1W-A)	*No SDC*	29.624 ± 7.006	56.430	0.606	0.030	0.043	0.006	0.256	0.026	0.013
	Proposed	**35.982 ± 2.105**	76.189	0.476	0.034	0.062	0.000	0.214	0.020	0.014
	MLP	26.936 ± 3.773	59.178	0.511	0.090	0.109	0.000	0.207	0.015	0.004
	PID	32.282 ± 1.899	58.748	0.635	0.004	0.112	0.010	0.181	0.026	0.017
	Both	32.396 ± 4.531	58.367	0.622	0.004	0.108	0.000	0.227	0.012	0.028
	Expert	41.579 ± 1.576	68.225	0.696	0.000	0.086	0.000	0.081	0.000	0.000

The result is averaged over three times in the experiment. The best performance is defined by the highest driving score (DS) in each scenario. Proposed: the configuration as described in Section 4.3, No SDC: there is no SDC map provided so that the controller module only receives RGB features, MLP: DeepIPC only uses MLP agent on its controller module, PID: DeepIPC only uses PID agent on its controller module, Both: DeepIPC drives the vehicle if and only if both MLP and PID throttles are above the threshold of 0.2.

model loses its capability in performing robust scene understanding. Unlike raw point clouds that only provide the height information, the SDC map holds semantic information for each class on each layer. With this representation, the model can understand much useful information easily. Moreover, by doing concatenation rather than element summation on RGB and SDC features, the fusion block can learn the relation by itself to prevent information loss.

By default (*Proposed* variant), DeepIPC uses both MLP and PID agents in driving the ego vehicle as described in Subsection 4.3.1. DeepIPC will stop the vehicle if and only if both throttle values are below the threshold of 0.2. If only the MLP agent in which the throttle value is higher than 0.2, then the vehicle is fully controlled by the MLP agent, and so does for the PID agent. By using the same network architecture, we create three more DeepIPC variants namely *MLP*, *PID*, and *Both*. In *MLP* and *PID* variants, the vehicle is controlled by one agent only, MLP or PID. Meanwhile, the *Both* variant uses both agents as similar to the *Proposed* variant. However, this variant will drive the vehicle if and only if both throttle values are higher than 0.2. This means that if one of the throttle values is below 0.2, the vehicle will stop.

Based on Table 4.4, the *Proposed* variant still achieves the best performance compared to the other variants. With more driving options provided by two agents that represent different aspects of driving, the model can drive the vehicle farther and make a better trade-off between route completion (RC) percentage and infraction penalty (IP). In normal situations, the *Both* variant achieves the lowest score as it is too careful by considering both decisions made by MLP and PID agents. However, this carefulness can make the *Both* variant surpasses the *MLP* and *PID* variants in adversarial situations since being more careful is preferable in handling unexpected abnormal events such as a pedestrian crosses suddenly. Meanwhile, the *MLP* variant achieves the second-best in RC which means that the MLP agent is less aware compared to the PID agent and results in a lower IP (many infractions). On the other hand, the PID agent can be said to be better than the MLP agent as it achieves higher DS. This means that the *PID* variant is better than the *MLP* policy in managing the trade-off between RC and IP.

4.5.5 Task-wise Evaluation

The purpose of this evaluation is to analyze model performance in handling multiple perception and control tasks simultaneously. Hence, we conduct a comparative study with some task-specific models to reflect the intuitive performance on each task independently. In this experiment, we configure all models to perform inference on the expert's driving data on the evaluation routes (Town05 long route set) in four different scenarios (1W-N, 1W-A, AW-N, AW-A) which are completely unseen in the training and validation datasets used for the imitation learning process. We consider the expert's trajectory and navigational controls (steering, throttle, brake) record along with the segmentation map, traffic light state, and stop sign appearance provided by CARLA as the ground truth for evaluation.

For metric scoring, we use intersection over union (4.11) to determine the performance on the semantic segmentation task.

$$IoU_{SEG} = \frac{|\hat{y} \cap y|}{|\hat{y} \cup y|}, \tag{4.11}$$

where $\hat{y}$ and y are the prediction and ground truth respectively. Meanwhile, for the traffic light state and stop sign predictions, we use a simple accuracy scoring (4.12) as the metric function.

$$Acc._{\{TL,SS\}} = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.12}$$

where TP, TN, FP, and FN are the true positive, true negative, false positive, and false negative predictions made by the model. Then, to justify model performance in predicting waypoints and navigational controls, we use mean absolute error (MAE) as the metric function which is the same function used for their loss calculation as mentioned in Subsection 4.3.3 (known as L1 loss). Similar to Table 4.3, the results shown in Table 4.5, 4.6, 4.7, and 4.8 are averaged over three inference results for clear noon-only scenarios (1W-N and 1W-A) and averaged over fourteen inference results for all weathers scenarios (AW-N and AW-A).

Semantic Segmentation

Based on Table 4.5, it can be said that performing semantic segmentation in varying weather conditions is harder than in a single weather condition. In the varying weather scenarios (AW-N and AW-A), both DeepLabV3+ and DeepIPC have lower IoU scores with larger standard deviations than in the clear noon-only scenarios (1W-N and 1W-A). This result is in line with the slight degradation of the driving score shown in Table 4.3 and discussed in Subsection 4.5.2 meaning that performing prediction in various kinds of weather is more challenging. In all scenarios, it is expected that DeepLabV3+ has higher IoU scores with a smaller standard deviation meaning that it has better performance and stability than DeepIPC. However, with just a small gap difference in IoU score, DeepIPC is still preferable considering the large number of neurons used in DeepLabV3+ architecture that can cause a huge computational load.

TL State and Stop Sign Prediction

In the task of traffic light (TL) state and stop sign prediction, DeepIPC has comparable accuracy scores in all scenarios as shown in Table 4.6. Unlike in the semantic segmentation task, the model only deals with one specific object which is easier to

TABLE 4.5: Semantic Segmentation Score

Model	$IoU_{SEG} \uparrow$			
	1W-N	1W-A	AW-N	AW-A
DLV3+	**0.888 ± 0.001**	**0.885 ± 0.002**	**0.884 ± 0.003**	**0.883 ± 0.003**
DeepIPC	0.883 ± 0.001	0.880 ± 0.003	0.878 ± 0.004	0.879 ± 0.004

IoU_{SEG}: intersection over union score of semantic segmentation. DLV3+: DeepLabV3+ segmentation model [159] with ResNet101 backbone [145].

TABLE 4.6: TL State and Stop Sign Prediction Score

Scenario	Model	$Acc._{TL} \uparrow$	$Acc._{SS} \uparrow$
1W-N	ENB7	0.967 ± 0.008	0.995 ± <0.001
	DeepIPC	**0.986 ± 0.003**	**0.996 ± <0.001**
1W-A	ENB7	0.975 ± 0.004	**0.996 ± <0.001**
	DeepIPC	**0.982 ± 0.006**	**0.996 ± <0.001**
AW-N	ENB7	0.980 ± 0.005	**0.996 ± <0.001**
	DeepIPC	**0.989 ± 0.004**	**0.996 ± <0.001**
AW-A	**ENB7**	**0.979 ± 0.003**	**0.996 ± 0.001**
	DeepIPC	0.979 ± 0.004	**0.996 ± 0.001**

$Acc._{TL}$: accuracy of traffic light (TL) state prediction, $Acc._{SS}$: accuracy of stop sign prediction. ENB7: A classifier model based on Efficient Net B7 variant [151].

handle. Moreover, although DeepIPC uses a smaller Efficient Net version (Efficient Net B3) as its RGB encoder, it manages to be better than a classifier model based on Efficient Net B7 (the best and the biggest amongst Efficient Net model variants) almost in all scenarios. DeepIPC only loses in the AW-A scenario where it has a slightly larger standard deviation in the traffic light state prediction score. Thanks to the end-to-end and multi-task learning strategy, DeepIPC can receive more supervision since its prediction is encoded further and used to bias a certain latent space for other task predictions.

Waypoints Prediction

As mentioned in Section 4.5, we compare DeepIPC with AIM, LF, GF, and TF in the waypoints prediction task. These models predict four waypoints, while DeepIPC only predicts three waypoints. Yet, it is still considered a fair comparison since we use MAE which averages the error on all predictions. Based on Table 4.7, DeepIPC has the lowest MAE with the smallest standard deviation compared to the other models in all scenarios. Thanks to the SDC map which lies on BEV space the same as where the coordinates of the waypoint are located. This means that the SDC map is proven to play an important role in giving the model a strong intuition in predicting waypoints as it provides the information of free and occupied regions clearly in BEV space. Besides that, performing waypoint prediction in adversarial driving is harder than in normal driving. In a comparison between the result in 1W-N with 1W-A and AW-N with AW-A, the MAE constantly becomes larger. Furthermore, if we compare the result in 1W-N with AW-N and 1W-A with AW-A, the MAE also constantly becomes larger which means that performing waypoints prediction in

TABLE 4.7: Waypoints Prediction Score

Model	$MAE_{WP} \downarrow$			
	1W-N	1W-A	AW-N	AW-A
AIM	0.233 ± 0.017	0.326 ± 0.042	0.221 ± 0.015	0.292 ± 0.028
LF	0.209 ± 0.011	0.307 ± 0.039	0.192 ± 0.014	0.294 ± 0.036
GF	0.231 ± 0.010	0.326 ± 0.038	0.196 ± 0.014	0.271 ± 0.025
TF	0.183 ± 0.009	0.286 ± 0.039	0.186 ± 0.009	0.249 ± 0.018
DeepIPC	$\mathbf{0.114 \pm 0.003}$	$\mathbf{0.166 \pm 0.023}$	$\mathbf{0.120 \pm 0.006}$	$\mathbf{0.172 \pm 0.015}$

MAE_{WP}: mean absolute error of waypoints prediction. AIM: Auto-regressive IMage-based model [80], LF: Late Fusion-based model [80], GF: Geometric Fusion-based model [156] [157] [158], TF: TransFuser model [80].

TABLE 4.8: Navigational Controls Estimation Score

Scenario	Model	$MAE_{ST} \downarrow$	$MAE_{TH} \downarrow$	$MAE_{BR} \downarrow$
1W-N	**CILRS**	$\mathbf{0.022 \pm 0.002}$	$\mathbf{0.052 \pm 0.002}$	0.053 ± 0.003
	DeepIPC	0.025 ± 0.001	0.054 ± 0.001	$\mathbf{0.044 \pm 0.002}$
1W-A	**CILRS**	$\mathbf{0.025 \pm 0.005}$	$\mathbf{0.097 \pm 0.006}$	$\mathbf{0.116 \pm 0.011}$
	DeepIPC	0.041 ± 0.008	0.117 ± 0.016	0.126 ± 0.018
AW-N	**CILRS**	$\mathbf{0.024 \pm 0.001}$	$\mathbf{0.054 \pm 0.008}$	0.057 ± 0.011
	DeepIPC	0.029 ± 0.002	0.069 ± 0.005	$\mathbf{0.044 \pm 0.005}$
AW-A	**CILRS**	$\mathbf{0.024 \pm 0.003}$	$\mathbf{0.088 \pm 0.006}$	$\mathbf{0.102 \pm 0.007}$
	DeepIPC	0.035 ± 0.006	0.120 ± 0.009	0.107 ± 0.008

MAE_{ST}: mean absolute error of steering prediction, MAE_{TH}: mean absolute error of throttle prediction, MAE_{BR}: mean absolute error of brake prediction. CILRS: Conditional Imitation Learning-based model [144] (R: using ResNet [145], S: with Speed input).

varying weather is also harder than in one single weather. This result is in line with the result shown in Table 4.3 and discussed in Subsection 4.5.1 and 4.5.2 where all models are suffered from adversarial situations and varying weather conditions, yielding a lower driving score.

Navigational Controls Estimation

Based on Table 4.8, especially in a comparison between the result in 1W-N with 1W-A and AW-N with AW-A, both CILRS and DeepIPC have larger MAE scores in the adversarial conditions. This means that both models have inferior performance due to various unexpected situations such as predicting the navigational controls (steering, throttle, brake) when a pedestrian crosses the street suddenly. This phenomenon also appears in the waypoints prediction performance evaluation meaning that adversarial driving is a big challenge for all models, especially on their controller module. Furthermore, DeepIPC loses to CILRS where it constantly has a larger MAE, especially on steering and throttle estimation. This result is as expected since CILRS is fed with one-hot encoded high-level navigational commands that can boost its confidence in estimating navigational controls level. However, this result is contradictive to the result shown in Table 4.3 where CILRS cannot perform very well in the driving task on the CARLA simulator. This is because it only performs inference on a set of

recorded driving data and has nothing to do with the environment state. Therefore, each prediction made by the model will not affect the future state of the environment which critically affects the performance. Moreover, as explained in Subsection 4.2.2, providing high-level commands is less reliable in real-world autonomous driving as there is no sensor that can give high-level commands other than the command from the driver itself. Therefore, although results in a larger MAE, providing GNSS locations along with global-to-local coordinate transformation is preferable since they still can give an explicit intuition of navigational commands to the model which is useful for navigational controls estimation.

4.6 Findings

In this Chapter, we present an end-to-end deep multi-task learning model called DeepIPC to handle both perception and control tasks simultaneously for an autonomous driving vehicle. We consider point-to-point navigation tasks where the model is obligated to drive the ego vehicle by following a sequence of routes defined by the global planner. CARLA simulator is used to simulate a driving environment with four different scenarios for evaluating the model and understand several aspects of driving. A comparative study with some recent models is conducted to justify the performance. Then, an ablation study is also conducted to understand the behavior and roles of some modules inside the architecture of DeepIPC. Furthermore, an extensive evaluation with task-wise performance scoring is conducted to analyze DeepIPC's capability in handling multiple tasks simultaneously. Based on the experiment result, we disclosed several findings as follows.

- All models suffer in adversarial situations and various weather conditions (except the AIM model in the AW-A scenario).
- DeepIPC achieves the highest driving score (DS) as it can react properly to abnormalities thanks to the SDC map that provides stable features to the perception module. In addition, DeepIPC can maintain the trade-off between route completion (RC) and infraction penalty (IP) as it understands different aspects of driving supported by two agents. Moreover, DeepIPC is more efficient as it has fewer trainable parameters and uses less GPU utilization compared to the runner-up of each scenario.
- AIM has the highest RC, however, it cannot achieve the highest DS as it makes many infractions due to its less awareness. Meanwhile, the fusion-based models are too careful to drive and result in low RC. The LiDAR features give too much bias so the model loses important information.
- Based on the ablation study, the SDC map and multi-agent are proven to play important roles in enhancing model drivability as they provide better perception and more control options.
- Based on the task-wise performance comparison, DeepIPC has better performances in waypoint prediction, traffic light state prediction, and stop sign prediction tasks. Although it loses in semantic segmentation and navigational controls estimation tasks, DeepIPC is still preferable considering its size and reliability.

Chapter 5

Vision-based End-to-end Autonomous Driving

Following the success of DeepIPC (Deeply Integrated Perception and Control) in driving a vehicle in a simulated environment as discussed in Chapter 4, we continue to use this model and perform some improvements to make it suitable for driving a robotic vehicle in real environments. In brief, DeepIPC is an end-to-end autonomous driving model that handles both perception and control tasks simultaneously. The model consists of two main parts which are the perception module and the controller module. The perception module takes an RGBD image to perform semantic segmentation and bird's eye view (BEV) semantic mapping along with providing their encoded features. Meanwhile, the controller module processes these features with the measurement of GNSS locations and angular speed to estimate waypoints that come with latent features. Then, two different agents are used to translate waypoints and latent features into a set of navigational controls to drive the vehicle. We modify several parts of DeepIPC to address some real-world implementation challenges. Unlike in an ideal simulated environment, we have to deal with several issues such as sensor inaccuracy and noise. We also need to think about the global coordinate which is completely different from a simulation. The modified model is evaluated by predicting driving records and performing automated driving under various conditions in real environments. We define a different evaluation metric where the best drivability is defined by the lowest driver intervention, not by the lowest number of collisions. This is necessary to avoid any damage to the robotic vehicle during evaluation. The experimental results show that DeepIPC achieves the best drivability and multi-task performance even with fewer parameters compared to the other models.

5.1 Real-world Imitation Learning

In order to achieve end-to-end autonomous driving, one approach is to proceed with behavior cloning or imitation learning strategies which can be done easily in a supervised learning manner [160] [161]. By using the end-to-end imitation learning strategy, we can create a single deep learning model to imitate the behavior of an expert driver in manipulating controls or effectors for handling complicated situations in the environment [162] [163]. This can be derived from publicly available datasets or simulated with a simulator to enrich the driving experiences [164] [165]. Therefore, the model will be able to perform human-like autonomous driving [166] [167].

Imitation learning has been widely used for real-world experiments. To be more specific in the field of mobile robotics and autonomous vehicles, recent work is proposed by Cai et. al. [168] where a vision-based model is employed for driving a

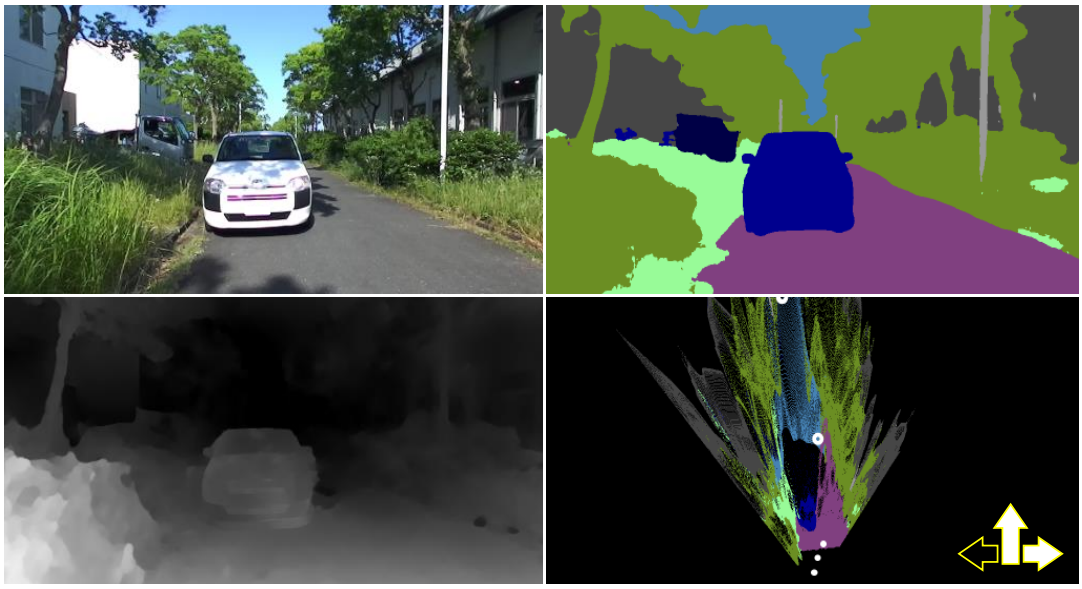


FIGURE 5.1: The inputs and outputs of the modified DeepIPC.

DeepIPC perceives the environment by performing semantic segmentation and BEV semantic mapping. At the same time, it also estimates waypoints and navigational controls to drive the vehicle. The detailed architecture of DeepIPC can be seen in Fig. 5.2.

toy-size autonomous race car in a fixed circuit. This work shows how the imitation learning technique can be used to train a simple model to learn the mapping function that transforms an RGB image into navigational controls. Not only simple models, but this technique is also applicable to multi-input multi-output models that process multiple data. A recent work by Chatty et. al. [169] demonstrates the use case of imitation learning for cognitive map building used for navigating a mobile robot. Then, Hoshino et. al. [170] also use the imitation-based end-to-end multi-task learning technique for motion planning and controlling a mobile robot in a challenging environment. Another work is proposed by Yan et. al. [171] where an end-to-end model is used to control a robotic shark. These complex models are supported with multiple sensors and are used to control several end-effectors. Following the success of these works in using end-to-end imitation learning for complex multi-input multi-output models, we also use this approach to train DeepIPC for driving a robotic vehicle in real environments. However, although this method seems promising, imitation learning sometimes causes an issue of generalization ability in unknown environments. To overcome this problem, we employ two control agents to manipulate the vehicle's end-effectors. As there are more decision-makers in its architecture, DeepIPC will be able to consider different aspects of drivability.

5.2 DeepIPC: Deeply Integrated Perception and Control

To deal with the implementation issues, the modified DeepIPC is forced to learn how to compensate for noise and inaccuracy of sensor measurement implicitly by mimicking expert behavior to achieve human-like autonomous driving [172] [173]. As shown in Fig. 5.1, the model must be able to safely avoid the obstacles by predicting navigational controls and waypoints correctly in the traversable area although the given route points (two white circles on the bottom-right image) are not located accurately in the local coordinate. DeepIPC processes multi-modal data that contain

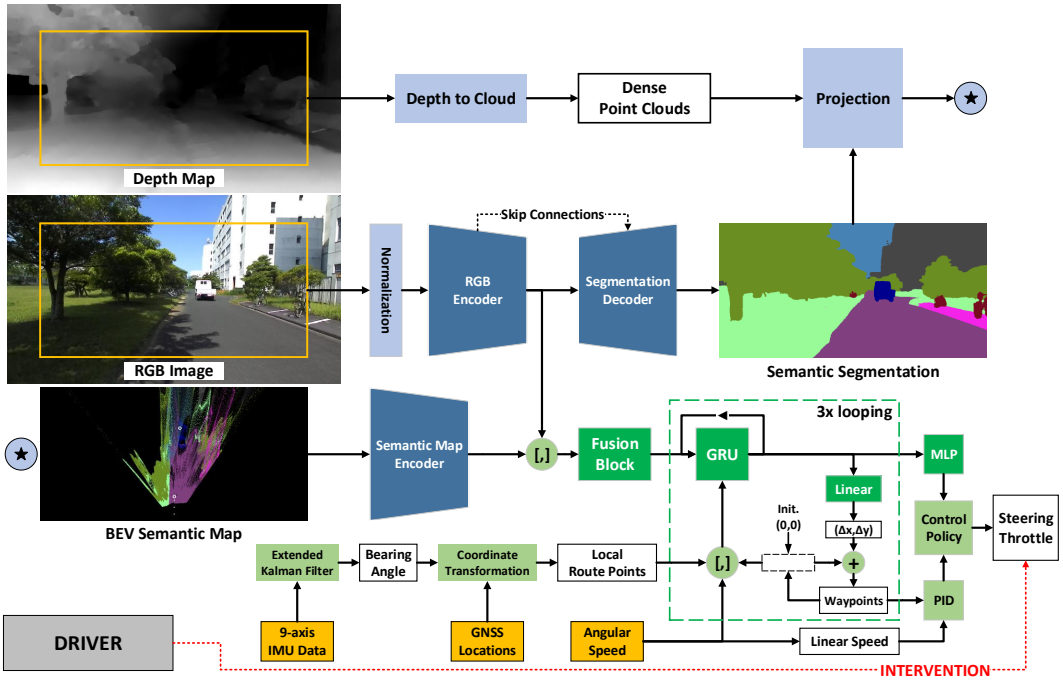


FIGURE 5.2: The architecture of the modified DeepIPC.

Blue blocks are considered as part of the perception module, while green blocks are considered as part of the controller module. Light-colored blocks are not trainable, while the darker ones are trainable.

several quantities needed to perceive the environment and drive the vehicle in one forward pass. The perception parts take an RGBD image to perform semantic segmentation and BEV semantic mapping. Simultaneously, the controller parts estimate waypoints and navigational controls based on the extracted perception features, the wheel's angular speeds, and route points. Unlike in an ideal simulated environment, DeepIPC must deal with real implementation issues. For example, it must compensate for the issue of inaccurate route points positioning caused by the inaccuracy of the GNSS receiver and IMU sensor. Then, there are also noises on the RGBD camera that can affect the scene understanding capability.

5.2.1 Network Architecture

The architecture of DeepIPC is similar to our previous work [55] that is composed of perception and controller parts. As shown in Fig. 5.2, the perception phase begins with semantic segmentation on RGB image with a standard encoder-decoder network enhanced with several skip connections [153]. The RGB encoder is made of Efficient Net B3 [151] while the decoder is composed of multiple convolution blocks where each block consists of (2×(3 × 3 convolution + batch normalization [116] + ReLU [117]) + bilinear interpolation) and a pointwise 1 × 1 convolution followed with sigmoid activation. Furthermore, we generate point clouds from the depth map and make projections with the predicted segmentation map to obtain a BEV semantic map with a coverage area of 24 meters to the front, left, and right from the vehicle location. Thus, the vehicle is always positioned at the bottom center of the BEV semantic map. Then, the BEV semantic map is encoded by an Efficient Net B1 encoder [151] to obtain its features. With this configuration, DeepIPC has both front and top perspectives to perceive the surrounding area.

In the controller module, both RGB and BEV semantic features are processed by a fusion block module which is composed of pointwise (1×1) convolution, global average pooling, and linear layer. This module is responsible for learning the relation between features from the front and top-view perspectives. Then, a gated recurrent unit (GRU) [83] is used to decode the latent features based on the measurement of the left and right wheel's angular speeds, predicted waypoints, and two route points that have been transformed into local BEV coordinates. The decoded features are processed further by a linear layer to predict Δx and Δy. Thus, the coordinate of the next waypoint can be calculated with (5.1).

$$x_{i+1}, y_{i+1} = (x_i + \Delta x), (y_i + \Delta y) \tag{5.1}$$

To be noted, the waypoints prediction process is looped over three times as there are three waypoints to be predicted. In the first loop, the waypoint is initialized with the vehicle position in the local BEV coordinate which is always at (0,0). In the end, the waypoints are translated into a set of navigational controls (steering and throttle) by two PID controllers in which their Kp, Ki, Kd parameters are tuned empirically. The final features used to predict the last waypoint are also processed by a multilayer perceptron (MLP) block to estimate the navigational controls directly. The final action that actually drives the vehicle is made by a control policy that combines both PID and MLP controls as shown in Algorithm 5.1. We set a confidence threshold of 0.1 as a minimum requirement for an agent to be able to drive the vehicle. This means that the prediction output on each navigational control must be greater than or equal to 0.1. This mechanism allows an agent to take control completely over the other agent and results in better maneuverability.

5.2.2 Model Improvement

Different from our previous work [55], DeepIPC is modified to improve its performance and deal with real-world implementation issues. First, as the input to the perception module, we consider a wider ROI of $H \times W = 512 \times 1024$ at the center of the RGBD image. Then, they are resized to $H \times W = 256 \times 512$ to reduce the computational load. With a wider coverage area, the model is expected to have a better scene understanding capability to perceive the environment. Second, as the input to the controller module, we feed the left and right wheel's angular speeds instead of the vehicle's linear speed. This information is expected to be helpful, especially during turning as the angular speed will be different on each wheel. Third, two route points are given instead of one route point at a time. Relying only on one route point is very risky due to the possibility of mislocation caused by GNSS and IMU inaccuracies that affect the global-to-local coordinate transformation. If the route point is mislocated, the model will likely fail to predict waypoints and navigational controls correctly. Besides that, giving two route points that have been transformed into local coordinates will give the model a better intuition in deciding whether the vehicle should drive straight or turn depending on the location of the route points. Fourth, we also modify the control policy by allowing an agent to take the steering control completely over the other agent for better maneuverability.

We feed the model with GNSS and IMU data to measure several quantities needed to perform global-to-local coordinate transformation precisely. To get the local BEV coordinate for each route point i, the relative distance Δx_i and Δy_i between vehicle location Ro and route point location Rp_i must be known. The distance can be estimated from the global longitude-latitude with (5.2) and (5.3).

Algorithm 5.1: Control Policy

$\Theta = \frac{Wp_1+Wp_2}{2}$; $\theta = \tan^{-1}\left(\frac{\Theta[1]}{\Theta[0]}\right)$
$\gamma = 1.75 \times ||Wp_1 - Wp_2||_F$; $\nu = \frac{(\omega_l+\omega_r)}{2} \times r$
$\mathbf{PID}_{ST} = \mathbf{PID}^{Lat}(\theta - 90)$; $\mathbf{PID}_{TH} = \mathbf{PID}^{Lon}(\gamma - \nu)$
. . . . . .
if $\mathbf{MLP}_{TH} \geq 0.1$ *and* $\mathbf{PID}_{TH} \geq 0.1$ **then**
  if $|\mathbf{MLP}_{ST}| \geq 0.1$ *and* $|\mathbf{PID}_{ST}| < 0.1$ **then**
    steering $= \mathbf{MLP}_{ST}$
  if $|\mathbf{MLP}_{ST}| < 0.1$ *and* $|\mathbf{PID}_{ST}| \geq 0.1$ **then**
    steering $= \mathbf{PID}_{ST}$
  else
    steering $= \beta_{00}\mathbf{MLP}_{ST} + \beta_{10}\mathbf{PID}_{ST}$
  throttle $= \beta_{01}\mathbf{MLP}_{TH} + \beta_{11}\mathbf{PID}_{TH}$
else if $\mathbf{MLP}_{TH} \geq 0.1$ *and* $\mathbf{PID}_{TH} < 0.1$ **then**
  steering $= \mathbf{MLP}_{ST}$; throttle $= \mathbf{MLP}_{TH}$
else if $\mathbf{MLP}_{TH} < 0.1$ *and* $\mathbf{PID}_{TH} \geq 0.1$ **then**
  steering $= \mathbf{PID}_{ST}$; throttle $= \mathbf{PID}_{TH}$
else
  steering $= 0$; throttle $= 0$
. . . . . .
$Wp_{\{1,2\}}$: first and second predicted waypoints
$\mathbf{MLP}_{\{ST,TH\}}$: steering and throttle estimated by MLP agent
$\omega_{\{l,r\}}$: left/right angular speed measured with rotary encoder
r: vehicle's wheel radius, 0.15 m
Θ: aim point, a middle point between Wp_1 and Wp_2
θ: heading angle derived from the aim point Θ
γ: desired speed, 1.75 $\times$ Frobenius norm of Wp_1 and Wp_2
ν: linear speed, the average of ω_l and ω_r multiplied by r
$\beta \in \{0, ..., 1\}^{2\times 2}$ is a set of control weights initialized with:
$\beta_{00} = \frac{\alpha_2}{\alpha_2+\alpha_1}$; $\beta_{10} = 1 - \beta_{00}$; $\beta_{01} = \frac{\alpha_3}{\alpha_3+\alpha_1}$; $\beta_{11} = 1 - \beta_{01}$
where $\alpha_1, \alpha_2, \alpha_3$ are loss weights tuned by MGN algorithm [54] (see Subsection 5.2.4)

$$\Delta x_i = (Rp_i^{Lon} - Ro^{Lon}) \times \frac{\mathcal{C}_e \times \cos(Ro^{Lat})}{360}, \tag{5.2}$$

$$\Delta y_i = (Rp_i^{Lat} - Ro^{Lat}) \times \frac{\mathcal{C}_m}{360}, \tag{5.3}$$

where $\mathcal{C}_e$ and $\mathcal{C}_m$ are earth's equatorial and meridional circumferences which are around 40,075 and 40,008 kilometers, respectively. Then, the route point coordinates $Rp_i^{(x,y)}$ can be obtained by applying a rotation matrix as in (5.4).

$$\begin{bmatrix} Rp_i^x \\ Rp_i^y \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ro}) & -\sin(\theta_{ro}) \\ \sin(\theta_{ro}) & \cos(\theta_{ro}) \end{bmatrix}^T \begin{bmatrix} \Delta x_i \\ \Delta y_i \end{bmatrix}, \tag{5.4}$$

where θ_{ro} is the vehicle's absolute orientation to the north pole (bearing angle). It is estimated by a 9-axis IMU sensor's built-in function based on Kalman filtering on 3-axial acceleration, angular speed, and magnetic field. The global-to-local route points transformation may not be so accurate due to sensor inaccuracy and noisy measurement. Hence, the model is forced implicitly to learn how to compensate for this issue by mimicking expert manipulation to navigational controls.

FIGURE 5.3: The area for experiments.

White circles are an example of a route that consists of start, finish, and route points. The area can be viewed in more detail at https://goo.gl/maps/9rXobdhP3VYdjXn48.

5.2.3 Dataset

Considering the advantage of imitation learning or behavior cloning as mentioned in Chapter 4, we adopt this approach and collect a considerable amount of expert driving records for training and validation (train-val) purposes [174] [175]. To create the dataset, we drive the vehicle at a speed of 1.25 m/s in a certain area inside our university, Toyohashi University of Technology, Japan. As shown in Fig. 5.3, the left region is used for the train-val, and the right region is used for the test. We consider two different experiment times which are noon and evening to vary the environmental conditions. For each condition, we record the driving data one time for the train-val and three times for the test. There are 12 routes in the train-val region and 6 routes in the test region. Each route is composed of several route points with a gap of 12 meters between each other. The model must follow the route points in driving the vehicle and completing the route. The observation is recorded at 4 Hz where one set of observations contains an RGBD image, GNSS location, 9-axis IMU measurement, the wheel's angular speed, and the level of steering and throttle. The devices used to retrieve the data are mentioned in Table 5.1, while their placement can be seen in Fig. 5.4.

DeepIPC predicts waypoints, navigational controls, and semantic segmentation maps. As for waypoints ground truth, we leverage the vehicle's trajectory where the vehicle's location in one second, two seconds, and three seconds in the future are considered as the waypoints to be predicted. Meanwhile, navigational controls ground truth can be obtained from the record of steering and throttle levels. To avoid time-consuming manual annotation, we use SegFormer [176] pre-trained on the Cityscapes dataset [68] to perform segmentation on all RGB images in twenty different classes as mentioned in Table 5.1. SegFormer is chosen for its awesome performance in the semantic segmentation task.

TABLE 5.1: Dataset Information

Conditions	Noon and evening
Total routes	12 (train-val) and 6 (test)
$\mathcal{N}$ Samples*	10151 (train), 9679 (val), 18975 (test)
Devices	WHILL C2 vehicle (+ rotary encoder) Stereolabs Zed RGBD camera U-blox Zed-F9P GNSS receiver Witmotion HWT905 9-axis IMU sensor
Object classes	None, road, sidewalk, building, wall, fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, bicycle

*$\mathcal{N}$ Samples is the number of observation sets. Each consists of an RGBD image, GNSS location, IMU measurement, the wheel's angular speed, and the level of steering and throttle.

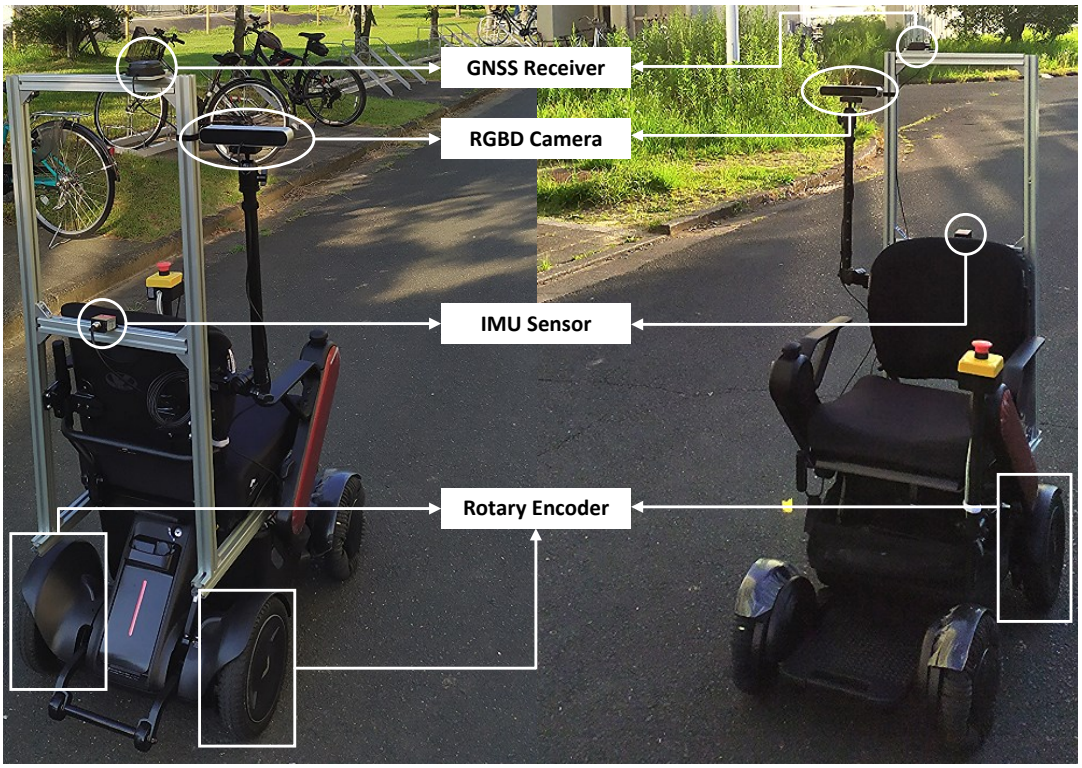


FIGURE 5.4: Sensor placement on a robotic vehicle.

5.2.4 Training Configuration

With using the multi-task learning paradigm, DeepIPC can be supervised by a combination of weighted loss functions as in (5.5).

$$\mathcal{L}_{MTL} = \alpha_0 \mathcal{L}_{SEG} + \alpha_1 \mathcal{L}_{WP} + \alpha_2 \mathcal{L}_{ST} + \alpha_3 \mathcal{L}_{TH}, \tag{5.5}$$

where $\alpha_{0,1,2,3}$ are loss weights tuned adaptively by an algorithm called modified gradient normalization (MGN) [54]. To learn semantic segmentation, we use a combination of pixel-wise cross entropy and dice loss as in (5.6).

$$\mathcal{L}_{SEG} = \left(\frac{1}{N}\sum_{i=1}^{N} y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)\right) + \left(1 - \frac{2|\hat{y} \cap y|}{|\hat{y}| + |y|}\right), \tag{5.6}$$

where N is the total elements at the last layer of the segmentation decoder, while y_i and $\hat{y}_i$ are ground truth and prediction of element i. Then, we use L1 loss to supervise waypoints prediction as in (5.7).

$$\mathcal{L}_{WP} = \frac{1}{M}\sum_{i=1}^{M} |\hat{y}_i - y_i|, \tag{5.7}$$

where M is equal to 6 as there are three predicted waypoints that have x,y coordinates for each. Similarly, we also use L1 loss to supervise navigational controls prediction as in (5.8). However, averaging is not needed as there is only one element for each output (steering and throttle).

$$\mathcal{L}_{\{ST,TH\}} = |\hat{y} - y| \tag{5.8}$$

The model is implemented with PyTorch deep learning framework [122] and trained on NVIDIA RTX 3090 with a batch size of 8. We use Adam optimizer [127] with a decoupled weight decay of 0.001 [155] to train the model until convergence. The initial learning rate is set to 0.0001 and divided by 2 if the validation loss $\mathcal{L}_{MTL}$ is not dropping in 5 epochs in a row. To avoid excessive computation, the training is stopped if there is no improvement in 30 epochs in a row. We attach the training log and its explanation in Appendix A.2.

5.3 Experiment and Analysis

In this section, we explain the evaluation and metrics used to justify the model performance. Then, we provide the discussion and analysis of the experiment results.

5.3.1 Evaluation and Scoring

DeepIPC is evaluated under two conditions with varying cloud intensity with two different tests namely offline and online tests. For each condition, the final score is obtained by averaging the scores from three experimental results. In the offline test, the model is deployed to predict driving records. Then, its performance on each task is calculated by a specific metric function. To evaluate waypoints and navigational controls, we use mean absolute error (MAE) or L1 loss as in (5.7) and (5.8). Meanwhile, we compute intersection over union (IoU) as in (5.9) for evaluating the segmentation performance.

$$IoU_{SEG} = \frac{|\hat{y} \cap y|}{|\hat{y} \cup y|} \tag{5.9}$$

We define the best model by the lowest total metric (TM) score as formulated with (5.10). This formula only combines semantic segmentation IoU and navigational controls estimation MAE. Depth MAE and waypoints MAE are excluded since not every model has these outputs.

$$TM = (1 - IoU_{SEG}) + MAE_{ST} + MAE_{TH} \tag{5.10}$$

Algorithm 5.2: Route points to Commands Conversion

if $Rp_1^x \leq -4m$ *or* $Rp_2^x \leq -8m$ **then**
  command = turn left
else if $Rp_1^x \geq 4m$ *or* $Rp_2^x \geq 8m$ **then**
  command = turn right
else
  command = go straight

$Rp_{\{1,2\}}^x$: the route point's x position in the local coordinate (BEV space)

TABLE 5.2: Model Specification

Model	Total Parameters↓	Model Size ↓	Input/Sensor	Output
Huang et al. [86]	74953258	300.196 MB	RGBD, High-level commands	Segmentation, Steering, Throttle
AIM-MT [72]	27967063	112.078 MB	RGB, GNSS, 9-axis IMU, Rotary encoder	Segmentation, Depth, Waypoints, Steering, Throttle
DeepIPC	20983128	84.972 MB	RGBD, GNSS, 9-axis IMU, Rotary encoder	Segmentation, BEV Semantic, Waypoints, Steering, Throttle

AIM-MT [72] is implemented based on the codes shared in the author's repository at https://github.com/autonomousvision/neat. Meanwhile, Huang et al.'s model [86] is implemented based on the explanation written in the paper. All models are deployed on a laptop powered with NVIDIA GTX 1650 GPU in performing real-world autonomous driving. As the models can run smoothly during evaluation, we believe that calculating their inference speeds is not necessary.

In the online test, the model is deployed to drive a vehicle by following a set of routes. Unlike in our previous work [55], the vehicle is prevented from colliding with other objects to avoid unnecessary damage. Therefore, we determine the drivability score by counting driver interventions needed to prevent collisions.

In addition, we conduct a comparative study with some recent models to get a clearer performance justification. Table 5.2 shows the specification of the models for comparison. DeepIPC is preferred for deployment as it has the smallest model as it has the lowest number of parameters. We evaluate a model proposed by Huang et al. [86] that also takes RGB images and depth maps but with a different fusion strategy. This model uses high-level commands in selecting a command-specific controller. Hence, we generate these commands automatically based on the route point position in the local coordinate using a certain rule as described in Algorithm 5.2. We also evaluate AIM-MT [72] which only takes RGB images and predicts multiple vision tasks for extra supervision. By performing more vision tasks, the perception module can provide better features for the controller. For a fair comparison, we slightly modify both models to process the same data as provided to DeepIPC.

5.3.2 Offline Test

The offline test is used to evaluate the model's performance in handling multiple perception and control tasks simultaneously. All models are deployed to predict driving records and evaluated with multi-task and task-wise scoring. The test dataset is recorded three times in a completely different area from the train-val dataset. Each record is taken on different days to vary the conditions.

Table 5.3 shows that DeepIPC achieves the best performance by having the lowest total metric score in all conditions. However, all models including DeepIPC have performance degradation in the evening. This means that doing inference in the low light condition is harder than in the normal condition. Specifically, in the segmentation task, DeepIPC has a higher IoU than AIM-MT even though it does not

TABLE 5.3: Multi-task Performance Score

Condition	Model	Total Metric↓	IoU_{SEG} ↑	MAE_{DE} ↓	MAE_{WP} ↓	MAE_{ST} ↓	MAE_{TH} ↓
Noon	Huang et al. [86]	0.4778 ± 0.0281	0.8300	-	-	0.2422	0.0484
	AIM-MT [72]	0.2932 ± 0.0300	0.8863	0.0593	0.0983	0.1734	**0.0061**
	DeepIPC	**0.2807 ± 0.0335**	**0.8899**	-	**0.0683**	**0.1632**	0.0074
Evening	Huang et al. [86]	0.4875 ± 0.0453	0.7952	-	-	0.2384	0.0443
	AIM-MT [72]	0.3088 ± 0.0346	0.8578	0.0669	0.0931	0.1639	**0.0026**
	DeepIPC	**0.3030 ± 0.0369**	**0.8623**	-	**0.0645**	**0.1611**	0.0041

perform depth estimation for extra supervision that can enhance the RGB encoder. Thanks to the end-to-end learning strategy where the segmentation prediction can be processed further through the encoding and decoding process of the BEV semantic map. Therefore, the segmentation decoder receives a more useful gradient signal to tune the network weights properly. Meanwhile, Huang et al.'s model has the worst segmentation performance that is caused by conflicting features as its perception module fuses RGB images and depth maps from the early perception stage.

In the waypoints prediction task, DeepIPC has a lower MAE compared to AIM-MT. Thanks to the BEV semantic features, DeepIPC can distinguish free and occupied areas easily from the top-view perspective. Thus, it can properly estimate the waypoints which are also laid in BEV space. Although AIM-MT predicts four waypoints and DeepIPC only predicts three waypoints, it is still considered a fair comparison because the MAE formula averages the error across all predictions. The reason the AIM-MT predicts four waypoints is to let its controller module have more learning experiences in estimating the waypoints correctly. However, DeepIPC still performs better as its controller module gets boosted by BEV semantic features and fed with angular speed measurement which enhances its intuition.

In the navigational controls estimation task, DeepIPC also has the best performance in line with the waypoints prediction result. The MLP agent can leverage useful features encoded from both RGB and BEV semantic maps. Therefore, the MLP agent can perform as well as the PID agent in estimating steering and throttle. With two different agents considering various aspects of driving, more appropriate action can be decided. Compared to AIM-MT, DeepIPC is better at estimating the steering but worse at estimating the throttle. Yet, it can be said that DeepIPC is better than AIM-MT considering that better steering control is more important than better throttle control in low-speed driving. Meanwhile, Huang et al.'s model performs the worst as its controller module gets stuck with certain behavior. Be noted that the offline test results can be different from the online test results. This is because any predictions will not affect the next states as they are prerecorded.

5.3.3 Online Test

The purpose of the online test is to evaluate the model's drivability. The model must drive the vehicle safely by following a set of route points while avoiding obstacles (e.g., a vehicle stopped on the left side of the road). The experiment is conducted three times for each condition. The experiment is conducted on different days to vary the situations. The performance is evaluated based on the average intervention count and intervention time. The less the driver does intervention means the better the driving performance. For a fair comparison, the experiments for all models are monitored by the same driver. Thus, each intervention is based on the same perspective of the degree of danger. Some driving records can be seen in Fig. 5.5.

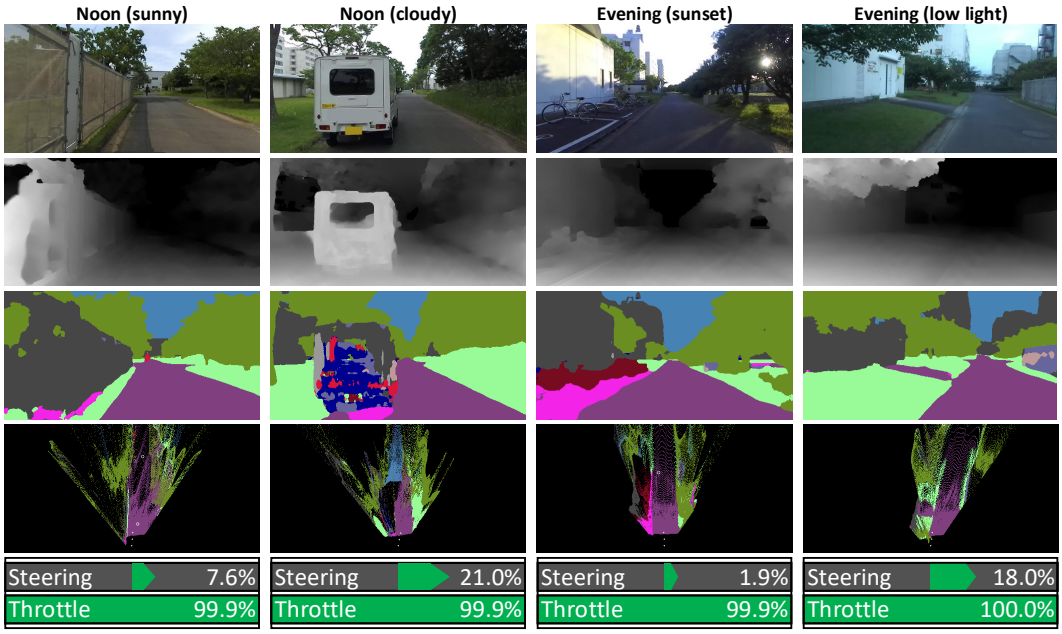


FIGURE 5.5: Driving footage.

See the driving record video at https://youtu.be/AiKotQ-lAzw for more details, including failure cases where we intervene to avoid collisions. Sunny noon: DeepIPC makes a small steering adjustment to the right as the vehicle is too close to the terrain. Cloudy noon: Although DeepIPC cannot segment the car properly, it can avoid collision as it knows that the left side is occupied. Sunset evening: DeepIPC makes a small steering adjustment to keep on its lane. Low light evening: We intervene in DeepIPC to avoid driving off-road on the vegetation as it keeps the throttle maximum and fails to make a right turn.

TABLE 5.4: Drivability Score

Condition	Model	Intervention↓ Count	Intervention↓ Time (secs)
Noon	Huang et al. [86]	1.8889 ± 0.4157	5.6039 ± 1.7272
	AIM-MT [72]	2.2778 ± 0.3425	4.2161 ± 0.8380
	DeepIPC	**1.1111 ± 0.3928**	**2.3092 ± 0.9841**
Evening	**Huang et al. [86]**	**1.6111 ± 0.2079**	4.5532 ± 0.2160
	AIM-MT [72]	2.6667 ± 0.1361	4.6736 ± 0.4293
	DeepIPC	1.8889 ± 0.3928	**4.2286 ± 0.6102**

Table 5.4 shows that DeepIPC achieves the best drivability at noon where it has the lowest intervention count and intervention time. Meanwhile, DeepIPC is comparable to Huang et al.'s model in the evening where it achieves the lowest intervention time but has a higher intervention count. Keep in mind that a model with a lower intervention count can have a longer intervention time. For example, a model that fails to make a turn and going to collide with terrain or sidewalk needs more correction time than a model that makes a small deviation on a straight path. Hence, it depends on the degree of danger in which the collision is going to happen. Based on the intervention time per intervention count, it is obvious that Huang et al.'s model needs more correction time for each intervention which means that it has the highest danger level compared to other models.

Furthermore, in a comparison of drivability in the evening, DeepIPC and AIM-MT perform worse than Huang et al.'s model. In line with the offline result, the model that mainly takes RGB images failed to perceive the environment in the evening as the provided image is not as clearly visible as when driving at noon. On the contrary, Huang et al.'s model become better as it can leverage the information from the depth map that is concatenated with the RGB image from the beginning of the perception phase. This means that although the early fusion strategy causes conflicting features for semantic segmentation, it is useful for driving in low-light conditions. Moreover, even though Huang et al.'s model shows inferior performance on navigational controls estimation in the offline test, its drivability can be said good enough for performing real-world automated driving in the evening with lower traffic compared when driving at noon. Regardless of its comparable performance with DeepIPC in the evening, this exposes the limitation of imitation learning for a model that purely relies on human behavior (by directly predicting steering and throttle levels) without considering another driving aspect that can be obtained from predicting future trajectories in the form of waypoints location in the local coordinate.

5.4 Findings

In this Chapter, we present a modified version of DeepIPC, an end-to-end model that can drive a vehicle in real environments. The model is evaluated by predicting a set of driving records and performing automated driving. Furthermore, a comparative study with some recent models is conducted to justify its performance. Based on the experimental results, we disclosed several findings as follows.

- In line with our previous work [55], the BEV semantic feature is proven can improve the model performance in predicting waypoints and navigational controls. With a better perception, the model can leverage useful information which results in better drivability.
- Driving under low light conditions is harder than in the normal condition, especially for DeepIPC and AIM-MT which only rely on RGB images at the early perception stage. Meanwhile, Huang et al.'s model can tackle this issue as it fuses RGB and depth features earlier.
- DeepIPC can be said as the best model considering its performance and the number of parameters in its architecture.
- Since the experiments discussed in this Chapter are the real-world implementation of the works in Chapter 4, we also disclose that the end-to-end imitation learning approach is also useful for real-world autonomous driving. Furthermore, this also exposes the usefulness of the end-to-end behavior cloning technique for multi-input multi-output models.

Chapter 6

LiDAR-based End-to-end Autonomous Driving

As discussed in Chapter 5, DeepIPC can drive a robotic vehicle in real environments and achieve the best performance. However, its drivability get worsen in the evening due to visibility issue. Although the RGBD camera can capture high-resolution images, it fails to provide clear information caused by poor illumination conditions. Therefore, we present DeepIPCv2, an improved version of DeepIPC that perceives the environment using a LiDAR sensor for more robust scene understanding. DeepIPCv2 takes a set of LiDAR point clouds for its main perception input. As point clouds are not affected by illumination changes, they can provide a clear observation of the surroundings no matter what the condition is. This results in a better scene understanding and stable features provided by the perception module to support the controller module in estimating navigational controls properly. To evaluate its performance, we conduct several tests by deploying the model to predict a set of driving records and perform real automated driving under three different conditions, including driving at night. We also conduct ablation and comparative studies with some recent models to justify its performance. Based on the experimental results, DeepIPCv2 shows a robust performance by achieving the best drivability in all conditions, including when driving at night.

6.1 LiDAR-powered Perception

LiDAR is a sensor that is considered to be more robust than an RGB camera when dealing with poor illumination conditions. Unlike RGB images, the point clouds are not affected by the illumination changes since the LiDAR has its own lasers as the light source to observe the environment [177] [178]. Furthermore, together with plenty of point cloud segmentation models and projection techniques, many kinds of data representations can be formed to provide meaningful information [179] [180]. Hence, we consider a point cloud segmentation model to support the perception.

To date, there are plenty of works in the development of point cloud segmentation models. In the Semantic KITTI dataset [181], the current state-of-the-art in the semantic point cloud segmentation task is achieved by a model named 2DPASS [182]. However, its performance needs to be justified further in a very poor illumination condition as this model uses RGB images to assist the segmentation process. A point cloud segmentation model that only uses a LiDAR is proposed by Hou et al. [183] which is currently the runner-up in the semantic point cloud segmentation challenge. Although this model has a great performance, its size and latency are not suitable for performing real-time inference on a device with limited computation power. For deployment purposes, we need to consider the trade-off between

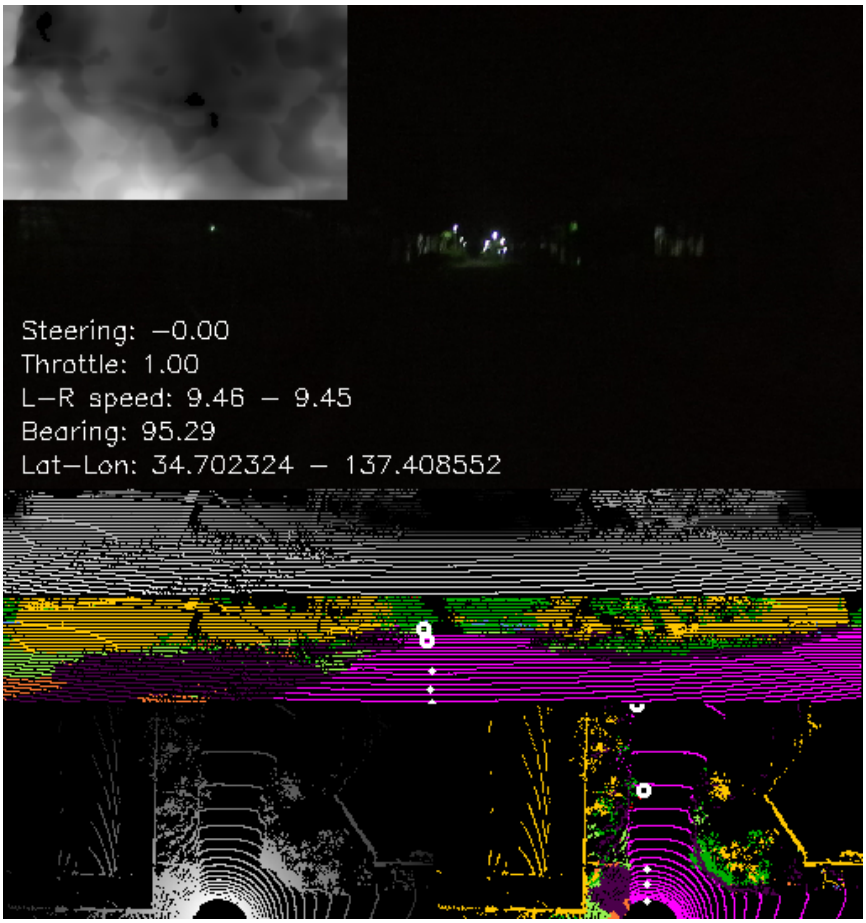


FIGURE 6.1: The inputs and outputs of DeepIPCv2.

DeepIPCv2 perceives the environment by encoding a set of segmented point clouds that are projected into front and top-view perspectives. Then, the extracted features are used to estimate waypoints (white dots) and navigational controls to drive the vehicle following the route points (white hollow circles). Meanwhile, the RGBD image is only for record purposes. It shows how the RGBD camera fails in capturing surrounding information as it cannot provide a clearly visible set of RGB image and depth map. Instead, DeepIPCv2 employs a LiDAR sensor to provide point clouds that are not affected by poor illumination conditions. The detailed architecture of DeepIPCv2 can be seen in Fig. 6.2.

speed and performance. Therefore, a model with great performance but causing a huge computation load is not preferable. Since we also seek robustness, the model must only use LiDAR in performing point cloud segmentation. Hence, we select PolarNet [111], a lightweight model that has an acceptable performance.

6.2 DeepIPCv2: Highly Robust Perception and Control

The improvisation of DeepIPCv2 architecture is intended to deal with poor illumination conditions. We modify the perception module by replacing RGBD encoders with LiDAR encoders. As shown in Fig. 6.1, the RGBD camera fails to provide a clearly visible set of RGB image and depth map. Hence, DeepIPCv2 uses a LiDAR sensor and employs a point cloud segmentation model to perceive the environment. This enables better reasoning as the model can distinguish traversable and non-traversable areas easily and avoid collision by knowing the existence of other

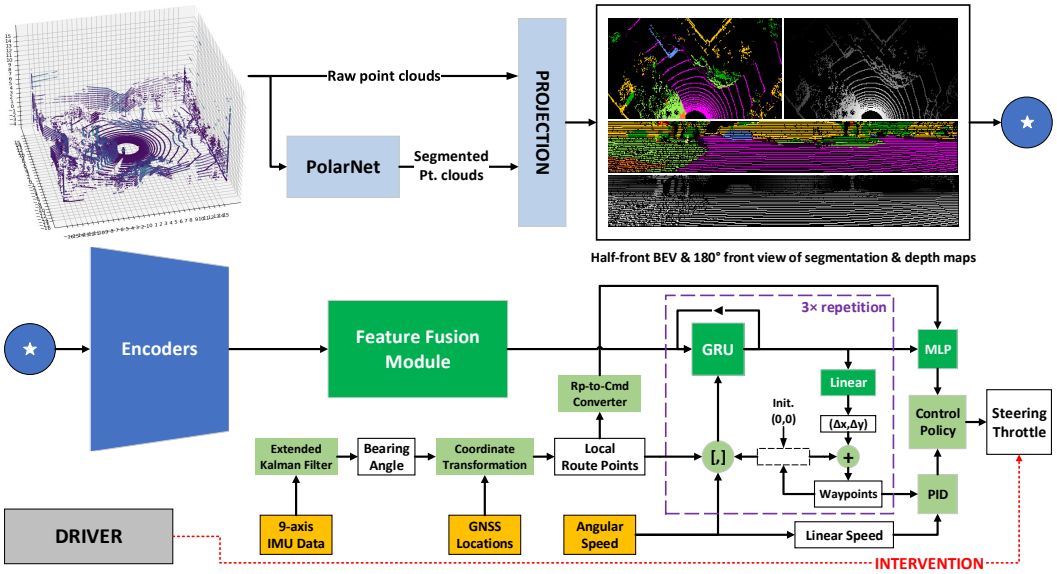


FIGURE 6.2: The architecture of DeepIPCv2.

The blue and green blocks are the perception and controller modules respectively. Darker blocks are trainable, while light-colored blocks are not. In the perception module, PolarNet [111] is employed to support point cloud segmentation. Then, the architecture of encoders and feature fusion modules can be seen in Fig. 6.3.

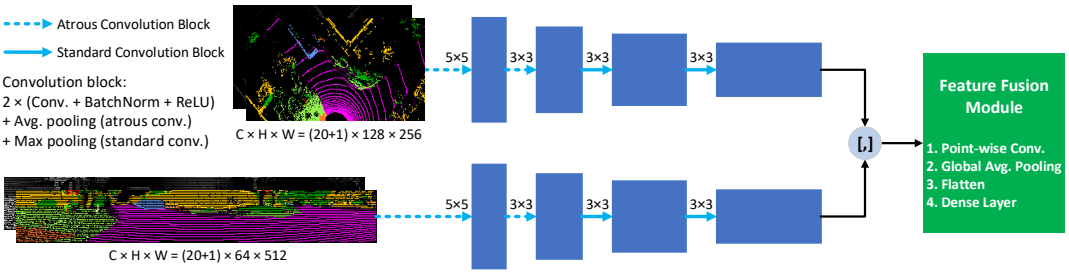


FIGURE 6.3: The encoders and the feature fusion module.

We use atrous convolution blocks [184] with different kernel sizes and dilation rates to capture low-level features from the projected point clouds that have vacant regions. Then, both top and front features are fused and extracted further by the feature fusion module.

objects around the ego vehicle. Hence, the perception module can provide stable and better features to the controller module for estimating waypoints and navigational control. Thus, DeepIPCv2 can maintain its drivability performance even when driving at night. We also modify the controller module by adding a set of command-specific multi-layer perceptrons (MLP) to improve its maneuverability.

6.2.1 Network Architecture

Similar to DeepIPC [56], DeepIPCv2 is also a model that handles perception and control tasks simultaneously. However, unlike DeepIPC which takes an RGBD image, DeepIPCv2 takes a set of LiDAR point clouds to perceive the environment. Since LiDAR is not affected by poor illumination conditions, the perception module becomes more robust and can provide stable features to the controller module. Thus, the model can estimate waypoints and navigation control properly even

when driving at night. As shown in Fig. 6.2, DeepIPCv2 employs PolarNet [111], a light-weight point cloud segmentation model pre-trained on the Semantic KITTI dataset [181] to segment LiDAR point clouds into twenty object classes as mentioned in Table 6.1. Based on our previous work [55], perceiving the environment from more perspectives can improve perception and lead to better drivability. Hence, we project segmented point clouds to form one hot-encoded image-like array that shows front-view and bird's eye-view (BEV) perspectives of the surrounding area. Each array is expressed as $\mathbb{R} \in \{0,1\}^{C \times H \times W}$, where $H \times W$ is the spatial dimension with $H \times W = 64 \times 512$ for the front-view array and $H \times W = 128 \times 256$ for the BEV array. Meanwhile, $C = 21$ represents the number of channels that are responsible for twenty object classes and a logarithmic depth of the point clouds. In forming the BEV array, we consider an area of 16 meters to the front, left, and right of the vehicle. Meanwhile, for the front-view array, we consider all point clouds in front of the vehicle forming a 180-degree field of view.

To process these arrays, we use two different encoders that are made of atrous and standard convolution blocks as shown in Fig. 6.3. Atrous convolution blocks [184] are used to deal with some vacant regions in the projected point clouds at the early encoding process. As the kernel sizes and dilation rates can be adjusted, an atrous convolution layer is more suitable than a standard convolution layer for extracting the features. Then, we also configure the pooling size after each convolution block to match the output size of both encoders. With this configuration, DeepIPCv2 has a better scene understanding capability as it can perceive from two different perspectives that clearly show traversable and non-traversable regions. Later, we conduct an ablation study by creating two additional model variants. The first variant only takes the logarithmic depth point clouds, while the other one only takes the segmented point clouds. After obtaining the best variant, we also conduct an extensive ablation study by comparing it with another variant that perceives the surroundings with one perspective, front or BEV. This is necessary to justify the importance of multi-view perception for better reasoning.

The control phase begins by fusing both high-level perceptions features to produce a latent space composed of 192 feature elements that encapsulate the information of the surrounding based on two perspectives of view. This process is done by the feature fusion module that consists of a point-wise convolution layer, a global average pooling layer, and a dense layer. Then, we use the first and second route points, the left and right wheel's angular speed, and predicted waypoints to bias the latent space in the gated recurrent unit (GRU) layer [83]. Finally, the biased latent space is decoded further by a set of command-specific multi-layer perceptrons (MLP) to estimate navigational control directly and by two linear layers to predict waypoints that will be translated into navigational control by a set of two PID controllers. To be noted, both MLP and PID controllers assume the model of the robotic vehicle as a nonholonomic unicycle since it has motorized rear wheels and omnidirectional front wheels. Thus, it cannot perform translational movement on the lateral axis, but it can move only along its longitudinal axis (forward and backward) and can rotate around a vertical axis passing through its center. As shown in Fig. 6.2, the process inside the purple box is looped three times as DeepIPCv2 predicts three waypoints. Two linear layers are used to predict Δx and Δy between the current waypoint and the next waypoint. Thus, the exact coordinate of the next waypoint can be calculated with (6.1).

$$x_{i+1}, y_{i+1} = (x_i + \Delta x), (y_i + \Delta y) \tag{6.1}$$

Algorithm 6.1: Control Policy

$\Theta = \frac{Wp_1+Wp_2}{2}$; $\theta = \tan^{-1}\left(\frac{\Theta[1]}{\Theta[0]}\right)$
$\gamma = 1.75 \times ||Wp_1 - Wp_2||_F$; $\nu = \frac{(\omega_l+\omega_r)}{2} \times r$
.....
if $Rp_1^x \leq -4m$ *or* $Rp_2^x \leq -8m$ **then**
  $Cmd = 2$ (turn right)
else if $Rp_1^x \geq 4m$ *or* $Rp_2^x \geq 8m$ **then**
  $Cmd = 1$ (turn left)
else
  $Cmd = 0$ (go straight)
$\mathbf{MLP}_{\{ST,TH\}} = \mathbf{MLP}^{Cmd}(\mathcal{Z})$
$\mathbf{PID}_{ST} = \mathbf{PID}^{Lat}(\theta - 90)$; $\mathbf{PID}_{TH} = \mathbf{PID}^{Lon}(\gamma - \nu)$
.....
if $\mathbf{MLP}_{TH} \geq 0.1$ *and* $\mathbf{PID}_{TH} \geq 0.1$ **then**
  if $|\mathbf{MLP}_{ST}| \geq 0.1$ *and* $|\mathbf{PID}_{ST}| < 0.1$ **then**
    steering $= \mathbf{MLP}_{ST}$
  if $|\mathbf{MLP}_{ST}| < 0.1$ *and* $|\mathbf{PID}_{ST}| \geq 0.1$ **then**
    steering $= \mathbf{PID}_{ST}$
  else
    steering $= \beta_{00}\mathbf{MLP}_{ST} + \beta_{10}\mathbf{PID}_{ST}$
  throttle $= \beta_{01}\mathbf{MLP}_{TH} + \beta_{11}\mathbf{PID}_{TH}$
else if $\mathbf{MLP}_{TH} \geq 0.1$ *and* $\mathbf{PID}_{TH} < 0.1$ **then**
  steering $= \mathbf{MLP}_{ST}$; throttle $= \mathbf{MLP}_{TH}$
else if $\mathbf{MLP}_{TH} < 0.1$ *and* $\mathbf{PID}_{TH} \geq 0.1$ **then**
  steering $= \mathbf{PID}_{ST}$; throttle $= \mathbf{PID}_{TH}$
else
  steering $= 0$; throttle $= 0$
.....
$Rp^x_{\{1,2\}}$: route point's x position in the local coordinate
$Wp_{\{1,2\}}$: first and second waypoints
$\mathcal{Z}$: GRU's latent space
$\omega_{\{l,r\}}$: left/right angular speed (rad/s)
r: vehicle's rear wheel radius (0.15 m)
Θ: aim point, a middle point between Wp_1 and Wp_2
θ: heading angle derived from the aim point Θ
γ: desired speed, 1.75 $\times$ Frobenius norm of Wp_1 and Wp_2
ν: linear speed (m/s), the mean of ω_l and ω_r multiplied by r
$\beta \in \{0, ..., 1\}^{2\times 2}$ is a set of control weights initialized with:
$\beta_{00} = \frac{\alpha_1}{\alpha_1+\alpha_0}$; $\beta_{10} = 1 - \beta_{00}$; $\beta_{01} = \frac{\alpha_2}{\alpha_2+\alpha_0}$; $\beta_{11} = 1 - \beta_{01}$
where $\alpha_0, \alpha_1, \alpha_2$ are loss weights tuned by MGN algorithm [54] (see Subsection 6.2.3)

To predict the first waypoint, the current waypoint is initialized with the vehicle position in the local coordinate which is always at (0,0). Then, the waypoints are processed by two PID controllers to produce a set of navigational control consisting of steering and throttle levels. Besides using PID controllers, DeepIPCv2 also predicts navigational control directly by decoding biased latent using MLP. However, unlike DeepIPC which employs only one MLP, DeepIPCv2 employs a set of command-specific MLPs for better maneuverability as demonstrated by Huang et. al [86]. Each of the command-specific MLPs act as a task-specific decoder that receives the same features from the same encoder. Then, since each decoder treats each action (turn left, turn right, or go straight) independently, the model has better

maneuverability as it has more focus by deploying a dedicated MLP for each action. Moreover, this configuration can also deal with the imbalance number of actions in the driving records (e.g. the number of observation sets for go straight is larger than turn left or turn right). Meanwhile, the commands are generated automatically based on the route point's x position. The rule that generates the command and the policy which outputs the final action is summarized on Algorithm 6.1.

Furthermore, other measurement quantities and formulas are needed to transform the route points from the global GNSS coordinate to the local coordinate where the vehicle is always positioned at (0,0). To obtain the local coordinate for each route point i, we need the relative distance Δx_i and Δy_i between vehicle location Ro and route point location Rp_i. Using the information of global longitude-latitude given by the GNSS receiver, the relative distance can be calculated with (6.2) and (6.3).

$$\Delta x_i = (Rp_i^{Lon} - Ro^{Lon}) \times \frac{\mathcal{C}_e \times \cos(Ro^{Lat})}{360}, \tag{6.2}$$

$$\Delta y_i = (Rp_i^{Lat} - Ro^{Lat}) \times \frac{\mathcal{C}_m}{360}, \tag{6.3}$$

where $\mathcal{C}_e$ and $\mathcal{C}_m$ are earth's equatorial and meridional circumferences which are 40,075 and 40,008 km, respectively. Then, the local coordinate of each route point $Rp_i^{(x,y)}$ can be obtained by applying a rotation matrix as in (6.4).

$$\begin{bmatrix} Rp_i^x \\ Rp_i^y \end{bmatrix} = \begin{bmatrix} \cos(\theta_{ro}) & -\sin(\theta_{ro}) \\ \sin(\theta_{ro}) & \cos(\theta_{ro}) \end{bmatrix}^T \begin{bmatrix} \Delta x_i \\ \Delta y_i \end{bmatrix}, \tag{6.4}$$

where θ_{ro} is the vehicle's absolute orientation to the north pole (bearing angle). The bearing angle is estimated by the extended Kalman filter (EKF) based on the measurement of 3-axial acceleration, angular speed, and magnetic field retrieved from a 9-axis IMU sensor. To be noted, due to the GNSS inaccuracy and noisy IMU measurements, the global-to-local transformation may not be perfect. Thus, the model is expected to learn how to compensate for this issue during the training process.

6.2.2 Dataset

As explained in Chapter 4 and Chapter 5, a dataset that consists of expert driving records is needed for imitation learning or behavior cloning process [185] [186]. Similar to our experiment described in Chapter 5, we record the observation data to create the dataset for training, validation, and testing (train-val-test). One set of observations is composed of an RGBD image, GNSS location, 9-axis IMU measurement, the wheel's angular speed, and the level of steering and throttle. We collect these data by driving a robotic vehicle at a speed of 1.25 m/s in an area inside our university, Toyohashi University of Technology, Japan as shown in Fig. 6.4. In order to vary the experiment conditions, we record the driving data at noon, in the evening, and at night with different cloud intensities. In the train-val area, there are 12 different routes where the driving data is recorded one time for each condition. Meanwhile, in the test area, there are 6 different routes where the driving data is recorded three times for each condition. Each route has a set of route points with 12 meters gap that shows the path from the start point to the finish point. The model must drive the vehicle by following this path to complete the route.

Recorded at a rate of 4 Hz, one sample of observation data is composed of a set of LiDAR point clouds, GNSS latitude-longitude, 9-axis IMU measurement, left and right wheel's angular speeds, and the level of steering and throttle. We also

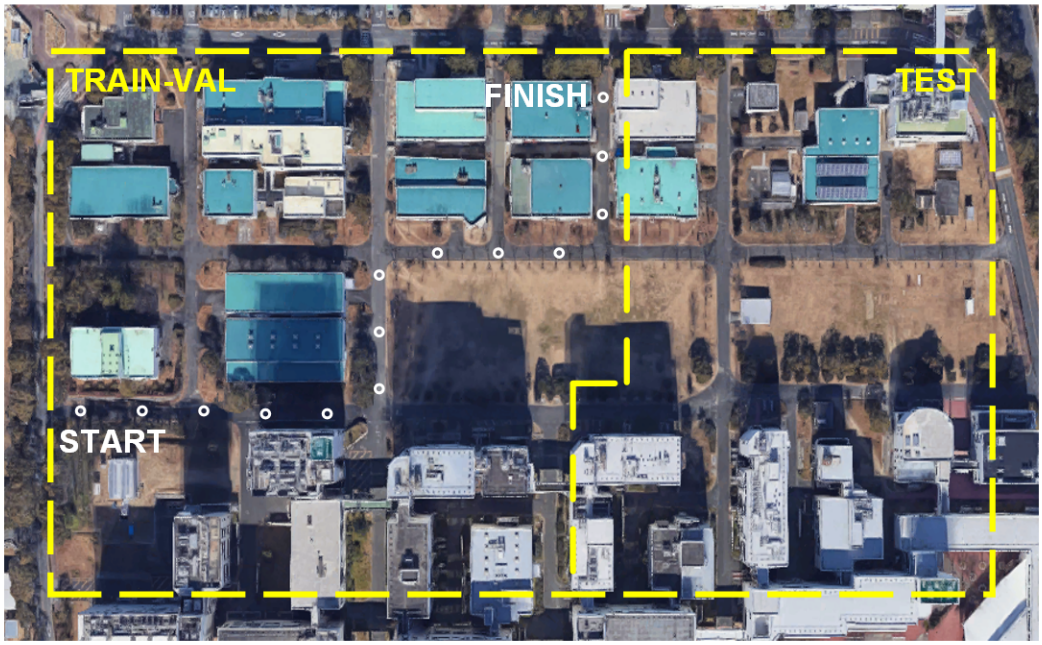


FIGURE 6.4: The area for experiments.

White hollow circles represent a route that consists of a start, finish, and route points. The area can be viewed in more detail at https://goo.gl/maps/9rXobdhP3VYdjXn48.

record the RGB image which is used by another model for comparison. Then, as the ground truth for the waypoints prediction task, we use the vehicle's trajectory location in one second, two seconds, and three seconds in the future (relative to the vehicle's current location). The trajectory is estimated by a built-in IMU-based odometry algorithm embedded in the robotic vehicle. Meanwhile, as the ground truth for the navigational control estimation task, we use the record of steering and control levels at the time. The devices used to retrieve the data are mentioned in Table 6.1. Meanwhile, how they are mounted on the vehicle can be seen in Fig. 6.5.

6.2.3 Training Configuration

A multi-task loss function used to supervise DeepIPCv2 is formulated with (6.5).

$$\mathcal{L}_{MTL} = \alpha_0 \mathcal{L}_{WP} + \alpha_1 \mathcal{L}_{ST} + \alpha_2 \mathcal{L}_{TH}, \tag{6.5}$$

where $\alpha_{0,1,2}$ are loss weights tuned adaptively by an algorithm called modified gradient normalization (MGN) [54] to ensure that all tasks can be learned at the same pace. To supervise waypoints prediction, we use L1 loss as in (6.6).

$$\mathcal{L}_{WP} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{6.6}$$

where N is equal to 6 as there are three waypoints that have x,y elements in the local coordinate. Meanwhile, y_i and $\hat{y}_i$ are the ground truth and the prediction of component i respectively. Similarly, we also use L1 loss to supervise navigational control estimation formulated with (6.7).

$$\mathcal{L}_{\{ST,TH\}} = |\hat{y} - y| \tag{6.7}$$

TABLE 6.1: Dataset Information

Conditions	Noon, evening, and night
Total routes	12 (train-val) and 6 (test)
$\mathcal{N}$ Samples	19781 (train), 9695 (val), 29123 (test)
Devices	WHILL model C2 (+ rotary encoder) Velodyne LiDAR HDL-32e Stereolabs Zed RGBD camera U-blox Zed-F9P GNSS receiver Witmotion HWT905 9-axis IMU sensor
Object classes	None, car, bicycle, motorcycle, truck, other vehicle, person, bicyclist, motorcyclist, road, parking, sidewalk, ground, building, fence, vegetation, trunk, terrain, pole, traffic sign

*$\mathcal{N}$ Samples is the number of observation sets. Each consists of an RGBD image, GNSS location, IMU measurement, the wheel's angular speed, and the level of steering and throttle.

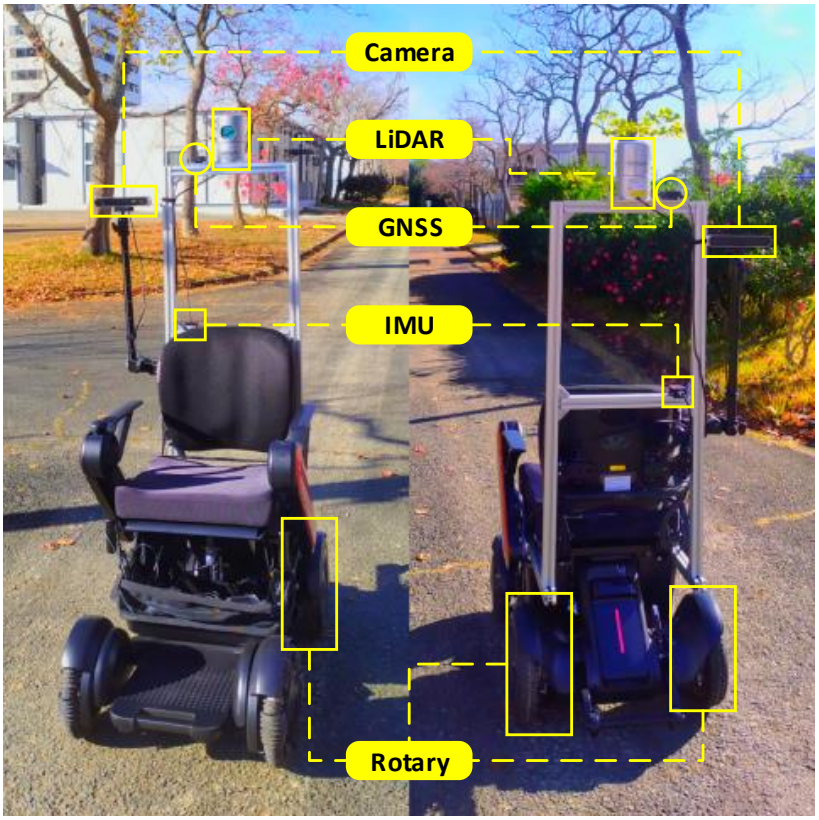


FIGURE 6.5: Sensor placement on a robotic vehicle.

Keep in mind that there is no averaging process as there is only one element for each output (steering and throttle). The model is implemented with PyTorch framework [122] and trained on NVIDIA RTX 3090 with a batch size of 10. We use Adam optimizer [127] with decoupled weight decay of 0.001 [155]. The initial learning rate is set to 0.0001 and reduced by half if the validation $\mathcal{L}_{MTL}$ is not dropping in 5 epochs. Then, the train-val process is stopped if there is no drop on the validation $\mathcal{L}_{MTL}$ in 30 epochs. The learning curve is discussed in Appendix A.3.

TABLE 6.2: Model Specification

Model	Variant	Total Parameters↓	Input/Sensor	Output
TransFuser [80] [82]	Late Fusion	32.64M	LiDAR, RGB, GNSS, IMU, Rotary	Waypoints, Steering, Throttle
	Transformer	66.23M	LiDAR, RGB, GNSS, IMU, Rotary	Waypoints, Steering, Throttle
DeepIPCv2	Log. Depth	5.91M	LiDAR, GNSS, IMU, Rotary	Waypoints, Steering, Throttle
	Segmentation	5.95M +14M	LiDAR, GNSS, IMU, Rotary	Segmentation, Waypoints, Steering, Throttle
	Segmentation + Log. Depth	5.96M +14M	LiDAR, GNSS, IMU, Rotary	Segmentation, Waypoints, Steering, Throttle

Some DeepIPCv2 variants employ PolarNet [111] which has total parameters of around 14 million to perform point cloud segmentation. We replicate TransFuser [80] [82] based on the codes shared by the authors at https://github.com/autonomousvision/transfuser. We cannot compute the inference speed fairly due to fluctuating GPU computation, hence we assume that smaller models will consume less GPU memory footprint and infer faster. Furthermore, as we limit the maximum speed to only 1.25m/s (the same as the data collection process), a high FPS rate is not necessary for driving the robotic vehicle.

6.3 Experiment and Analysis

In this section, we explain the functions used to justify the model's performance. For ablation and comparative studies between DeepIPCv2, TransFuser, and their variants, first, we conduct an offline test by deploying all model variants to predict several driving records. Then, we take the best variant of each model for the online test by deploying them for real automated driving in real environments.

6.3.1 Evaluation and Scoring

The evaluation is conducted under three different conditions (noon, evening, and night). We consider two different kinds of evaluations namely offline and online tests. In the offline test, DeepIPCv2 is deployed to predict a set of expert driving records on the test routes. The performance is defined by the total metric (TM) score as in (6.8).

$$TM = MAE_{WP} + MAE_{ST} + MAE_{TH} \tag{6.8}$$

where MAE stands for mean absolute error (L1 loss) which can be computed with (6.6) for MAE_{WP} and (6.7) for MAE_{ST} and MAE_{TH}. The smaller the total metric score means the better the model performance. Meanwhile, in the online test, DeepIPCv2 must drive the vehicle properly following a set of route points in six different routes. We determine the drivability performance by counting the number of interventions and intervention time needed to prevent any collisions. The smaller the number of interventions and intervention time means the better the performance. To be noted, the final score for both tests must be averaged as the evaluation is conducted three times for each condition.

As mentioned in Subsection 6.2.1, we create two model variants for the ablation study. The first variant only takes the logarithmic depth point clouds while the second variant only takes the segmented point clouds. Furthermore, we also conduct a comparative study by replicating TransFuser [80] [82] to compare with. Briefly, TransFuser is a camera-LiDAR fusion model that takes an RGB image and a set of point clouds. It perceives the environment from two different perspectives where the front view information is given by the RGB camera and the BEV information is given by the LiDAR. TransFuser fuses RGB and LiDAR features using several transformer modules. We also replicate its variant called late fusion, where the features are fused with a simple element-wise summation. The specification of DeepIPCv2 and TransFuser variants can be seen in Table 6.2.

TABLE 6.3: Multi-task Performance Score 1

Condition	Model	Variant	Total Metric↓	MAE_{WP} ↓	MAE_{ST} ↓	MAE_{TH} ↓
Noon	TransFuser [80] [82]	Late Fusion	0.211 ± 0.007	0.087	0.097	0.027
		Transformer	0.192 ± 0.006	0.073	0.093	0.026
	DeepIPCv2	Log. Depth	0.276 ± 0.004	0.116	0.123	0.037
		Segmentation	$\mathbf{0.168 \pm 0.005}$	**0.059**	**0.085**	**0.024**
		Segmentation + Log. Depth	0.196 ± 0.007	0.074	0.095	0.026
Evening	TransFuser [80] [82]	Late Fusion	0.213 ± 0.006	0.089	0.097	0.027
		Transformer	0.193 ± 0.008	0.073	0.094	0.026
	DeepIPCv2	Log. Depth	0.281 ± 0.007	0.119	0.126	0.036
		Segmentation	$\mathbf{0.167 \pm 0.006}$	**0.059**	**0.084**	**0.023**
		Segmentation + Log. Depth	0.199 ± 0.008	0.076	0.097	0.026
Night	TransFuser [80] [82]	Late Fusion	0.218 ± 0.002	0.090	0.099	0.029
		Transformer	0.197 ± 0.003	0.075	0.094	0.028
	DeepIPCv2	Log. Depth	0.278 ± 0.005	0.115	0.125	0.038
		Segmentation	$\mathbf{0.170 \pm 0.002}$	**0.059**	**0.086**	**0.026**
		Segmentation + Log. Depth	0.198 ± 0.004	0.072	0.097	0.028

6.3.2 Offline Test

An offline test is used to measure how good the model is in mimicking an expert manipulation in controlling vehicle actuators. The test is conducted by letting the model predict several driving records made for testing purposes. We measure the model performance by calculating the MAE on waypoints prediction and navigational control estimation together with the total metric (TM) score as explained in Subsection 6.3.1. Since there are three driving records for each condition, the final score is averaged from all inference results. Be noted that each driving record is taken on different days to vary the situation and the cloud intensity.

Based on Table 6.3, the DeepIPCv2 variant that only takes segmented point clouds achieves the best performance by having the lowest TM score in all conditions. The other two DeepIPCv2 variants that take logarithmic depth point clouds fall behind and the depth-only variant performs the worst. This pattern shows that processing logarithmic depth reduces overall model performance due to conflicting features between the segmentation map and the depth map extracted by the encoders. This also means that the data representation given by the projected point clouds in the segmentation map is more than enough and better than the combination with logarithmic depth. However, this hypothesis needs to be justified further by applying different encoder architectures to process the projected point clouds. Meanwhile, amongst TransFuser variants, the variant that employs transformer modules to fuse image and point cloud features achieves a better performance than the variant that only uses a simple element-wise summation. Although it costs a lot of parameters to train, the transformer modules can improve the model's reasoning as it understands the relationship between the front view and BEV perspectives.

Across different conditions, the total metric scores for TransFuser variants consistently become higher from noon to night. Although the gap is not too far from one another, this pattern shows that TransFuser which relies on RGB images gets a performance drop when the illumination condition is poor. This result is as expected since the RGB camera is sensitive to illumination changes. Unlike TransFuser, DeepIPCv2 is more robust against poor illumination conditions as it only relies on LiDAR to perceive the environment. However, there is no clear pattern amongst DeepIPCv2 variants as their performance differs on every condition. DeepIPCv2 performance is more affected by road situations rather than illumination conditions. This is supported by the fact that two DeepIPCv2 variants have the best performance at night when there is not so much traffic on the road.

TABLE 6.4: Multi-task Performance Score 2

Condition	Perspective	Total Metric↓	$MAE_{WP}\downarrow$	$MAE_{ST}\downarrow$	$MAE_{TH}\downarrow$
Noon	Front	0.258 ±0.006	0.062	0.173	**0.023**
	BEV	0.171 ±0.006	0.063	**0.084**	0.024
	Front + BEV	**0.168 ±0.005**	**0.059**	0.085	0.024
Evening	Front	0.258 ±0.014	0.063	0.173	**0.022**
	BEV	0.171 ±0.005	0.062	0.085	0.023
	Front + BEV	**0.167 ±0.006**	**0.059**	**0.084**	0.023
Night	Front	0.263 ±0.005	0.061	0.177	**0.025**
	BEV	0.174 ±0.004	0.062	**0.086**	0.026
	Front + BEV	**0.170 ±0.002**	**0.059**	**0.086**	0.026

To understand the importance of perceiving from multiple perspectives of view, we also conduct an extensive ablation study by creating two more DeepIPCv2 variants that take one perspective of view. Hence, the model can only perceive the environment based on the front-view perspective or the top-view/bird's eye view (BEV) perspective. We develop these variants based on DeepIPCv2 variant that takes the point cloud segmentation map. Table 6.4 shows that the model variant that perceives the environment from both front and BEV perspectives has the lowest total metric score meaning that it achieves the best performance compared to the variants which only use one perspective. This result is in line with the findings in our previous work [56] [55] and strengthens the importance of perceiving from multiple perspectives of view.

To be more detailed, DeepIPCv2 variant that perceives from the front-view perspective has the worst performance as its steering estimation is heavily affected by the absence of BEV perception features. Without these features, the controller module faces difficulties in estimating the steering angle which lies in the BEV coordinate. Meanwhile, DeepIPCv2 variant that perceives from the BEV perspective is slightly behind the best variant. Although it has a comparable performance in steering angle and throttle level estimation, this variant fails to estimate waypoints properly as predicting future vehicle position needs the combination of both front and BEV perception features that consider more aspects of driving. Therefore, judging from this result and analysis, we pick the DeepIPCv2 variant that only takes segmented point clouds and perceives the environment from multiple perspectives of view for further comparison in the online test. Meanwhile, we pick TransFuser variant that employs transformer modules as the comparator to study the importance of data modality and representation in performing real-world autonomous driving.

6.3.3 Online Test

An online test is made for evaluating the drivability performance of the model after imitating expert behavior in driving a vehicle during the training process. In this evaluation, we use the best variant of DeepIPCv2 and TransFuser to perform automated driving in real environments. Each model variant must be able to handle various situations and conditions when driving a vehicle from the starting point to the finish point by following a set of route points. Similar to the offline test, we conduct the evaluation three times on six different routes for each condition. However, the

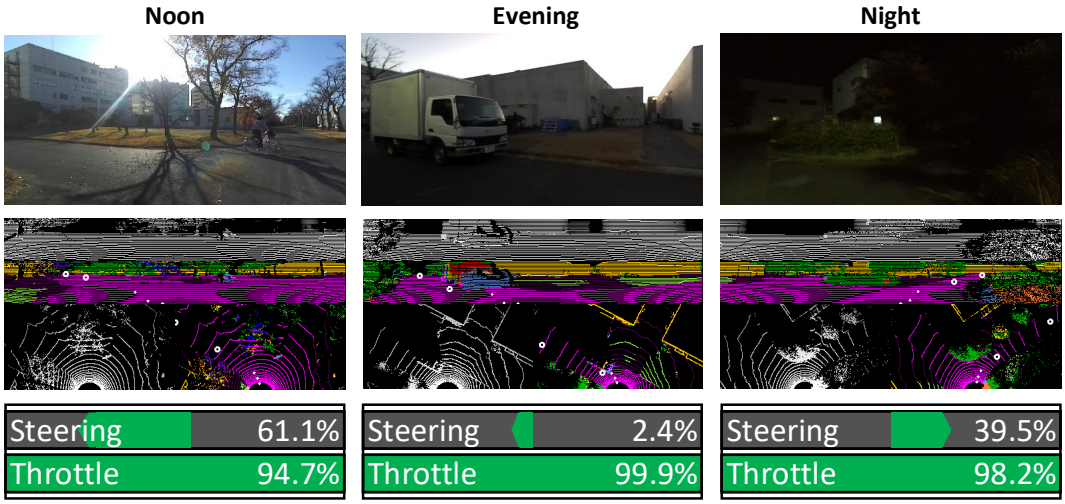


FIGURE 6.6: Driving footage.

Noon: DeepIPCv2 makes a left turn following two route points located on the left side of the vehicle. Evening: A moment when we are going to intervene in DeepIPCv2 as it fails to make a hard left turn to compromise a narrowing path caused by a stopping truck. Night: DeepIPCv2 makes a right turn following two route points while maintaining the distance from the road boundaries. To be noted, the RGB images are only for record purposes since DeepIPCv2 uses LiDAR point clouds to perceive the environment. We share some of the driving records at https://youtu.be/IsZ1HP5QjWc which include the driving records of TransFuser [80] [82] for comparison.

TABLE 6.5: Drivability Score

Condition	Model	Intervention↓ Count	Intervention↓ Time (secs)
Noon	TransFuser [80] [82]	1.389 ± 0.208	3.537 ± 0.648
	DeepIPCv2	**1.000 ± 0.236**	**2.389 ± 0.831**
Evening	TransFuser [80] [82]	1.222 ± 0.079	3.093 ± 0.457
	DeepIPCv2	**0.944 ± 0.157**	**2.407 ± 0.466**
Night	TransFuser [80] [82]	1.889 ± 0.283	4.556 ± 0.181
	DeepIPCv2	**0.667 ± 0.136**	**1.870 ± 0.340**

drivability performance is justified by the number of interventions and how long the interventions are. Then, the best performance is determined by the lowest number and the shortest time of interventions. Keep in mind that the result of the online test may not be in line with the result of the offline test. This is because any decisions made on every observation state in the online test will affect the next observation state. Meanwhile, in the offline test, although the model makes a wrong prediction, it will not affect anything as the observation is already fixed in the driving records. Furthermore, we also provide some driving footage that can be seen in Fig. 6.6.

Table 6.5 shows a clear pattern for each model when driving under different conditions. For DeepIPCv2 which is not affected by illumination conditions, the best drivability is achieved at night when there is not so much traffic on the road. Then, it has lower performance at noon and in the evening as it is affected by denser traffic on the road. Meanwhile, as for TransFuser which relies on the RGB camera, better

performance is achieved when the model drives at noon and in the evening. Thanks to enough light illumination, the RGB camera can capture an image clearly so that TransFuser can maintain its drivability. However, TransFuser performance is degraded when driving at night as it fails to capture the information in front of the vehicle due to poor illumination conditions. Therefore, as the perception module cannot extract useful features, the controller module also fails to estimate navigational control properly.

In all conditions, DeepIPCv2 has the best performance based on the lowest intervention count and intervention time compared to TransFuser. This means that a set of segmented point clouds projected into two different perspectives of view contains more valuable information than a combination of a raw RGB image and a 2-bin point cloud histogram. By projecting the segmented point clouds to form image-like arrays that contain a unique class on each layer, the model has a better scene understanding capability as it can distinguish traversable and non-traversable areas clearly and lead to better driving performance. This also shows that data representation (e.g. segmented and projected point clouds) is matter and more meaningful than a combination of some data modalities (e.g. RGB images and LiDAR point clouds) but still in their raw form or improperly pre-processed. Aside from the perception parts, DeepIPCv2 also has a better controller module that gives a higher degree of maneuverability.

6.4 Findings

In this Chapter, we propose DeepIPCv2 which perceives the environment using LiDAR for more robust drivability. DeepIPCv2 is evaluated by predicting driving records and performing automated driving. To justify its performance, we conduct ablation and comparative studies with other models under different conditions to vary the situations. Based on the experimental results, we disclose several findings as follows.

- Using LiDAR to perceive the environment increases the model's robustness. Unlike an RGB camera, LiDAR is not affected by poor illumination conditions. Thus, the perception module can provide stable features to the controller module in estimating navigational control properly. Therefore, the model can maintain its drivability even when driving at night. Meanwhile, the performance of a camera-powered model drops as it fails to perceive the surrounding area.

- Perceiving the environment with segmented point clouds is better than logarithmic depth and 2-bin histogram point clouds. This is because the model can distinguish traversable and non-traversable areas easily. However, the selection of the point cloud segmentation model must be carried out carefully considering the trade-off between computational load, latency, and performance for achieving a real-time inference.

Chapter 7

Summary

In this study, we develop a novel model called DeepIPC (Deeply Integrated Perception and Control) for end-to-end autonomous driving. This model can handle multiple perception and control tasks simultaneously in one forward pass. We conduct the study gradually from the perception-only to the integrated perception-action, from inference on driving records to deployment for automated driving, and from simulation-based experiments to real-world experiments. The study begins with the development of a loss weighting algorithm namely Modified Gradient Normalization (MGN), which is used to balance the learning signal for a multi-task model during the training process. Then, focusing on the point-to-point navigation task, DeepIPC is employed to drive a vehicle safely in dynamic simulated environments with various conditions and scenarios. Finally, as a proof-of-concept study, we also conduct real-world demonstrations using a modified DeepIPC and DeepIPCv2, two models that are improved specifically to deal with implementation issues and deployed to drive a robotic vehicle in real environments. Our proposed models achieve better performance in many criteria compared to other models.

7.1 Conclusion

Based on the experimental findings disclosed in the previous chapters, we draw several conclusions that are summarized as follows.

- The MGN algorithm successfully balances the learning signal for a multi-task model, improving its performance. Based on the results discussed in Chapter 3, a model trained with the MGN algorithm performs better than a model trained normally. This shows that the MGN algorithm can manage the trade-off between multiple learning signals to ensure all tasks can be learned at the same pace. Furthermore, as the multi-task model outperforms a combination of single-task models, this also strengthens plenty of studies that leverage the usefulness of the feature-sharing mechanism in a multi-task architecture.

- Perceiving the environment from different perspectives and employing multi-agent to make decisions can improve the performance of an autonomous driving model. Based on the comparative and ablation studies discussed in Chapter 4, our proposed model (DeepIPC) that performs semantic depth cloud mapping and employs two agents achieves the best driving performance in a simulated environment. This shows that having better scene understanding and decision-making can boost drivability. However, the performance gets decreased as dynamic weather and adversarial scenarios put more challenges to the model's adaptability and maneuverability.

- Imitation learning is also useful for real-world experiments. Based on the experimental results discussed in Chapter 5, some real implementation problems such as sensor inaccuracies and noise can be solved with a strong behavior cloning from a set of expert's driving records. All models including DeepIPC that are based on simulation can be brought to drive the vehicle seamlessly in real environments by mimicking an expert driver. This also exposes the effectiveness of the end-to-end behavior cloning technique for a complex multi-input multi-output model. Furthermore, the best performance is achieved by DeepIPC, as it has a better architecture compared to the other models. However, its performance decreased when driving under low-light conditions as the RGBD camera fails to provide stable information.
- Using LiDAR to perceive the environment can achieve higher robustness in scene understanding and lead to better drivability. As discussed in Chapter 6, DeepIPCv2 which only uses LiDAR achieves better performance compared to a camera-LiDAR fusion model, especially when driving at night. This is because a LiDAR sensor is not affected by poor illumination conditions as it has its own lasers to sense the environment. Moreover, since DeepIPCv2 is also supported with a point cloud segmentation model, it can distinguish different objects around the vehicle with ease.

7.2 Future Work

In the future, we consider developing a more advanced model that will be trained with expert driving data recorded using a real car as described in Appendix B. As the environment including traffic conditions becomes more complex and challenging, an end-to-end model needs to be improved further in many aspects to achieve a highly reliable navigation system for autonomous driving vehicles.

7.2.1 Future Research Direction

The following is the list of several key ideas that can be used as future research directions in enhancing the model's drivability to meet the standard or even a higher degree of automation for deployment on a real car.

More Sensors with Better Fusion Technique

Perception holds an important role to achieve excellent drivability as understanding the surrounding area before making any actions is a must. Be noted that with better perception, the model is expected to have better drivability as the controller is provided with more useful features for performing navigational controls. One way to improve the perception part is by adding more sensors to collect more data that contain rich information. For instance, we can use multiple cameras to support the LiDAR sensor to cover a complete 360^{o} view. In addition, we can also use an event camera or dynamic vision sensor to detect local changes in brightness. This can be helpful to recognize any objects that move relative to the vehicle. However, using multiple sensors comes with plenty of different data modalities to handle. Therefore, a better sensor fusion strategy is needed to process the data. To settle this issue, we plan to adopt TransFuser [80] [82] and modify some of its parts to strengthen our future model.

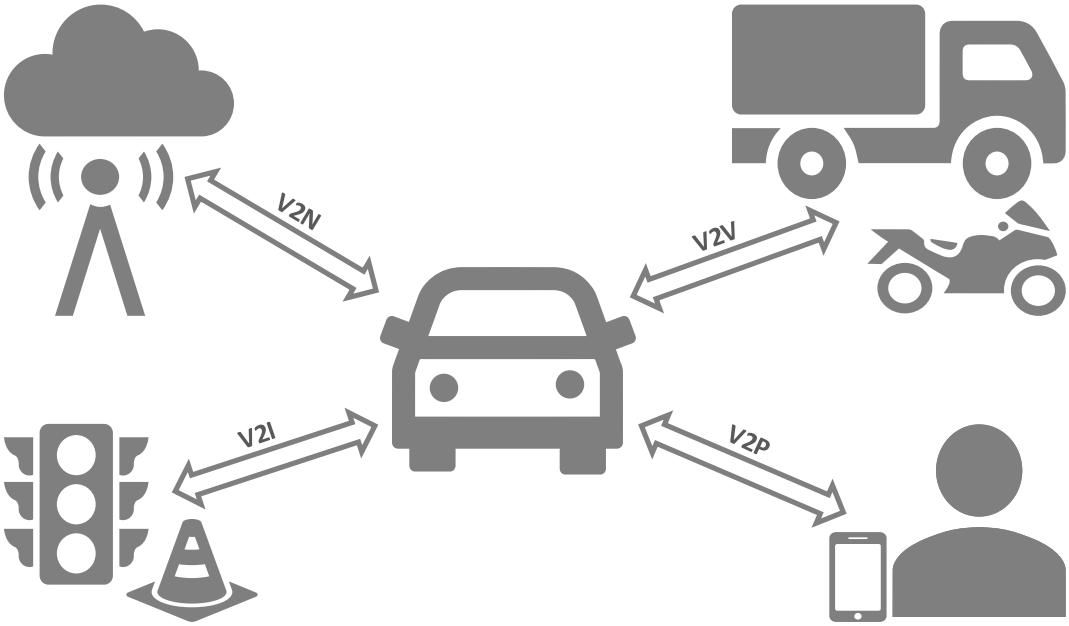


FIGURE 7.1: The illustration of vehicle-to-everything communication (V2X).

Vehicle-to-everything communication (V2X) incorporates many kinds of communication systems such as vehicle-to-network (V2N), vehicle-to-vehicle (V2V), vehicle-to-pedestrian (V2P), vehicle-to-infrastructure (V2I).

Better Reasoning for A Higher Degree of Understanding

Perception should not be done only by performing multiple vision tasks to distinguish different objects, estimate their distance, etc. Beyond that, we need a better method to examine which object in a certain circumstance gives influence the model in making decisions. This is because not all objects that appear in front of the vehicle are always important. By selecting certain objects considered for the decision-making process, we believe it can increase the reasoning capability of the model, which lead to a higher degree of scene understanding and result in better driving performance. To achieve better reasoning for our future model, we plan to add object-level attention modules [187] [188] to allow the model to examine important objects all by itself. Therefore, the learned policy and any decisions made by the model are explainable.

Vehicle-to-Everything (V2X)

V2X communication can be an interesting future research direction to work on. Besides focusing on the performance of the model that drives the vehicle itself, communication is also an important factor to achieve reliable autonomous driving [189]. A smart vehicle needs to coordinate with any entities that may affect or may be affected by the vehicle such as other vehicles, networks, infrastructure, and pedestrians as illustrated in Fig. 7.1. Coordination between these entities can lead to a well-organized intelligent transportation system that incorporates many factors in the environment. One interesting work to begin with is decentralized V2X (D-V2X) using a blockchain approach that can be implemented on top of any communication protocol and does not require any trusted authority [190].

Imitation Learning and Reinforcement Learning

Based on the experimental results discussed in Chapter 5 and Chapter 6, imitation learning has shown its usefulness for training an end-to-end model to drive a robotic vehicle in real environments. However, there is always a possibility that the model may fail under a certain adversarial scenario (e.g., pedestrians crossing the street suddenly) that is not covered in the dataset. To explore plenty of unexpected scenarios, reinforcement learning can be used to train the model. However, this technique requires huge amounts of risky data that is unsuitable and irrelevant to real-world autonomous driving, especially with a real car. To solve this problem, we plan to adopt training algorithms namely Controllable Imitative Reinforcement Learning (CIRL) [191] and/or Deep Imitative Reinforcement Learning (DIRL) [192] that combines imitation learning and reinforcement learning. Concisely, these algorithms let a model train with imitation learning first using a lot of driving records including publicly available datasets to enrich its driving experience. After having a prior knowledge of driving a vehicle, the model is refined with reinforcement learning where any interventions the driver makes are used as the learning signals to train the model.

Appendix A

Learning Curve and Task Balancing Behavior

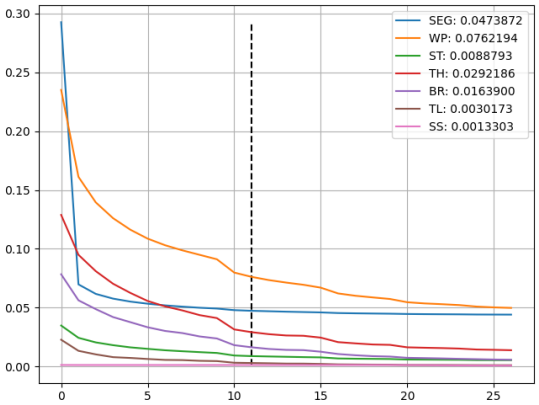
In this appendix, we provide the training log that shows the learning curve of our models and the behavior of the modified gradient normalization (MGN) in tuning the loss weights for balancing the learning process.

A.1 DeepIPC Training with Simulation Dataset

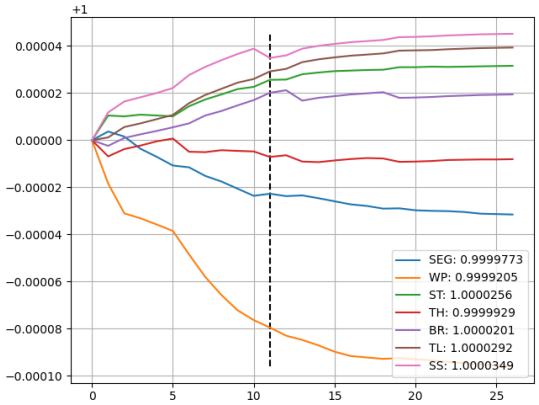
Fig. A.1 shows the learning curve of DeepIPC during the training process on CARLA simulation datasets, 1W (one weather, clear noon) and AW (all weather). At the time when the model convergence, the modified gradient normalization (MGN) algorithm gives the lowest loss weight to the waypoints prediction task (WP) and the highest loss weight to the stop sign prediction task (SS). Between these tasks, the order from the second lowest loss weight to the second highest loss weight are as follows: image segmentation task (SEG), navigational controls estimation task (throttle (TH), brake (BR), and steering (ST)), and traffic light prediction task (TL). This result is in line with the training loss trends where the waypoints prediction loss is the highest and followed by the other tasks in the same order. On dataset AW, only throttle estimation and image segmentation loss weights are placed differently. However, they are swapped in the next epochs following the training loss trends. These records strengthen the conclusion disclosed in Chapter 3 where the MGN algorithm tends to give more loss weights to the tasks with smaller losses to prevent the model from losing its focus.

A.2 DeepIPC Training with Real-world Dataset

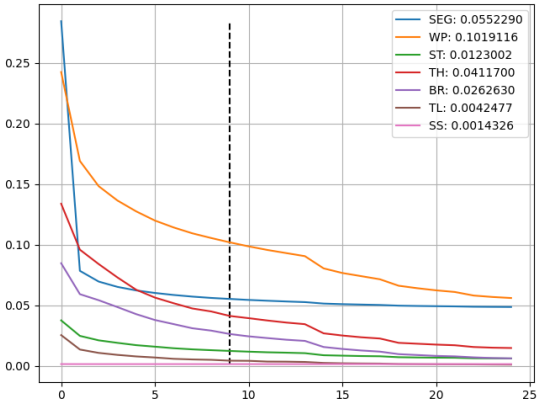
Fig. A.2 shows the learning curve of DeepIPC during the training process on real-world datasets. At the time when the model convergence, the modified gradient normalization (MGN) algorithm gives the highest loss weight to the throttle estimation task (TH) followed by the waypoints prediction task (WP), the image segmentation task (SEG), and the steering estimation task (ST) respectively. The loss weighting behavior is in line with the training loss trends where the throttle estimation loss is the smallest followed by the waypoints prediction loss. This result strengthens our findings stated in Appendix A.1. However, the loss weights for steering estimation and image segmentation tasks are not in line due to the difficulty of estimating the steering level with fewer data. This is because the number of frames when the vehicle turns is less than when the vehicle goes straight.



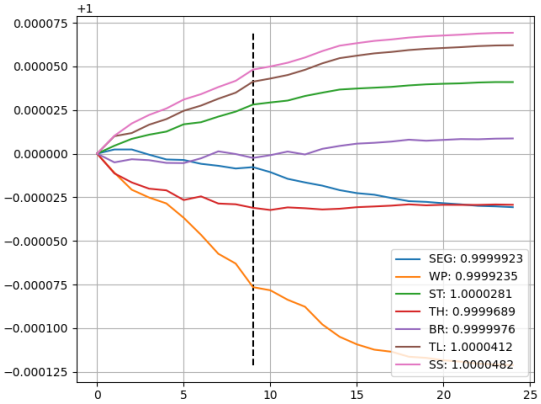
(A) Training loss on dataset 1W



(B) Loss weights update on dataset 1W



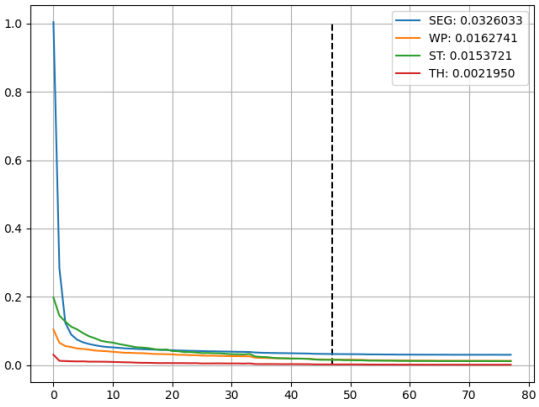
(C) Training loss on dataset AW



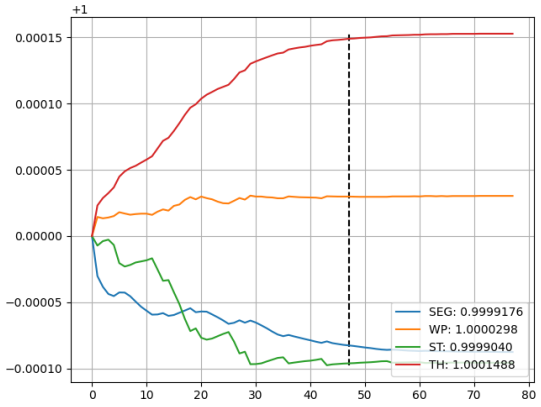
(D) Loss weights update on dataset AW

FIGURE A.1: DeepIPC training log on simulation dataset.

The vertical black dashed line shows the exact epoch where the model convergence. The vertical axis on each figure is the training loss and the loss weight while the horizontal axis is the epoch.



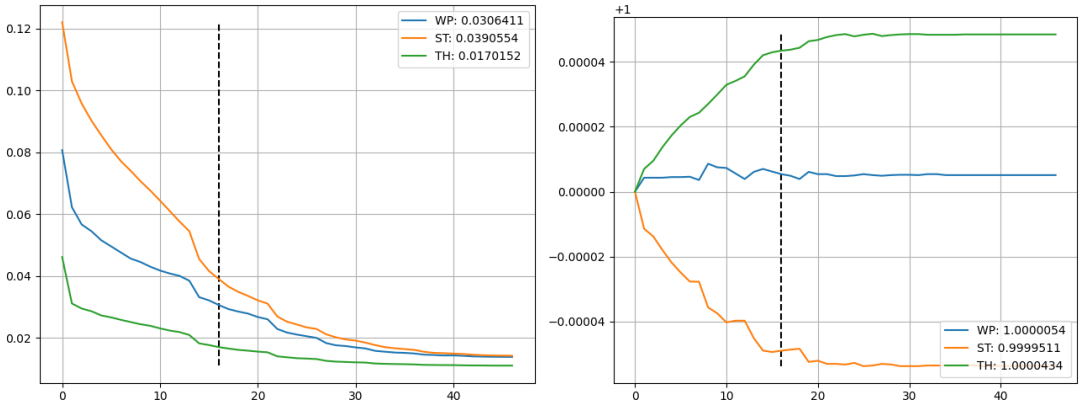
(A) Training loss



(B) Loss weights update

FIGURE A.2: DeepIPC training log on real-world dataset.

The vertical black dashed line shows the exact epoch where the model convergence. The vertical axis on each figure is the training loss and the loss weight while the horizontal axis is the epoch.



(A) Training loss (B) Loss weights update

FIGURE A.3: DeepIPCv2 training log on real-world dataset.

The vertical black dashed line shows the exact epoch where the model convergence. The vertical axis on each figure is the training loss and the loss weight while the horizontal axis is the epoch.

A.3 DeepIPCv2 Training with Real-world Dataset

Fig. A.3 shows the learning curve of DeepIPCv2 during the training process on real-world datasets. At the time when the model convergence, the modified gradient normalization (MGN) algorithm gives the highest loss weight to the throttle estimation task (TH) followed by the waypoints prediction task (WP) and the steering estimation task (ST) respectively. The loss weighting behavior is in line with the training loss trends where the throttle estimation loss is the smallest followed by the waypoints prediction loss and the steering estimation loss. This result also strengthens our findings stated in Appendix A.1 Appendix A.2 where the MGN algorithm tends to give a higher loss weight to the tasks with smaller losses to keep the model focus.

Appendix B

Preliminary Experiments with a Real Car

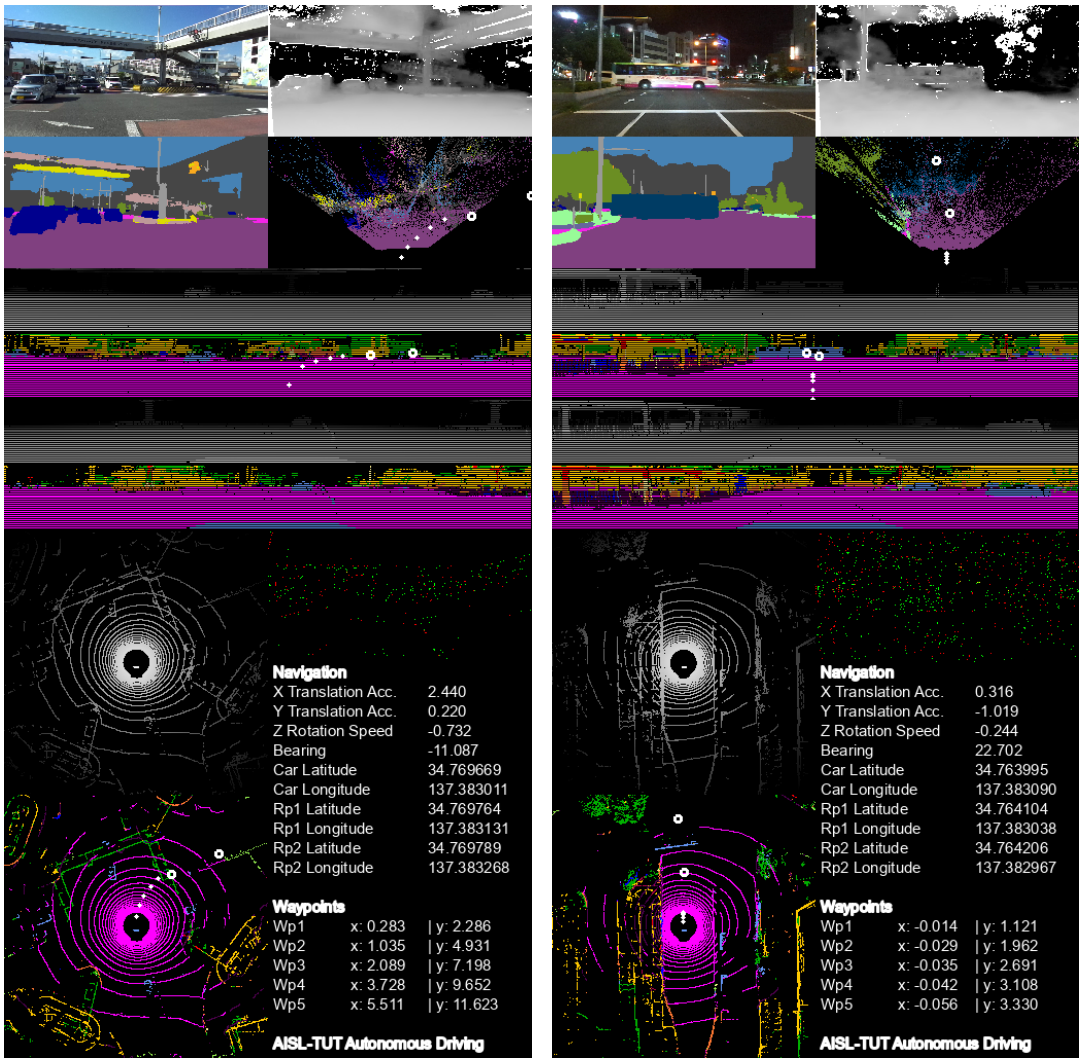
(A) Car (ego vehicle) (B) Sensors (C) RTK base station

FIGURE B.1: Devices and sensors placement.

In this appendix, we provide a detail of our preliminary experiments with a real car. To be more specific, we briefly explain the devices used to retrieve the data and the plan to improve our model to achieve better drivability.

B.1 Sensors and Observation Data

The sensors used for the experiments are the same as those installed on the robotic vehicle mentioned in Chapter 5 and Chapter 6, with the addition of a dynamic vision sensor (DVS) camera used to detect event changes at the front of the car. The DVS camera is expected to give the model a better scene understanding as it can capture moving objects faster than any conventional camera [193]. Furthermore, we use a powerful edge device for data acquisition and ensure the data can be retrieved synchronously. As shown in Fig. B.1, we mount all sensors on a metal plate and place them on the top of the car. Additionally, we also make our own base station for real-time kinematic (RTK) positioning to enable the mounted GNSS receiver to measure the car's relative position in real-time with better accuracy [194]. With this



(A) Daytime (B) Nighttime

FIGURE B.2: Sets of driving records.

kind of setup, we can retrieve a set of records composed of RGBD images, DVS images, LiDAR point clouds, GNSS locations, and 9-axis IMU data as shown in Fig. B.2. Meanwhile, the segmentation maps for RGB images and LiDAR point clouds are provided by SegFormer [176] and PolarNet [111] respectively. In the future, a more advanced sensor such as hemispherical LiDAR can be used to deal with blind spots. Then, the GNSS receiver can be enhanced with an inertial navigation system (INS) to solve the blocked signal issues when driving in a tunnel or near high buildings.

B.2 Addressing New Challenges

Applying our methods for autonomous driving with a real car comes with plenty of new challenges. Aside from enhancing the model architecture itself, we also need to think about how to eliminate some issues in the data-gathering process that will be used for training the model. For instance, in generating the pseudo-labels for the semantic segmentation task, we cannot rely on SegFormer [176] when dealing with nighttime driving records. Thus, we apply domain adaptation techniques called Refign [195] and HRDA [196] to enhance SegFormer. Hence, it can be used to provide

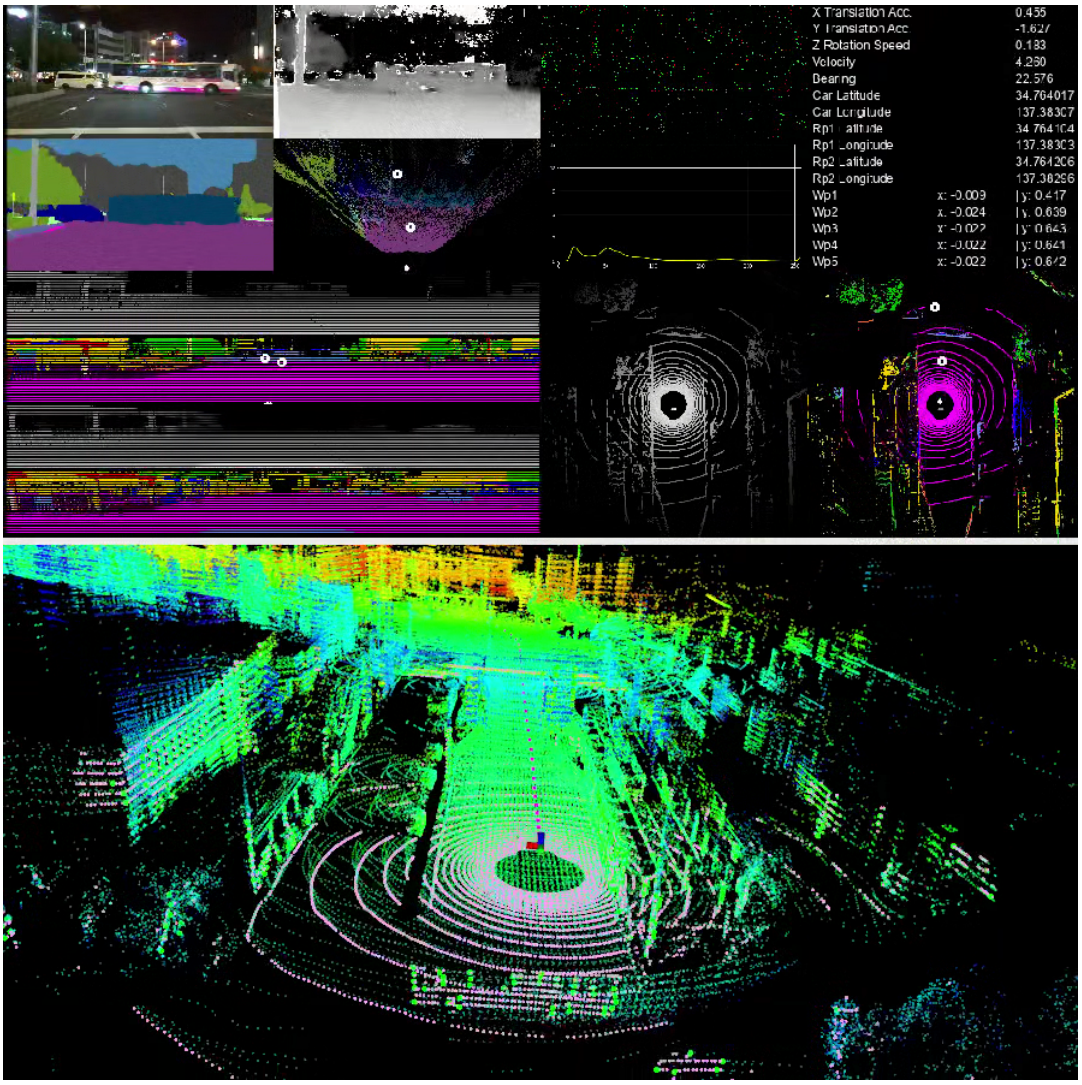


FIGURE B.3: The utilization of LeGO-LOAM to estimate trajectory.

proper segmentation maps based on the adaptation from the Cityscapes dataset [68] to the ACDC dataset [197] that contains nighttime driving data.

Another challenge comes from the IMU-based trajectory estimation algorithm used for generating the waypoints ground truth. The algorithm tends to drift quickly when the vehicle moves faster. Meanwhile, we cannot rely on visual-inertial odometry as it fails at night when everything is not clearly visible. Therefore, we use LeGO-LOAM [198] (lightweight and ground-optimized LiDAR odometry and mapping [199]) to estimate the trajectory based on point clouds that are not affected by poor illumination conditions. A set of driving records with LeGO-LOAM used to estimate the vehicle's trajectory can be seen in Fig. B.3.

B.3 Transformer-powered Model

Transformer has been widely used in many fields of research, including computer vision [200] [201]. This kind of deep learning model is known for its powerful self-attention mechanism, which allows the network to capture and learn contextual relationships in the data [202]. Hence, to better process multiple data simultaneously,

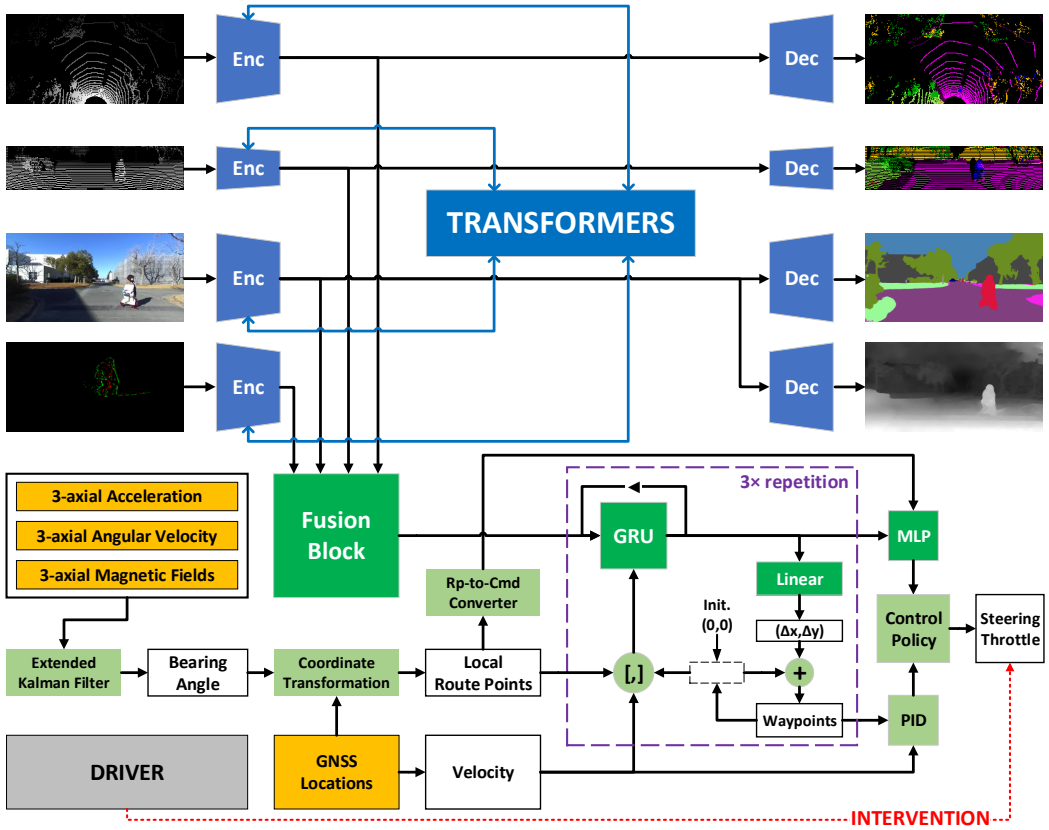


FIGURE B.4: DeepIPC architecture with transformers.

The perception module of DeepIPC employs a set of transformers to learn the relationship between features extracted by a set of CNN-based encoders.

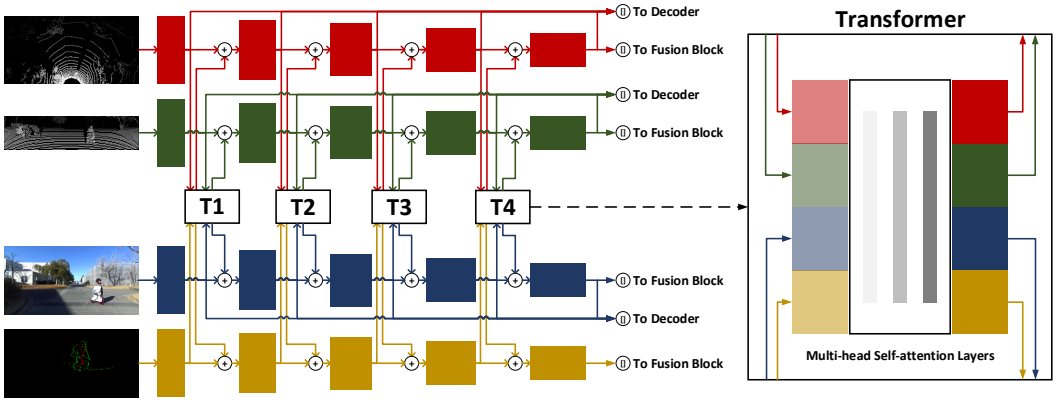


FIGURE B.5: Encoder blocks and transformers.

Using its unique self-attention mechanism, each transformer is responsible for learning the relationship between features extracted from each encoder's convolution block.

we plan to enhance our model with a set of transformer modules to learn the relationship between extracted features. Thus, the model will have better reasoning and can provide more useful features for the controller module in making action decisions. In brief, our future model can be illustrated in Fig. B.4 and Fig. B.5.

Bibliography

[1] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Dec. 2015.

[2] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, Jun. 2011, pp. 163–168.

[3] E. Khatab, A. Onsy, M. Varley, and A. Abouelfarag, "Vulnerable objects detection for autonomous driving: A review," *Integration*, vol. 78, pp. 36–48, May 2021.

[4] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, Mar. 2021.

[5] Q. Liu, X. Li, S. Yuan, and Z. Li, "Decision-making technology for autonomous vehicles: Learning-based methods, applications and future outlook," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, USA, Sep. 2021, pp. 30–37.

[6] 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, Oct. 2020.

[7] F. Kunz, D. Nuss, J. Wiest, H. Deusch, S. Reuter, F. Gritschneder, A. Scheel, M. Stübler, M. Bach, P. Hatzelmann, C. Wild, and K. Dietmayer, "Autonomous driving at Ulm University: A modular, robust, and sensor-independent fusion approach," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Seoul, South Korea, Jun. 2015, pp. 666–673.

[8] J. Tang, L. Shaoshan, S. Pei, S. Zuckerman, L. Chen, W. Shi, and J.-L. Gaudiot, "Teaching autonomous driving using a modular and integrated approach," in *Proceedings of the IEEE Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, Jul. 2018, pp. 361–366.

[9] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Toward autonomous driving: The CMU navlab. part I: Perception," *IEEE Expert*, vol. 6, no. 4, pp. 31–42, Aug. 1991.

[10] C. Thorpe, M. Herbert, T. Kanade, and S. Shafter, "Toward autonomous driving: The CMU navlab. II. architecture and systems," *IEEE Expert*, vol. 6, no. 4, pp. 44–52, 1991.

[11] C. Thorpe, M. H. Hebert, T. Kanade, and S. A. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, p. 362–372, May 1988.

[12] D. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Denver, USA, Jan. 1988, pp. 305 – 313.

[13] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly, "The DARPA grand challenge - development of an autonomous vehicle," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Parma, Italy, Jun. 2004, pp. 226–231.

[14] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *Proceedings of the American Control Conference (ACC)*, Seattle, USA, May 2017, pp. 4914–4919.

[15] D. Coelho and M. Oliveira, "A review of end-to-end autonomous driving in urban environments," *IEEE Access*, vol. 10, pp. 75 296–75 311, Jul. 2022.

[16] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues," *Array*, vol. 10, p. 100057, Jul. 2021.

[17] J. Horgan, C. Hughes, J. McDonald, and S. Yogamani, "Vision-based driver assistance systems: Survey, taxonomy and advances," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Gran Canaria, Spain, Sep. 2015, pp. 2032–2039.

[18] G. Adam, V. Chitalia, N. Simha, A. Ismail, S. Kulkarni, V. Narayan, and M. Schulze, "Robustness and deployability of deep object detectors in autonomous driving," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, Oct. 2019, pp. 4128–4133.

[19] C. Wang and N. Aouf, "Fusion attention network for autonomous cars semantic segmentation," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Aachen, Germany, Jul. 2022, pp. 1525–1530.

[20] A. Gurram, A. F. Tuna, F. Shen, O. Urfalioglu, and A. M. López, "Monocular depth estimation through virtual-world supervision and real-world SfM self-supervision," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 12 738–12 751, Aug. 2022.

[21] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, "Multi-task learning for dense prediction tasks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3614–3633, Jul. 2022.

[22] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3523–3542, Jul. 2022.

[23] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, Dec. 2022.

[24] T.-J. Song, J. Jeong, and J.-H. Kim, "End-to-end real-time obstacle detection network for safe self-driving via multi-task learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16 318–16 329, Sep. 2022.

[25] E. Marti, M. A. de Miguel, F. Garcia, and J. Perez, "A review of sensor technologies for perception in automated driving," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 4, pp. 94–108, Sep. 2019.

[26] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, Mar. 2020.

[27] M. Berk, O. Schubert, H.-M. Kroll, B. Buschardt, and D. Straub, "Exploiting redundancy for reliability analysis of sensor perception in automated driving vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 12, pp. 5073–5085, Dec. 2020.

[28] E. E. Aksoy, S. Baci, and S. Cavdar, "SalsaNet: Fast road and vehicle segmentation in LiDAR point clouds for autonomous driving," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, USA, Oct. 2020, pp. 926–932.

[29] B. Shakibajahromi, A. Jabalameli, A. S. Krishnan, S. Kanzler, and S. Shayestehmanesh, "Instantaneous velocity estimation for 360° perception with multiple high quality radars: An experimental validation study," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, USA, Oct. 2020, pp. 789–794.

[30] Y. Hu, J. Binas, D. Neil, S.-C. Liu, and T. Delbruck, "DDD20 end-to-end event camera driving dataset: Fusing frames and events with deep learning for improved steering prediction," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Rhodes, Greece, Sep. 2020, pp. 1–6.

[31] F. Munir, S. Azam, M. Jeon, B.-G. Lee, and W. Pedrycz, "LDNet: End-to-end lane marking detection approach using a dynamic vision sensor," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9318–9334, Jul. 2022.

[32] A. Yousefzadeh, G. Orchard, T. S. Gotarredona, and B. L. Barranco, "Active perception with dynamic vision sensors. minimum saccades with optimum recognition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 4, pp. 927–939, Aug. 2018.

[33] L. Sun, K. Yang, X. Hu, W. Hu, and K. Wang, "Real-time fusion network for RGB-D semantic segmentation incorporating unexpected obstacle detection for road-driving images," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5558–5565, Oct. 2020.

[34] J. Fayyad, M. A. Jaradat, D. Gruyer, and H. Najjaran, "Deep learning sensor fusion for autonomous vehicle perception and localization: A review," *Sensors*, vol. 20, no. 15, p. 4220, Jul. 2020.

[35] S. Xu, D. Zhou, J. Fang, J. Yin, Z. Bin, and L. Zhang, "FusionPainting: Multimodal fusion with adaptive attention for 3D object detection," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, USA, Oct. 2021, pp. 3047–3054.

[36] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, May 2018.

[37] G. Velasco-Hernandez, D. J. Yeong, J. Barry, and J. Walsh, "Autonomous driving architectures, perception and data fusion: A review," in *Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, Sep. 2020, pp. 315–321.

[38] A. Best, S. Narang, D. Barber, and D. Manocha, "AutonoVi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, Canada, Sep. 2017, pp. 2629–2636.

[39] J. Chen, S. E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5068–5078, Jun. 2022.

[40] S. Chen, M. Wang, W. Song, Y. Yang, Y. Li, and M. Fu, "Stabilization approaches for reinforcement learning-based end-to-end autonomous driving," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 4740–4750, May 2020.

[41] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, USA, Jun. 2017, pp. 1856–1860.

[42] 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 *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, Nov. 2019, pp. 2629–2634.

[43] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A survey of end-to-end driving: Architectures and training methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, Apr. 2022.

[44] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, "A survey on imitation learning techniques for end-to-end autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 128–14 147, Sep. 2022.

[45] S. Moten, F. Celiberti, M. Grottoli, A. van der Heide, and Y. Lemmens, "X-in-the-loop advanced driving simulation platform for the design, development, testing and validation of ADAS," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Changshu, China, Jun. 2018, pp. 1–6.

[46] T. Wu, A. Luo, R. Huang, H. Cheng, and Y. Zhao, "End-to-end driving model for steering control of autonomous vehicles with future spatiotemporal features," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, Nov. 2019, pp. 950–955.

[47] D. Omeiza, H. Web, M. Jirotka, and L. Kunze, "Towards accountability: Providing intelligible explanations in autonomous driving," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Nagoya, Japan, Jul. 2021, pp. 231–237.

[48] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, Mar. 2021.

[49] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, Apr. 2020.

[50] A. Ngo, M. P. Bauer, and M. Resch, "A multi-layered approach for measuring the simulation-to-reality gap of radar perception for autonomous driving," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, USA, Sep. 2021, pp. 4008–4014.

[51] J. Zhou, R. Wang, X. Liu, Y. Jiang, S. Jiang, J. Tao, J. Miao, and S. Song, "Exploring imitation learning for autonomous driving with feedback synthesizer and differentiable rasterization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, Sep. 2021, pp. 1450–1457.

[52] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kndall, "Urban driving with conditional imitation learning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, Aug. 2020, pp. 251–257.

[53] O. Natan and J. Miura, "Semantic segmentation and depth estimation with RGB and DVS sensor fusion for multi-view driving perception," in *Proceedings of the Asian Conference on Pattern Recognition (ACPR)*, Jeju Island, South Korea, Nov. 2021, pp. 352–365.

[54] O. Natan and J. Miura, "Towards compact autonomous driving perception with balanced learning and multi-sensor fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16 249–16 266, Sep. 2022.

[55] O. Natan and J. Miura, "End-to-end autonomous driving with semantic depth cloud mapping and multi-agent," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 557–571, Jan. 2022.

[56] O. Natan and J. Miura, "DeepIPC: Deeply integrated perception and control for an autonomous vehicle in real environments," *arXiv:2207.09934*, 2022. [Online]. Available: https://arxiv.org/abs/2207.09934

[57] O. Natan and J. Miura, "DeepIPCv2: LiDAR-powered robust environmental perception and navigational control for autonomous vehicle," *arXiv:2307.06647*, 2023. [Online]. Available: https://arxiv.org/abs/2307.06647

[58] B. Ranft and C. Stiller, "The role of machine vision for intelligent vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 8–19, Mar. 2016.

[59] S. Matsuzaki, J. Miura, and H. Masuzawa, "Multi-source pseudo-label learning of semantic segmentation for the scene recognition of agricultural mobile robots," *Advanced Robotics*, vol. 36, no. 19, pp. 1011–1029, Aug. 2022.

[60] H. Masuzawa and J. Miura, "Image-based recognition of green perilla leaves using a deep neural network for robotic harvest support," *Advanced Robotics*, vol. 35, no. 6, pp. 359–367, Jan. 2021.

[61] M. Hahner, D. Dai, C. Sakaridis, J.-N. Zaech, and L. V. Gool, "Semantic understanding of foggy scenes with purely synthetic data," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand, Oct. 2019, pp. 3675–3681.

[62] R. N. Rajaram, E. Ohn-Bar, and M. M. Trivedi, "RefineNet: Refining object detectors for autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 4, pp. 358–368, Dec. 2016.

[63] J. Wang, X. Wang, T. Shen, Y. Wang, L. Li, Y. Tian, H. Yu, L. Chen, J. Xin, X. Wu, N. Zheng, and F.-Y. Wang, "Parallel vision for long-tail regularization: Initial results from IVFC autonomous driving testing," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 2, pp. 286–299, Jun. 2022.

[64] P. C. Ravoor and S. T.S.B., "Deep learning methods for multi-species animal re-identification and tracking - a survey," *Computer Science Review*, vol. 38, p. 100289, Nov. 2020.

[65] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, and R. Urtasun, "MultiNet: Real-time joint semantic reasoning for autonomous driving," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Changshu, China, Jun. 2018, pp. 1013–1020.

[66] J. Kocic, N. Jovicic, and V. Drndarevic, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, vol. 19, no. 9, p. 2064, Mar. 2019.

[67] R. Cipolla, Y. Gal, and A. Kendall, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, USA, Jun. 2018, pp. 7482–7491.

[68] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, Jun. 2016, pp. 3213–3223.

[69] J. Yoo and R. Langari, "A predictive perception model and control strategy for collision-free autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 11, pp. 4078–4091, Nov. 2019.

[70] S. Teng, L. Chen, Y. Ai, Y. Zhou, Z. Xuanyuan, and X. Hu, "Hierarchical interpretable imitation learning for end-to-end autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 673–683, Jan. 2023.

[71] K. Ishihara, A. Kanervisto, J. Miura, and V. Hautamaki, "Multi-task learning with attention for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Nashville, USA, Jun. 2021, pp. 2896–2905.

[72] K. Chitta, A. Prakash, and A. Geiger, "NEAT: Neural attention fields for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, Canada, Oct. 2021, pp. 15 773–15 783.

[73] C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, P. Furgale, T. Sattler, and M. Pollefeys, "3D visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection," *Image and Vision Computing*, vol. 68, pp. 14–27, Dec. 2017.

[74] T. Suzuki, K. Ohno, S. Kojima, N. Miyamoto, T. Suzuki, T. Komatsu, Y. Shibata, K. Asano, and K. Nagatani, "Estimation of articulated angle in six-wheeled dump trucks using multiple gnss receivers for autonomous driving," *Advanced Robotics*, vol. 35, no. 23, pp. 1376–1387, Sep. 2021.

[75] I. Alonso and A. C. Murillo, "EV-SegNet: Semantic segmentation for event-based cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Long Beach, USA, Jun. 2019, pp. 1624–1633.

[76] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A deep learning-based radar and camera sensor fusion architecture for object detection," in *Proceedings of the Sensor Data Fusion: Trends, Solutions, Applications (SDF)*, Bonn, Germany, Oct. 2019, pp. 1–7.

[77] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, USA, Jun. 2020, pp. 11 618–11 628.

[78] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, USA, Jul. 2017, pp. 936–944.

[79] U. Niesen and J. Unnikrishnan, "Camera-radar fusion for 3-D depth reconstruction," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Las Vegas, USA, Oct. 2020, pp. 265–271.

[80] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, USA, Jun. 2021, pp. 7073–7083.

[81] N. Rhinehart, R. Mcallister, K. Kitani, and S. Levine, "PRECOG: Prediction conditioned on goals in visual multi-agent settings," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, South Korea, Nov. 2019, pp. 2821–2830.

[82] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "TransFuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. [Online]. Available: https://doi.org/10.1109/TPAMI.2022.3200245

[83] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proceedings of the Workshop Syntax, Semantics and Structure in Statistical Translation (SSST)*, Doha, Qatar, Oct. 2014, pp. 103–111.

[84] H.-C. Shao, L. Wang, R. Chen, H. Li, and Y. T. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," in *Proceedings of the Annual Conference on Robot Learning (CoRL)*, Auckland, New Zealand, Dec. 2022, pp. 1–12.

[85] M. Shan, Y. Zou, M. Guan, C. Wen, and C.-L. Ng, "A leader-following approach based on probabilistic trajectory estimation and virtual train model," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Yokohama, Japan, Oct. 2017, pp. 1–6.

[86] Z. Huang, C. Lv, Y. Xing, and J. Wu, "Multi-modal sensor fusion-based deep neural network for end-to-end autonomous driving with scene understanding," *IEEE Sensors Journal*, vol. 21, no. 10, pp. 11 781–11 790, May 2021.

[87] J. V. Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 384–406, Apr. 2018.

[88] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, Dec. 2019.

[89] V. Ravi Kumar, S. Yogamani, H. Rashed, G. Sitsu, C. Witt, I. Leang, S. Milz, and P. Mäder, "OmniDet: Surround view cameras based multi-task visual perception network for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2830–2837, Apr. 2021.

[90] 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, Nov. 2020.

[91] T. Gong, T. Lee, C. Stephenson, V. Renduchintala, S. Padhy, A. Ndirango, G. Keskin, and O. H. Elibol, "A comparison of loss weighting strategies for multi-task learning in deep neural networks," *IEEE Access*, vol. 7, pp. 141 627–141 632, Sep. 2019.

[92] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Venice, Italy, Oct. 2017, pp. 2070–2079.

[93] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, USA, Jun. 2019, pp. 1871–1880.

[94] I. Leang, G. Sistu, F. Bürger, A. Bursuc, and S. Yogamani, "Dynamic task weighting methods for multi-task networks in autonomous driving systems," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Rhodes, Greece, Sep. 2020, pp. 1–8.

[95] H. Lv, C. Liu, X. Zhao, C. Xu, Z. Cui, and J. Yang, "Lane marking regression from confidence area detection to field inference," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 1, pp. 47–56, Mar. 2021.

[96] L. Chen, Z. Yang, J. Ma, and Z. Luo, "Driving scene perception network: Real-time joint detection, depth estimation and semantic segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Lake Tahoe, USA, Mar. 2018, pp. 1283–1291.

[97] A. T. M. Nakamura, V. Grassi, and D. F. Wolf, "An effective combination of loss gradients for multi-task learning applied on instance segmentation and depth estimation," *Engineering Applications of Artificial Intelligence*, vol. 100, p. 104205, Apr. 2021.

[98] F. Yan, K. Wang, B. Zou, L. Tang, W. Li, and C. Lv, "LiDAR-based multi-task road perception network for autonomous vehicles," *IEEE Access*, vol. 8, pp. 86 753–86 764, May 2020.

[99] M. P. Muresan and S. Nedevschi, "Multi-object tracking of 3D cuboids using aggregated features," in *Proceedings of the IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, Cluj-Napoca, Romania, Sep. 2019, pp. 11–18.

[100] N. Dawar and N. Kehtarnavaz, "Action detection and recognition in continuous action streams by deep learning-based sensing fusion," *IEEE Sensors Journal*, vol. 18, no. 23, pp. 9660–9668, Dec. 2018.

[101] J. Nie, J. Yan, H. Yin, L. Ren, and Q. Meng, "A multimodality fusion deep neural network and safety test strategy for intelligent vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 310–322, Jun. 2021.

[102] L. Reiher, B. Lampe, and L. Eckstein, "A sim2real deep learning approach for the transformation of images from multiple vehicle-mounted cameras to a semantically segmented image in bird's eye view," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Rhodes, Greece, Sep. 2020, pp. 1–7.

[103] A. Palazzi, G. Borghi, D. Abati, S. Calderara, and R. Cucchiara, "Learning to map vehicles into bird's eye view," in *Proceedings of the International Conference on Image Analysis and Processing (ICIAP)*, Catania, Italy, Sep. 2017, pp. 233–243.

[104] K. Mani, S. Daga, S. Garg, N. S. Shankar, J. K. Murthy, and K. M. Krishna, "Mono lay out: Amodal scene layout from a single image," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Snowmass, USA, Mar. 2020, pp. 1678–1686.

[105] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, USA, Jul. 2017, pp. 77–85.

[106] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "PointCNN: Convolution on X-transformed points," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Montreal, Canada, Dec. 2018, pp. 828–838.

[107] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 945–953.

[108] A. Kanezaki, Y. Matsushita, and Y. Nishida, "RotationNet for joint object categorization and unsupervised pose estimation from multi-view images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 269–283, Jan. 2021.

[109] M. Imad, O. Doukhi, and D.-J. Lee, "Transfer learning-based semantic segmentation for 3D object detection from point cloud," *Sensors*, vol. 21, no. 12, p. 3964, Jun. 2021.

[110] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, USA, Jun. 2018, pp. 7652–7660.

[111] Y. Zhang, Z. Zhou, P. David, X. Yue, Z. Xi, B. Gong, and H. Foroosh, "PolarNet: An improved grid representation for online LiDAR point clouds semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, USA, Jun. 2020, pp. 9598–9607.

[112] J. Chen, Z. Xu, and M. Tomizuka, "End-to-end autonomous driving perception with sequential latent representation learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, USA, Oct. 2020, pp. 1999–2006.

[113] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "GradNorm: Gradient normalization for adaptive loss balancing in deep multi-task networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, Stockholm, Sweden, Jul. 2018, pp. 794–803.

[114] J. C. Ye and W. K. Sung, "Understanding geometry of encoder-decoder CNNs," in *Proceedings of the International Conference on Machine Learning (ICML)*, Long Beach, USA, Jun. 2019, pp. 7064–7073.

[115] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Munich, Germany, Oct. 2015, pp. 234–241.

[116] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the International Conference on Machine Learning (ICML)*, Lille, France, Jul. 2015, pp. 448–456.

[117] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the International Conference on Machine Learning (ICML)*, Haifa, Israel, Jun. 2010, pp. 807–814.

[118] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, Jun. 2014.

[119] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Denver, USA, Dec. 1991, pp. 950–957.

[120] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the*

IEEE/CVF International Conference on Computer Vision (ICCV), Santiago, Chile, Dec. 2015, pp. 1026–1034.

[121] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, Atlanta, USA, Jun. 2013, pp. 1139–1147.

[122] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high performance deep learning library," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, Dec. 2019, pp. 8024–8035.

[123] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the Annual Conference on Robot Learning (CoRL)*, Mountain View, USA, Nov. 2017, pp. 1–16.

[124] K. J. Cantrell., C. D. Miller., and C. W. Morato., "Practical depth estimation with image segmentation and serial U-Nets," in *Proceedings of the International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS)*, Online, May 2020, pp. 406–414.

[125] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[126] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, USA, May 2015.

[127] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, USA, May 2015.

[128] W. Ren, K. Jiang, X. Chen, T. Wen, and D. Yang, "Adaptive sensor fusion of camera, GNSS and IMU for autonomous driving navigation," in *Proceedings of the International Conference on Vehicular Control and Intelligence (CVCI)*, Hangzhou, China, Dec. 2020, pp. 113–118.

[129] Y. Cui, R. Chen, W. Chu, L. Chen, D. Tian, Y. Li, and D. Cao, "Deep learning for image and point cloud fusion in autonomous driving: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 722–739, Feb. 2022.

[130] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, "Multimodal end-to-end autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 537–547, Jan. 2022.

[131] S. Chowdhuri, T. Pankaj, and K. Zipser, "MultiNet: Multi-modal multi-task learning for autonomous driving," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, USA, Jan. 2019, pp. 1496–1504.

[132] P. Palanisamy, "Multi-agent connected autonomous driving using deep reinforcement learning," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Glasgow, UK, Jul. 2020, pp. 1–7.

[133] K. Nagarajan and Z. Yi, "Lane changing using multi-agent DQN," in *Proceedings of the IEEE International Conference on Autonomous Systems (ICAS)*, Montreal, Canada, Aug. 2021, pp. 1–6.

[134] F. Salehi Rizi and M. Granitzer, "Multi-task network embedding with adaptive loss weighting," in *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, Hague, Netherlands, Dec. 2020, pp. 1–5.

[135] T. Wang and S.-C. Chen, "Multi-label multi-task learning with dynamic task weight balancing," in *Proceedings of the IEEE International Conference on Information Reuse and Integration for Data Science (IRI)*, Las Vegas, USA, Aug. 2020, pp. 245–252.

[136] R. Yamada, K. Yamamori, and T. Tasaki, "Pose estimation of a simple-shaped object based on poseclass using RGBD camera," in *Proceedings of the IEEE/SICE International Symposium System Integration (SII)*, Fukushima, Japan, Jan. 2021, pp. 426–430.

[137] R. Pandey, A. Tkach, S. Yang, P. Pidlypenskyi, J. Taylor, R. Martin-Brualla, A. Tagliasacchi, G. Papandreou, P. Davidson, C. Keskin, S. Izadi, and S. Fanello, "Volumetric capture of humans with a single RGBD camera via semi-parametric learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, USA, Jun. 2019, pp. 9701–9710.

[138] L. Xu, Z. Su, L. Han, T. Yu, Y. Liu, and L. Fang, "UnstructuredFusion: Realtime 4D geometry and texture reconstruction using commercial RGBD cameras," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2508–2522, Oct. 2020.

[139] K. Yousif, Y. Taguchi, and S. Ramalingam, "MonoRGBD-SLAM: Simultaneous localization and mapping using both monocular and RGBD cameras," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017, pp. 4495–4502.

[140] J. Cao, S. Leng, and K. Zhang, "Multi-agent learning empowered collaborative decision for autonomous driving vehicles," in *Proceedings of the International Conference on UK-China Emerging Technologies (UCET)*, Glasgow, UK, Aug. 2020, pp. 1–4.

[141] N. Ayache, A. Yahyaouy, and S. M. Abdelouahed, "An autonomous vehicular system based on muli-agents control: Architecture and behavior simulation," in *Proceedings of the Intelligent Systems and Computer Vision (ISCV)*, Fez, Morocco, Apr. 2017, pp. 1–7.

[142] J. Park, K. Min, and K. Huh, "Multi-agent deep reinforcement learning for cooperative driving in crowded traffic scenarios," in *Proceedings of the International Symposium Intelligent Signal Processing and Communication Systems (ISPACS)*, Taipei, Taiwan, Dec. 2019, pp. 1–2.

[143] A. O. Ly and M. Akhloufi, "Learning to drive by imitation: An overview of deep behavior cloning methods," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 195–209, Jun. 2021.

[144] F. Codevilla, E. Santana, A. M. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, South Korea, Nov. 2019, pp. 9328–9337.

[145] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, Jun. 2016, pp. 770–778.

[146] D. Chen, B. Zhou, V. Koltun, and P. Krahenbuhl, "Learning by cheating," in *Proceedings of the Annual Conference on Robot Learning (CoRL)*, Osaka, Japan, Oct. 2019, pp. 1–10.

[147] A. Filos, P. Tigkas, R. Mcallister, N. Rhinehart, S. Levine, and Y. Gal, "Can autonomous vehicles identify, recover from, and adapt to distribution shifts?" in *Proceedings of the International Conference on Machine Learning (ICML)*, Vienna, Austria, Jul. 2020, pp. 3145–3153.

[148] C. Guo, K. Kidono, R. Terashima, and Y. Kojima, "Humanlike behavior generation in urban environment based on learning-based potentials with a low-cost lane graph," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 1, pp. 46–60, Mar. 2018.

[149] C. Guo, K. Kidono, T. Machida, R. Terashima, and Y. Kojima, "Human-like behavior generation for intelligent vehicles in urban environment based on a hybrid potential map," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, USA, Jun. 2017, pp. 197–203.

[150] C. Guo, T. Owaki, K. Kidono, T. Machida, R. Terashima, and Y. Kojima, "Toward human-like lane following behavior in urban environment with a learning-based behavior-induction potential map," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, Jun. 2017, pp. 1409–1416.

[151] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, Long Beach, USA, Jun. 2019, pp. 6105–6114.

[152] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, USA, Jun. 2009, pp. 248–255.

[153] O. Natan, D. U. K. Putri, and A. Dharmawan, "Deep learning-based weld spot segmentation using modified UNet with various convolutional blocks," *ICIC Express Letters Part B: Applications*, vol. 12, no. 12, pp. 1169–1176, Dec. 2021.

[154] S. Yang, X. Yu, and Y. Zhou, "LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example," in *Proceedings of the International Workshop Electronic Communication and Artificial Intelligence (IWECAI)*, Shanghai, China, Jun. 2020, pp. 98–101.

[155] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, New Orleans, USA, May 2019.

[156] M. Liang, B. Yang, W. Zeng, Y. Chen, R. Hu, S. Casas, and R. Urtasun, "PnPNet: End-to-end perception and prediction with tracking in the loop," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, USA, Jun. 2020, pp. 11 550–11 559.

[157] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3D object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, USA, Jun. 2019, pp. 7337–7345.

[158] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3D object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, Sep. 2018, pp. 1–16.

[159] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and A. Hartwig, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, Sep. 2018, pp. 833–851.

[160] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, May 2009.

[161] M. Teti, W. E. Hahn, S. Martin, C. Teti, and E. Barenholtz, "A controlled investigation of behaviorally-cloned deep neural network behaviors in an autonomous steering task," *Robotics and Autonomous Systems*, vol. 142, p. 103780, Aug. 2021.

[162] F. Sasaki, T. Yohira, and A. Kawaguchi, "Adversarial behavioral cloning," *Advanced Robotics*, vol. 34, no. 9, pp. 592–598, Feb. 2020.

[163] H. Shen, W. Wan, and H. Wang, "Learning category-level generalizable object manipulation policy via generative adversarial self-imitation learning from demonstrations," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 166–11 173, Oct. 2022.

[164] D.-T. Pham, T.-N. Tran, S. Alam, and V. N. Duong, "A generative adversarial imitation learning approach for realistic aircraft taxi-speed modeling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 2509–2522, Mar. 2022.

[165] X. Fang, Q. Zhang, Y. Gao, and D. Zhao, "Offline reinforcement learning for autonomous driving with real world driving data," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Macau, China, Oct. 2022, pp. 3417–3422.

[166] R. Bhattacharyya, B. Wulfe, D. J. Phillips, A. Kuefler, J. Morton, R. Senanayake, and M. J. Kochenderfer, "Modeling human driving behavior through generative adversarial imitation learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 3, pp. 2874–2887, Mar. 2023.

[167] P. M. Kebria, A. Khosravi, I. Hossain, N. Mohajer, H. D. Kabir, S. M. J. Jalali, D. Nahavandi, S. M. Salaken, S. Nahavandi, A. Lagrandcourt, and N. Bhasin, "Autonomous navigation via deep imitation and transfer learning: A comparative study," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Toronto, Canada, Oct. 2020, pp. 2907–2912.

[168] P. Cai, H. Wang, H. Huang, Y. Liu, and M. Liu, "Vision-based autonomous car racing using deep imitative reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7262–7269, Oct. 2021.

[169] A. Chatty, P. Gaussier, S. K. Hasnain, I. Kallel, and A. M. Alimi, "The effect of learning by imitation on a multi-robot system based on the coupling of low-level imitation strategy and online learning for cognitive map building," *Advanced Robotics*, vol. 28, no. 11, pp. 731–743, Feb. 2014.

[170] S. Hoshino and K. Unuma, "End-to-end motion planners through multi-task learning for mobile robots with 2D LiDAR," in *Proceedings of the IEEE/SICE International Symposium on System Integration (SII)*, Atlanta, USA, Jan. 2023, pp. 1–6.

[171] S. Yan, Z. Wu, J. Wang, Y. Huang, M. Tan, and J. Yu, "Real-world learning control for autonomous exploration of a biomimetic robotic shark," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 4, pp. 3966–3974, Apr. 2023.

[172] D. Xu, Z. Ding, X. He, H. Zhao, M. Moze, F. Aioun, and F. Guillemard, "Learning from naturalistic driving data for human-like autonomous highway driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7341–7354, Dec. 2021.

[173] H. Ma, Y. Wang, R. Xiong, S. Kodagoda, and L. Tang, "DeepGoal: Learning to drive with driving intention from human control demonstration," *Robotics and Autonomous Systems*, vol. 127, p. 103477, May 2020.

[174] F. S. Acerbo, M. Alirczaei, H. Van Der Auweraer, and T. D. Son, "Safe imitation learning on real-life highway data for human-like autonomous driving," in *Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, USA, Sep. 2021, pp. 3903–3908.

[175] H. Fujiishi, T. Kobayashi, and K. Sugimoto, "Safe and efficient imitation learning by clarification of experienced latent space," *Advanced Robotics*, vol. 35, no. 16, pp. 1012–1027, Jul. 2021.

[176] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Online, Dec. 2021, pp. 1–18.

[177] R. W. Wolcott and R. M. Eustice, "Robust LiDAR localization using multiresolution Gaussian mixture maps for autonomous driving," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 292–319, Apr. 2017.

[178] S. McCrae and A. Zakhor, "3D object detection for autonomous driving using temporal LiDAR data," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Abu Dhabi, UAE, Oct. 2020, pp. 2661–2665.

[179] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for LiDAR point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, Aug. 2021.

[180] Y. Li and J. Ibanez-Guzman, "LiDAR for autonomous driving: The principles, challenges, and trends for automotive LiDAR and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, Jul. 2020.

[181] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss, "Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset," *The International Journal of Robotics Research*, vol. 40, no. 8-9, pp. 959–967, Apr. 2021.

[182] X. Yan, J. Gao, C. Zheng, C. Zheng, R. Zhang, S. Cui, and Z. Li, "2DPASS: 2D priors assisted semantic segmentation on LiDAR point clouds," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Tel Aviv, Israel, Oct. 2022, pp. 677–695.

[183] Y. Hou, X. Zhu, Y. Ma, C. C. Loy, and Y. Li, "Point-to-voxel knowledge distillation for LiDAR semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, USA, Jun. 2022, pp. 8469–8478.

[184] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[185] D. Sun, Q. Liao, and A. Loutfi, "Type-2 fuzzy model-based movement primitives for imitation learning," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2462–2480, Aug. 2022.

[186] M. Alibeigi, M. N. Ahmadabadi, and B. N. Araabi, "A fast, robust, and incremental model for learning high-level concepts from human motions by imitation," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 153–168, Feb. 2017.

[187] D. Wang, C. Devin, Q.-Z. Cai, F. Yu, and T. Darrell, "Deep object-centric policies for autonomous driving," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019, pp. 8853–8859.

[188] Y. Xu, X. Yang, L. Gong, H.-C. Lin, T.-Y. Wu, Y. Li, and N. Vasconcelos, "Explainable object-induced action decision for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, USA, Jun. 2020, pp. 9520–9529.

[189] W. Viriyasitavat, M. Boban, H.-M. Tsai, and A. Vasilakos, "Vehicular communications: Survey and challenges of channel and propagation models," *IEEE Vehicular Technology Magazine*, vol. 10, no. 2, pp. 55–66, Jun. 2015.

[190] I. Agudo, M. Montenegro-Gómez, and J. Lopez, "A blockchain approach for decentralized V2X (D-V2X)," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4001–4010, May 2021.

[191] X. Liang, T. Wang, L. Yang, and E. Xing, "CIRL: Controllable imitative reinforcement learning for vision-based self-driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Munich, Germany, Sep. 2018, pp. 604–620.

[192] P. Cai, H. Wang, H. Huang, Y. Liu, and M. Liu, "Vision-based autonomous car racing using deep imitative reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7262–7269, Oct. 2021.

[193] H. Blum, A. Dietmüller, M. Milde, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor," in *Proceedings of Robotics: Science and Systems Conference (RSS)*, Massachusetts, USA, Jul. 2017.

[194] T. Speth, A. Kamann, T. Brandmeier, and U. Jumar, "Precise relative egopositioning by stand-alone RTK-GPS," in *Proceedings of the Workshop on Positioning, Navigation and Communications (WPNC)*, Bremen, Germany, Oct. 2016, pp. 1–6.

[195] D. Brüggemann, C. Sakaridis, P. Truong, and L. Van Gool, "Refign: Align and refine for adaptation of semantic segmentation to adverse conditions," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, USA, Jan. 2023, pp. 3173–3183.

[196] L. Hoyer, D. Dai, and L. Van Gool, "HRDA: Context-aware high-resolution domain-adaptive semantic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Tel Aviv, Israel, Oct. 2022, pp. 1–31.

[197] C. Sakaridis, D. Dai, and L. Van Gool, "ACDC: The adverse conditions dataset with correspondences for semantic driving scene understanding," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, Canada, Oct. 2021, pp. 10 745–10 755.

[198] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and ground-optimized LiDAR odometry and mapping on variable terrain," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, Oct. 2018, pp. 4758–4765.

[199] J. Zhang and S. Singh, "LOAM: LiDAR odometry and mapping in real-time," in *Proceedings of Robotics: Science and Systems Conference (RSS)*, Berkeley, USA, Jul. 2014.

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

[201] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, Canada, Oct. 2021, pp. 12 159–12 168.

[202] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, Long Beach, USA, Dec. 2017, pp. 6000–6010.

Publication

Journal

- O. Natan and J. Miura, "End-to-end Autonomous Driving with Semantic Depth Cloud Mapping and Multi-agent," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 557-571, Jan. 2023.

  - Code: https://github.com/oskarnatan/end-to-end-driving

  - Presented at IEEE Intelligent Vehicles Symposium (IV) 2023

- O. Natan and J. Miura, "Towards Compact Autonomous Driving Perception with Balanced Learning and Multi-sensor Fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16249-16266, Sept. 2022.

  - Code: https://github.com/oskarnatan/compact-perception

  - Presented at IEEE Intelligent Vehicles Symposium (IV) 2023

Conference

- O. Natan and J. Miura, "Semantic Segmentation and Depth Estimation with RGB and DVS Sensor Fusion for Multi-view Driving Perception," in *Proceedings of the Asian Conference on Pattern Recognition (ACPR)*, Jeju Island, South Korea, Nov. 2021, pp. 352–365.

  - Code: https://github.com/oskarnatan/RGBDVS-fusion

Preprint (under review)

- O. Natan and J. Miura, “DeepIPCv2: LiDAR-powered Robust Environmental Perception and Navigational Control for Autonomous Vehicle,” arXiv:2307.06647, 2023.

  - Code: https://github.com/oskarnatan/DeepIPCv2

- O. Natan and J. Miura, "DeepIPC: Deeply Integrated Perception and Control for an Autonomous Vehicle in Real Environments," arXiv:2207.09934, 2022.

  - Code: https://github.com/oskarnatan/DeepIPC

Acknowledgement

I gratefully acknowledge everyone who gives me plenty of support, especially during the past years of my study in Japan. Tough words are limited, but I would like to express my sincere gratitude to:

- Professor Jun Miura, my sensei who gives me a lot of things. I would like to thank him for everything, especially for boosting my research experience including how he guides me in writing nice papers to be published and presented in prestigious journals and conferences. I really appreciate his expertise and vast knowledge, especially in the field of deep learning and sensor fusion for end-to-end autonomous driving. I am truly happy to be your student and hopefully, we can always stay keep in touch in the future.
- Professor Shigeru Kuriyama and Professor Yohei Kakiuchi, the examining committee who kindly provide valuable comments and suggestions to improve the presentation and contents of this thesis.
- Kazushige Yano, Atsuki Osanai, Yasuhiro Taniguchi, and other staff at Honda RnD Co., Ltd. who accept me as a research intern and give me an opportunity to join the autonomous driving project. I am really happy to be a part of the team, I learn a lot.
- Mikiko Kobayashi and Asaha Murai, secretary of Active Intelligent Systems Laboratory (AISL) who assist me in doing paperwork for many activities such as attending international conferences, procuring sensors, etc. Thank you very much.
- Edo, Santika, and Baskara, my best friends from Indonesia who also study at TUT. I am really glad I have you all here, thank you for caring and helping me in many ways. I wish you all success, ganbatte!
- All my labmates at AISL, especially those who stay in C2-501 (Matsuzaki, Mano, Higashimoto, Liliana, Masuzawa, Sinem, Uzawa, Liu, Chandra, David, Sano, Nakano, Tamiya, Baskara, Ørjan), members of the driving research group (Baskara, Ørjan, Teo, Emil, Patrik, Yamagata, Ishihara), my student supporter (Miake), and also Asst. Prof. Kotaro Hayashi who help me and discuss with me about many things. They also show their strong progress that drives me to be better day-by-day. Thanks guys!
- My beloved family in Indonesia (mom, dad, big bro and his little family), special people and a whole support system that always be there for me. Thank you so much!
- MEXT and TUT for providing me with generous financial support for my study and living expenses.