Dissertation of Practical Course SIMULATION-BASED AUTONOMOUS DRIVING IN CROWDED CITY

Zhaoyu Chen, Runsheng Chen

***Abstract*—Autonomous driving is a megatrend of the application of artificial intelligence in the recent decades. The first driverless car in public road came out in 1925 New York City, which was a radio-operated automobile. The first software concept for modern autonomous driving was proposed by John McCarthy, pioneer of AI, in 1969, who claimed that a system required a computer in the car equipped with television camera input that uses the same visual input available to the human driver[1]. Nowadays, we have made a great progress in the field of autonomous driving, but not yet reached the stage of full automation. In this article, we dive deeper into a so called End-to-End approach for autonomous driving aiming to achieve full automation in crowded city based on simulator of udacity. We review the state of current research in the field and the methodologies we used, including data collection, input and output of models, and our result of experiments. Discussion of End-to-End approach is carried out in the end of our dissertation as well.**

***Impact Statement*—Autonomous driving is a popular technology in the automobile industry. People gain a more convenient, efficient, and secure transportation from it. And a driver is no longer necessary anymore. However, with the more and more utilization of autonomous/assisted driving systems, an increased number of traffic accidents are recorded. Not only in terms of a number of accidents but also in terms of the level of danger in an accident, the issues with current autonomous driving are evident. The end-to-end approach introduced in this article might improve the performance of current autonomous driving systems in a crowded city, which is one of the most difficult scenarios for autonomous vehicles. Nevertheless, this technology is not validated in the real-world safeguarding test with actual vehicles equipped with our algorithm.**

***Index Terms*—Autonomous driving, end-to-end, simulation-based, neural networks, imitation learning, reinforcement learning**

I. INTRODUCTION

AUTONOMY in the context of robotics is the ability of a robot to operate without human intervention[2]. In the context of the automobile industry, automation can be interpreted as driverless driving or autonomous driving, which means the vehicle operates in a public traffic road from initial to destination without human intervention using multiple sensory, such as camera, radar, LiDAR, and advanced algorithms to map the data perceived by sensors into outputs in different semantic spaces. Autonomous driving aims to reach or even exceed the capability of human drivers, enabling the vehicle to steer, accelerate, and brake on its own in a reasonable trajectory. Often, autonomous driving ranges from low-level to higher-level automation, which is defined by the Society of

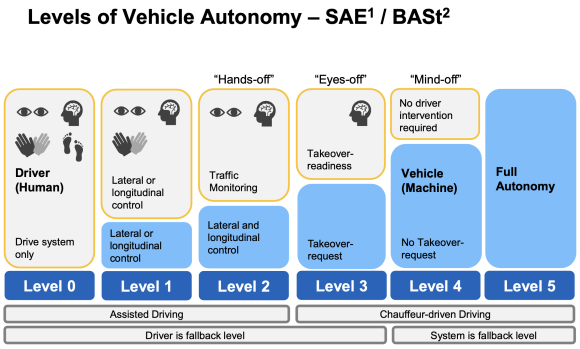

Fig. 1. Levels of vehicle automation - SAE/BASt. The level of automation (blue blocks) increases from level 0 to level 5 based on SAE, meanwhile human intervention (white blocks) reduces

Automotive Engineers (SAE) and Bundesanstalt Straßenwesen (BASt) shown in Figure 1. Most vehicles on the market are equipped with level 2 Partial automation, i.e. the automated system takes over longitudinal and lateral control, such as Lane keeping assistance (LKA), Adapted Cruise Control (ACC) and human driver monitors the traffic to prepare for taking over the control from systems in case, where the system cannot handle. The first consumer vehicle with a certified level 3 system is now equipped in Mercedes Benz, 2023, which can release the human driver's hands and feet, the human driver only keeps eyes on the traffic and has takeover readiness. However, there is still a long way to achieve level 5 Full automation. One of the difficult points in level 5 automation is that the vehicle is able to perform all driving functions under all conditions. And crowded city is one of the most common but complicated conditions for autonomous vehicles, due to incomplete object detection, stochastic processes of pedestrians and other vehicles, and interactions between other traffic participants.

On a broader level, the research on autonomous driving can be divided into two approaches: 1) modular and 2) end-to-end[3]. The modular approach, as the name suggests, have partitioned the system into several modules. The most widely used structure of modular approach consists of perception, localization (sometimes included in perception), planning, and control modules. It is also known as mediated perception approach, because each module outputs an intermediate result. Nowadays, the modular approach is considered a conventional approach. The major strength of conventional modular ap-

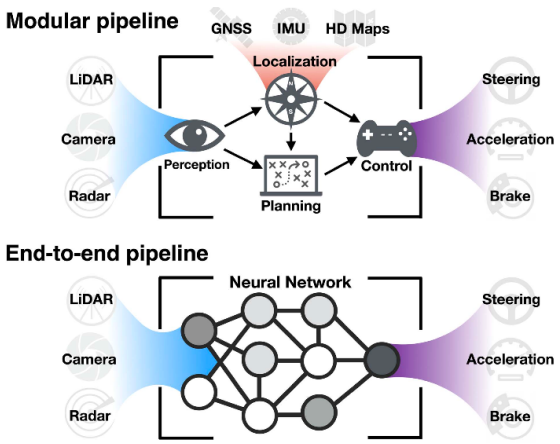

Fig. 2. Pipelines of modular and end-to-end approach. The modular pipeline for autonomous driving consists of multiple interconnected components, whereas the end-to-end pipeline covers the whole pipeline as a single learning task.[3]

proach is its interpretability – each subsystem represents a specific task.This property leads to easier identification of module at fault in case of malfunction and misbehavior. Besides, due to well-defined subtasks, people can only focus on the relatively simpler subsystems and improve each subsystem separately. However, the disadvantages of modular approach are indicated by the intricate design of interconnections between components and the human-defined information bottlenecks. For example, it is impossible for researcher to design the planner to cover all driving scenarios in the planning module, even the same objects detected, in the same location and the same map. Moreover, modular approaches usually take up unnecessary resources to implement relatively unimportant computations.

Compared to the modular approach, the end-to-end approach has become a growing trend in autonomous vehicle research[4][5][6][7][8][9][10][11]. End-to-end approach refers to modeling the entire process of modular approach, i.e from perception to controlling vehicles, by a single neural network. This unified structure directly maps raw data as inputs perceived by sensors into desired vehicle commands outputs, without any intermediate results of separate modules. Usually, the structure is much simpler than modular stack with much fewer components[3]. (See Fig.2) In end-to-end driving, the model is either learned via imitation learning (IL) given expert driving data to mimic human driver maneuvers or via reinforcement learning (RL) through sufficient exploration and training to improve its policy. The end-to-end driving (i.e. IL or RL) has its intrinsic superiority that exceeds the performance of modular methods, as there are no predefined information bottlenecks. Since in the modular approach, the driving task is regarded as several human-defined subsystems, each of them might not be optimal in all different driving scenarios, especially in a crowded city. The ability of end-to-end to extract task-oriented features leads to better performance. However, the utilization of end-to-end raises the weakness of interpretability. The model refers to a black box, the process of the model is unclear or not understandable for humans. This leads to difficulties of understanding malfunction and

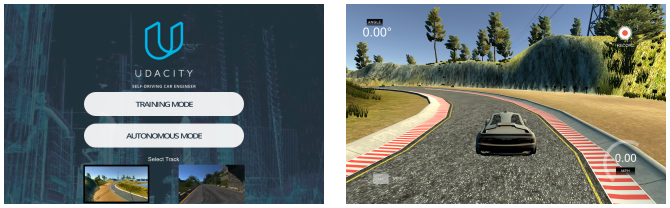

Fig. 3. The interface of our simulator is based on unity. Training mode, where we can collect driving data in the form of pictures and their corresponding steering angle, and throttle, either as expert driving data for IL or as dataset for modular approach. Test mode is connected to our trained model, so that it can receive commands for controlling the vehicle.

misbehavior. Debugging becomes even harder due to the lack of intermediate results.

In terms of testing and training the driving model, a simulation-based approach is one of the most common methods. This approach creates a virtual environment that approximates the real-world environment, where data collection and assessment of vehicle behavior can be applied safely. Moreover, the rapid iteration cannot be neglected. Developers can quickly modify and repeat simulations to improve algorithms, identify issues, and optimize performance. Fig.3 shows the simulator we used.

The paper is organized in terms of different learning methods important to autonomous driving systems. We start with introducing the current research status in the next section, showing the current technologies and algorithms used by not only academia but also manufacturers. In section III, we introduce the methodologies we used in our autonomous driving systems, including the detection module proposed by Runsheng Chen, and the end-to-end framework proposed by Zhaoyu Chen. Section IV indicates the experimental results of both two different approaches. Section V considers the evaluation and comparison of our methods, and analyzes the pros and cons of each methodology. In the last section, we will discuss the improvements in our methods, the safety of autonomous driving, and the future of autonomous driving, which are the prerequisites for the application of autonomous driving in the real world.

II. Current Status of Research

The current research in the field of autonomous driving is mainly partitioned by modular and end-to-end approaches as aforementioned. However there is no clear data or statement to support which approach is better or more promising. Nowadays, not only academia but also OEMs, such as Tesla, BMW, and Mercedes are committed to developing advanced autonomous driving systems. Most of them use modular approaches, such as Tesla, which heavily depends on camera-based systems and partially depends upon radar sensors. While some others are running on a different track – NIO became the first manufacturer to commercially use LiDAR sensors. However, commercial autonomous driving systems can only reach Level 2 for now, for instance, Autopilot from Tesla and many other similar systems. In this section, current technologies of autonomous driving will be introduced based on two different types, i.e. modular and end-to-end.

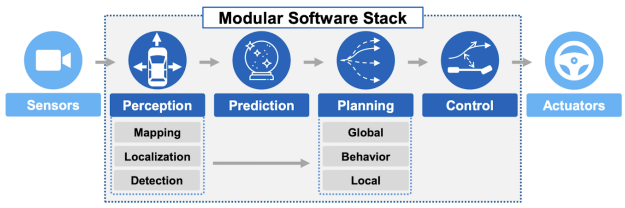

Fig. 4. The pipeline of a standard modular method. Each subsystem can be divided into several tiny steps, which are important for a self-driving vehicle

A. Modular approach

The modular approach is the most widely used method in reality. The entire pipeline of the modular approach consists of sensors, perception, prediction, planning, control, and actuators[12] (See Fig. 4). This pipeline starts with sending raw sensor data to the perception model to localize ego vehicle position and create the representation of the environment around the ego vehicle, followed by the prediction module, which foresees the trajectory or the maneuver of other traffic participants (vehicles, bicycles, pedestrians, etc.). Taking the results of prediction and perception into account, the ego vehicle implements decision-making hierarchically from global planning to local planning, creating a trajectory of the ego vehicle. All information collected by the ego vehicle will be sent to the last unit to generate motor commands(throttle, brake, steering angle) at the end of the stream by the control module [13][14]. In the following, we will see the main technologies applied in the aforementioned subsystems.

1) Perception: Localization and mapping is the task of finding ego-position relative to a reference frame in an environment [15], and it is fundamental to any mobile robot. It is especially crucial for ego vehicles [16]; the vehicle must use the correct lane and position itself in it accurately [17]. The three most common approaches in localization and mapping are detailed in the following, namely Global Positioning System - Inertial Measurement Unit (GPS-IMU) fusion, Simultaneous Localization And Mapping (SLAM), and Prior map-based localization. The main principle of GPS-IMU fusion is correcting the predicted value based on the measurement. Thus, Bayesian filters was first introduced into this method, including Kalman filter, extended Kalman filter, Unscented Kalman filter, and Particle filter (in order of increasing complexity). And mapping is done by other algorithms separately. However, there is a significant drawback: error accumulation over time, and this leads to failure in long-term operations [18]. Therefore, IMU-GPS fusion cannot be used in the reality. Compared to GPS-IMU fusion, SLAM is the current mainstream method used by manufacturers. It does not require prior knowledge of the environment, it makes a map and localizes the ego vehicle position at the same time. The details of SLAM can be regarded as using Bayesian filters to predict and update not only the position if ego vehicle (as same as GPS-IMU fusion), but also the mapping information. Nevertheless, computationally expensive is a fatal problem of this approach, since self-driving tasks are required in real time. The core idea of a priori map-based localization techniques is matching: localization is achieved through the comparison of online readings to the information on a detailed pre-built map and finding the location of the best possible match [19]. Often an initial pose estimation, for example with a GPS, is used at the beginning of the matching process. There are various approaches to map building and preferred modalities [17].

Detection plays a crucial role in autonomous driving, it refers to identifying the location and size of a particular object. Three detection tasks are traffic lights and traffic signs detection, road detection, and object detection. Detection tasks are regarded as the combination of classification and object localization, which means each detected object in a picture is bounded by a box called a ”bounding box”. There are three kinds of detection methods based on different sensor employment: camera-based (image-based) object detection, LiDAR detection, and Radar detection. Camera detection is the most common method since the camera is widely used due to its lower costs. In terms of camera-based object detection, speaking of You-Only-Look-Once (YOLO) [20] is not avoidable. YOLO consists of a CNN to extract image features on each grid that it divides into. A fully connected network then predicts the probability of different classes and the location of objects in each grid and creates a bounding box for each grid. Besides, LiDAR detection has become more and more popular in recent years. The difficulty is that the output of LiDAR sensor is a point cloud, which cannot be sent into the neural networks that we know by far. Two ways to tackle this challenge: Structured grid-based learning, 3D Point Cloud deep learning. The core of the former is voxelization into the 3D grid, then calculate the feature vector of each occupied voxel, for example, the density, mean position, and number of points inside the voxel. Another LiDAR-based detection method is 3D Point Cloud deep learning called PointNet, which accepts raw point cloud as input with 3-dimensional position (XYZ). With respect to Radar detection, a technique called Radar Cube is the latest. It creates a 3-dimensional tensor representing range, azimuth, and velocity respectively. This ’cube’ can then be treated with conventional neural networks.

2) Prediction: The prediction in autonomous driving is known as risk assessment. A robust automation system should constantly evaluate the overall risk level of the situation and predict the intentions of surrounding human drivers and pedestrians [17]. The prediction becomes the prerequisite for driving safely. For human drivers, anticipatory driving is realized through riding experience, which humans have learned in the past years from daily interaction with other traffic participants. In the case of automation, the prediction is referred to as motion anticipation through scenario understanding. However, scenario understanding is a challenging task for current systems, which are trapped by: interaction with other participants, stochastic processes, and incomplete object detection. Safety assessment can be achieved through two methodologies: knowledge-based and learning-based.

Knowledge-based prediction uses Bayesian methods to quantify and measure uncertainties of positions. The so-called ”Knowledge” is regarded as a State-Space Model, which is known as a motion model, for example, the Point-mass

Fig. 5. Global planning and local planning. The route (blue) from the initial to the goal. The local path (green) is generated in the local planner. The shown graph is produced by OpenPlanner [21].

model. Many of the motion models are based on kinematics, some variants are constant for simplification. The choice of a Bayesian filter that is applied in a knowledge-based approach depends upon the motion model and the distribution of states.

The concept of a learning-based model consists of sense-learn-predict, where *"learn"* is the difference between it and the knowledge-based approach. In the learning-based approach, the maneuvers of predicted objects are learnable through pattern clustering or classification. There are different ways to realize maneuver prediction with machine learning or deep learning methods that are low-dimensional and easily applicable. However, it is not possible to predict the trajectories, which extend the spatial space to temporal space giving the system much more information. Therefore, a trajectory-learnable method is crucial, which is known as an end-to-end task using encoder-decoder, transformer, and graph neural networks (GNN). To predict the trajectory of the object, the encoder-decoder accepts the previous trajectory until t_0 as input and the label is the trajectory from t_1. Thus, a well-trained network should be able to output the right trajectory given the past positions over time. The transformer architecture enhances the performance of this encoder-decoder structure. Moreover, GNN approximates the scenario by a graph with nodes of each traffic participant and links of velocity between nodes. Nowadays, inverse reinforcement learning is the megatrend in the prediction of objects. Nevertheless, inverse reinforcement learning in prediction is still an undiscovered area.

3) Planning: Planning is a task for the ego vehicle to find a route (global panning) and a path (local planning) from initial to destination. Figure 5 shows the differences between global planning and local planning in detail. Notice the difference between route, path, and trajectory planning. Route planning is the decision to take a route from source to destination. It is a sequence of discrete geometrical nodes in a map network. While path planning is a continuous curve in the spatial domain taking the geometrical and spatial boundary conditions into account. Compared to path planning, a trajectory planner considers the temporal domain additionally. It generates a continuous curve in the spatio-temporal domain.

The global planner is responsible for finding a route from a start node to the goal node. The map is reformulated as multiple nodes and links connecting each node. The global routes are examined under four categories: goal-directed, separator-based, hierarchical and bounded-hop techniques. A^* [22] and Dijkstra search algorithms are the standard goal-directed planning algorithm and has been used in various fields over decades. The principle of separator-based methods is removing a subset of vertices [23] or arcs from the nodes graph and computing an overlay graph over it. In the hierarchical method, designers divide the roads into several hierarchies according to their types. The shortest route may not be the fastest nor the most optimal route anymore. The bounded-hop techniques take advantage of precomputing distances between selected nodes and using them on the query time. The combination of the aforementioned methods are popular in state-of-the-art. The hybrid methods of separator-based and bounded-hop is proposed in [24].

Local planning refers to generating the trajectory that satisfies boundary conditions and criteria in the configuration space based on the pre-produced route in global planning. A local planner takes all previously collected information into account, including map, pose, predicted objects, and route. The methodologies of local planning can be divided into three groups: variational methods, graph-based methods, and incremental methods. In variational methods, a cost function is to be minimized under the constraints. For example, the energy function of following the reference path and reaching the desired velocity while reducing the acceleration and jerk.

$$L = w_{off}|x - x_{ref}|^2 + w_{vel}|\dot{x} - \dot{x}_{ref}|^2 + w_{acc}|\ddot{x}|^2 + w_{jerk}|x|^2 \tag{1}$$

The cost function is the integral of the energy function.

$$J(x(t)) = \int_{t_0}^{t_0+T} L(x, \dot{x}, \ddot{x})\, dt \tag{2}$$

The graph-based methods in local planning are similar to those in global planning – reformulate the route into nodes and apply Dijkstra[25] or A^*[22] search algorithm. Rapidly exploring random tree (RRT)[26] is a standard incremental method. RRT is a single query planner. The trajectories between the start and goal configuration are incrementally built with random tree-like branches. The main advantage of RRT is that it is probabilistically complete [26], which means a trajectory that satisfies all the constraints will be guaranteed to be found with a certain runtime. Moreover, deep learning-based planners are promising, but they are not widely used in real-world systems yet[17].

4) Control: Control systems are not peculiar to autonomous driving. It is responsible for transferring the desired metrics to vehicle commands for the actuators in the ego-vehicle. Usually, the control system accepts outputs from the local planner, and perception unit and generates the control values, such as acceleration, brake, and steering angle.

Basic control refers to PID controller (See equation3), which consists of three parts, namely the proportional, integral, and derivative gains, and is a widely used controller in the industry.

$$u = k_P e + k_I \int e\, dt + k_D \dot{e} \tag{3}$$

The PID controller is generally well performed in most of the controlling tasks. For self-driving vehicles, the errors and oscillation should be minimized as much as possible. However, PID controller does not track the system in case of rapid state-changing. A model-based controller tackles this problem with an additional controlling part parallel to the feedback

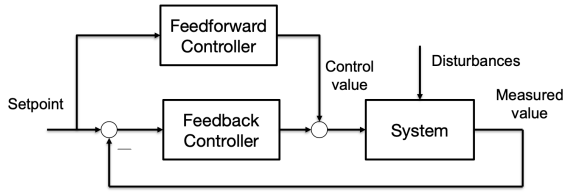

Fig. 6. Improved control with model knowledge. Two degrees of freedom structure allows to design of separate dynamics for setpoint changes and disturbance influences.

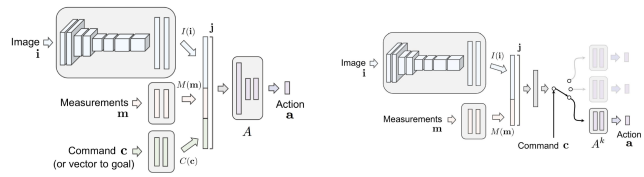

Fig. 7. Conditional imitation learning-Command input (left)][7] and conditional imitation learning-Branched output (right) [27]. In the command input method, commands will be additionally sent into the neural network and flattened to a vector; in the branched output method, several neural networks are responsible for different commands.

controller (See Fig. 6). This additional controller is regarded as the model and considers only the demands of controlled terms, which enhances the reaction time of rapidly changing states by solely computing the controlled term with this feedforward controller.

The model predictive controller is promising. This is a controller that takes the previous states into account and predicts the future state over time. It has much better performance than the aforementioned two controllers.

B. End-to-end approach

The end-to-end approach has become more and more popular in recent years and is regarded as one of the most promising methods in autonomous driving. An end-to-end model is responsible for the whole task of autonomous driving, including perception, prediction, planning, and control. The difference is that the subsystems in modular approaches are not clearly distinguished in end-to-end methods. Therefore, the end-to-end approach is also known as the block-box method, which has a weakness with respect to interpretability. The main approaches for end-to-end learning can be divided into imitation learning and reinforcement learning. The remainder of this section introduces the above two approaches in detail.

1) Imitation learning: Imitation learning is a means of learning and developing new skills from observing these skills performed by an expert. In autonomous driving, the experts are human drivers and the mimicked behavior is the driving commands, e.g., throttle, brake, and steering. Imitation learning is useful when it is easier for the expert to demonstrate the desired behavior rather than specifying the desired policy directly or a reward that would generate such behavior. Behavioral cloning, direct policy learning, and inverse reinforcement learning belong to imitation learning. All imitation learning techniques take advantage of Markov property (See equation 4)– The state of the system at time *t+1* depends only on the state of the system at time *t*, to model the state dynamics $P(s_{t+1|s_t,a_t})$.

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(S_{t+1}|s_t, a_t) \quad (4)$$

A policy is notated by $\pi(a_t|s_t)$ representing a probability for an action a_t given a state s_t. In imitation learning, an expert policy π^* is given to learn the optimal imitator policy π_θ.

The first, as well as a basic method of imitation learning, is behavior cloning. It is a supervised learning, since in the data set $D = ((s_t, a_t), ...)$ inputs and labels and given in pairs. The imitator, a neural network, maps the input s_t to its optimal action a_t. One of the most famous and successful models using behavior cloning is DAVE-2 [6], which we will describe in the next section in detail. Additionally, conditional imitation learning has been proposed in recent years. These architectures (See Fig. 7) integrate the route information based on behavior cloning structure. However, in behavior cloning methods, errors are compounded over time and the ego-vehicle cannot correct itself when it is off the ideal track line.

To overcome the problem that the ego vehicle is not able to correct the trajectory by itself, direct policy learning (DPL) [28] is proposed. DPL is an improved version of behavior cloning. In principle, in DPL, the expert provides feedback on the trajectory generated by the trained imitator and new expert driving data will be recorded. This loop continues until the convergence of ideal π^* is reached.

Inverse reinforcement learning [29] is the latest and most promising method in the end-to-end pipeline. It infers reward function from the demonstration of expert policy when directly learning reward function is more efficient than directly learning policy from the demonstrations. In reinforcement learning, the objective is to find the optimal policy that maximizes the pre-defined reward function. In contrast, in inverse reinforcement learning, the expert policy is regarded as the optimal policy through which the reward function is found.

2) Reinforcement learning: Reinforcement learning is a machine learning paradigm where a system learns to maximize the rewards given to it by acting in an environment [30]. This approach is a separate approach from supervised and unsupervised learning. Compared to supervised learning, which requires training data that might not cover all situations encountered in the real world, reinforcement learning is better at the exploration of unseen situations, which is beneficial in the context of autonomous driving.

The process of reinforcement learning is a loop between the agent (ego-vehicle) and the environment. The car interacts with the environment in a continuous form, at each time step t, the environment reacts to the selected action a_t and presents new state s_{t+1} to the ego-vehicle; the ego-vehicle also receives a new reward r_{t+1} and seeks to maximize rewards over time by choosing action a_{t+1}. Every action taken by the agent results in a reward given to the agent, in the context of autonomous driving, the reward function can be based on speed, distance traveled without crash, the distance towards the goal, collision damage, etc.

Q-learning is a reinforcement learning algorithm, which is

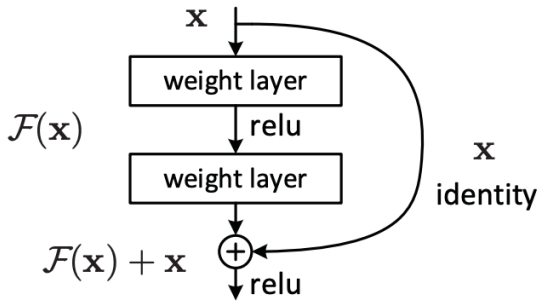

Fig. 8. Structure of a residual block.[31] The residual block information flow between non-connected layers and thus avoid the vanishing gradient problem.

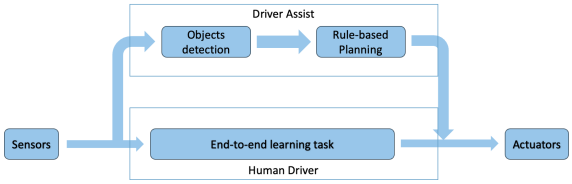

Fig. 9. Pipeline of our autonomous driving system. The end-to-end model approximates the capability of the human driver, who mainly controls the behavior of the ego vehicle. And detection subsystem is responsible for the Automatic Emergency Brake (AEB) as the driver assists.

used to learn each value of the Q-table. The Q-table helps the ego-vehicle to select the best action for each state (See Table 1: An example of Q-table. The horizontal headline is actions, the vertical headline is the states. Values in the table are rewards corresponding to the action and the state.).

	$\uparrow$	$\downarrow$	$\leftarrow$	$\rightarrow$
$State1$	1	3	2	0
$State2$	2	2	5	1
...	...	...	...	...
$StateN$	...	...	...	...

The basic update of the Q-table follows the equation 5 below:

$$Q^{new} \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (5)$$

, where $Q(s_t, a_t)$ is the old value, α is the learning rate, r_t is the reward, and γ is the discount factor. The $\max_a Q(s_{t+1}, a)$ maximizes the expected future reward.

III. Methodologies used in Simulation-Based Autonomous Driving in Crowded City

A. ResNet

ResNet was first introduced in 2016 in the paper "Deep Residual Learning for Image Recognition"[31], in which the residual block was the key innovation. Developed to address the challenges of training very deep neural networks, Residual blocks were introduced as a solution to the vanishing gradient problem. These blocks enable the creation of exceptionally deep neural networks by introducing skip connections, allowing information to flow directly from one layer to another, effectively "residualizing" the error. The core idea behind ResNet is to train the network to learn the residual functions, making it easier to optimize and train extremely deep models. This innovation has had a profound impact on the performance of deep neural networks, enabling the construction of models with hundreds of layers while maintaining or even improving their accuracy. ResNet's remarkable achievements include winning the ImageNet Large Scale Visual Recognition Challenge in 2015, demonstrating the practicality and effectiveness of very deep neural architectures in computer vision tasks.

The ResNet family has several variants, including ResNet 18, ResNet 34, ResNet 50, ResNet 101, ResNet 152, which are named after their number of layers. Among all the variants, ResNet 18 and ResNet 50 are the most popular choices, which can be attributed to their balance between performance and computing costs. ResNet-18 is a relatively shallow variant compared to its counterparts. It consists of 18 layers and is composed of several residual blocks, each containing two convolutional layers. ResNet-18 is a popular choice for tasks where a balance between performance and model size is crucial. It is often used in scenarios where computational resources are limited but high accuracy is still required. Compared to ResNet 18, ResNet 50 has 50 layers and introduces bottleneck blocks within each residual block. These bottleneck blocks reduce the computational cost while maintaining performance. ResNet-50 is known for its excellent trade-off between depth and performance and is commonly used in image classification and object detection tasks.

B. Behavior cloning

The behavior cloning approach was briefly introduced in the last section. Our final autonomous driving system pipeline is shown in Figure 9. In this subsection, we will discuss this approach in detail, especially the DAVE-2 model and our improved behavior cloning model.

One of the successful end-to-end models applied in DAPRA is DAVE-2 (See Fig.10), which was inspired by the pioneering work of ALVINN system. The network architecture of DAVE-2 is mainly based on CNN. The network consists of 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. The convolutional layers were designed to extract the feature of images with the so-called " shared weights", which significantly reduces the number of weights compared to ALVINN (pure fully connected layers network). In the first three convolutional layers, the 5X5 kernels stride 2X2; and 3X3 kernels stride in the last two layers 1X1. Convolutional layers followed by three fully connected layers, output a control value. The FC layers were designed for generating only the steering command.

Inspired by the DAVE-2 model, one of our model's pipelines has a similar architecture with convolutional layers and fully connected layers (See Fig.12). The input size of our model is 3@640X160, which is much bigger than that of DAVE-2 (3@66X200). Therefore, we used 4 fully connected layers with much more weights to avoid information loss during feature extraction in FC layers. Our network architecture starts with 3 convolutional layers with kernels of size 5X5 and stride 2X2, following 2 convolutional layers with kernel size of 3X3 and stride 1X1. Each convolutional layer performs

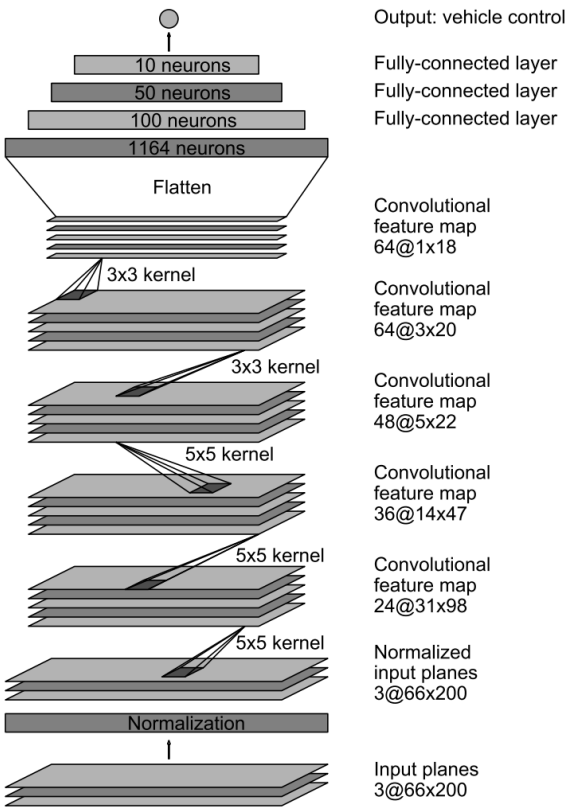

Fig. 10. The network architecture of the DAVE2 system. It uses only convolution layers and fully-connected layers and achieves promising performance.

BatchNormalization[32] to mitigate error accumulation and ELU as an activation function. The output of the fifth convolutional layer will then be flattened to be sent into fully connected layers. The first three FC layers use Dropout to prevent overfitting and ELU as an activation function.

C. YOLO (You Only Look Once)

Another pipeline in our model, which is parallel to the previously mentioned end-to-end pipeline, is the driver-assist pipeline. Our driver assist is a rule-based planner, which accepts the output of the detection algorithm using YOLO [20]. This is the approach that uses features from the entire image to predict each bounding box, meanwhile, predicts all bounding boxes across all classes for an image simultaneously.

The YOLO system consists of 24 convolutional layers followed by 2 fully connected layers, alternating 1X1 convolutional layers. The output of this network is a tensor with a size of $S \times S \times (B*5+C)$, which means the resolution of the output image is $S \times S$, and each pixel has a $B*5+C$-dimensional vector. This vector encodes the information of the size and position of B bounding boxes $[x, y, w, h, P]$ and C classes, where x, y is the coordinate of the center of a bounding box, and w, h is its width and height, P represents the confidence score of the bounding box defined as $Pr(Object)*IOU_{pred}^{truth}$. Therefore, $S \times S \times B$ bounding boxes will be generated. The final output is the bounding boxes with the highest confidence scores for each class. The result of our YOLO detector is

Fig. 11. Our YOLO system detects the objects in each frame. Our YOLO model can recognize classes, such as cars, traffic lights, and buses.

shown in Figure 11.

Our vehicle planner sends a brake command to the ego vehicle to conduct the Automated Emergency Brake (AEB) once the YOLO detector activates the rule that we designed. We defined the rule as:

IF IOU is larger than 0.5, THEN brake = 0.9

The IOU is the abbreviation of *Intersection Over Union*. A pre-defined region (See Fig.11) in the view of the center camera refers to the ground truth of the collision area, where an emergency brake must be immediately activated to avoid actual damage. The detected bounding boxes that represent the cars and buses (Since traffic lights' bounding boxes are relatively smaller than that of cars and buses and do not make sense for emergency brakes in practice, we omitted the bounding boxes of traffic lights.), are likely to intersect our ground truth. In case the ego vehicle is at a safe distance from the detected objects, the intersection of the bounding box and the ground truth is small; If the detected objects are not in the lane of the ego vehicle, there is no intersection at all (bounding boxes are outside the ground truth). Only if the detected objects are close to our ego vehicle, the intersection area will be large. The IOU is calculated by:

$$IOU = \frac{intersection}{Union_{box}^{truth}} \tag{6}$$

Therefore, we bridged the distance to other traffic participants and the detection output to conduct the automated emergency brake as a driver assist.

IV. Experimental and Evaluation

Section III introduces theoretically the methodologies that we used in our autonomous driving system. In this section, we will introduce the practical utilization of our aforementioned methodologies in detail, including the data collection, network architecture, training, and simulation.

A. Data collection

The first step to training our behavior cloning model is selecting the data to use. Our collected data is labeled with throttle (ranges from 0 to 1), brakes(ranges from 0 to 1), and steering(ranges from -1 to 1). To collect the data as expert driving policy, we manipulated the vehicle in the simulator, while the simulator recorded the views of the left, center, and right camera settled in the front of the vehicle and the throttle, brake, and steering at each time step. There are two ways to control the vehicle for data collection: 1) Keyboard control: discrete commands input and inaccurate; 2) Mouse control: continuous input commands, accurately control for fewer outliers. We used the mouse control for better results.

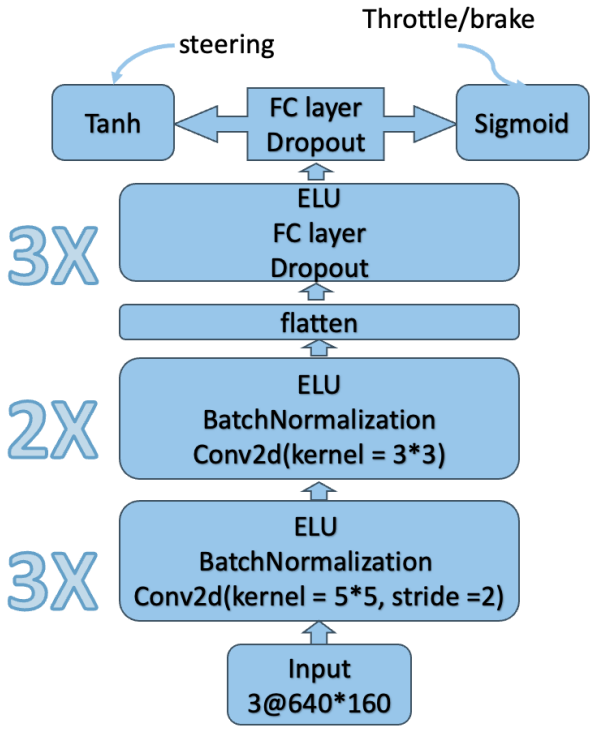

Fig. 12. Structure of our behavior cloning model. There are 5 convolutional layers with different kernels for feature extraction and 4 fully connected layers for the prediction of the vehicle commands.

In the simpler driving environment, in which there are no other traffic participants, at the beginning, we collected around 20 thousand data for training and validation, since the end-to-end neural network is ResNet18, which is a small model. However, after the first training, the training error was low and the validation error was higher, which means the model was overfitting. Thus, in the later training phase, we collected 40 thousand more data to mitigate this problem. We encountered the same problem in the crowded city environment, and we had to collect more data to fix the problem. We found that the number of collected data must match the size of the model, and the capability of a model depends on the complexity of the task —- a complicated task requires a larger model and more data.

B. Network architecture

To deal with the complicated and fast-moving traffic situation, we designed 3 versions of networks based on the vanilla DAVE2 network[6], which was developed by NVIDIA for its end-to-end autonomous driving program.

Since our image input size was much larger than the input size of the vanilla DAVE2 model, we modified its fully connected layer to fit our input size. Moreover, we used batch-normalization[32] after every convolution layer to avoid overfitting. In addition, we used ELU instead of ReLU as the activation function. By doing so we can not only solve the problem of gradients vanishing with negative inputs but also make the model more robust with inputs around zero point. In the end, we treated the steering and the throttle/brake outputs

```
My_model(
  (elu): ELU(alpha=1.0)
  (tanh): Tanh()
  (sigmoid): Sigmoid()
  (conv1): Conv2d(3, 24, kernel_size=(5, 5), stride=(2, 2))
  (conv2): Conv2d(24, 36, kernel_size=(5, 5), stride=(2, 2))
  (conv3): Conv2d(36, 48, kernel_size=(5, 5), stride=(2, 2))
  (conv4): Conv2d(48, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (bn1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn2): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn3): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (do): Dropout(p=0.5, inplace=False)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=60736, out_features=4580, bias=True)
  (fc2): Linear(in_features=4580, out_features=1180, bias=True)
  (fc3): Linear(in_features=1180, out_features=3, bias=True)
)
```

Fig. 13. The network architecture of the smaller version of our custom model. It was modified based on the DAVE2 model but used several methods to avoid overfitting and enhance robustness.

separately. This is because they range in different fields. The steering value ranges from -1 to 1 while throttle and brake only range from 0 to 1. So we used Tanh function to activate steering outputs and used Sigmoid function to activate throttle and brake outputs.

The most miniature version of our model is illustrated in Fig.13. The total parameters are around 90 million. The medium and large version of our model contains around 140 million and 300 million parameters, each with one or two extra fully connected layers. In addition to our self-designed model, we used a YOLO v8 model for object segmentation and to achieve the function of the emergence brake. The YOLO model runs parallelly with our self-designed model.

C. Training details

Since we created a new model based on DAVE-2, which we have introduced in the previous section, we do not have pre-trained weights. Thus, we need to train the weights in our new model to reach the global optima. During the training period, we uploaded our data set to Google Drive to train our model on the Google Colab, which is a cloud computer offering various types of services. We tried different types of GPU to accelerate the training speed. Moreover, hyperparameter tuning is crucial for achieving the global minima. According to different training hardware provided in Google Colab, we tuned several times the hyperparameters, including learning rate, choice of the optimizer, number of epochs, batch size, probability of dropout, etc. to reduce the error of validation. For example, the faster GPU we used, the larger the batch size, and the more number of epochs. Generally, with more batch size and epochs, we got better performance of our model.

In order to avoid overfitting in training, we also used several other techniques listed below.

- **Data processing.** Due to the aforementioned problems, we decided to use a similar data pre-processing method as in the DAVE2[6]: left side and right side images are used for training, while the center image was used for validation. After our data was collected with the simulator, every image was cropped into 3 parts, and their corresponding steering, throttle, and brake information was created from the original text file.
- **Data augmentation.** Moreover, the images are pre-processed with random brightness before feeding into the neuron network to reduce the chance of overfitting.

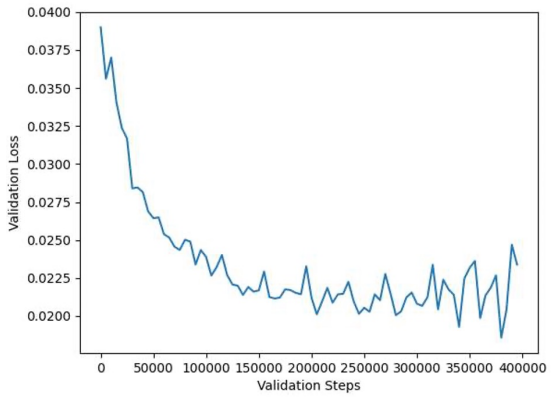

Fig. 14. The validation error. The validation error declines with the increasing number of steps. However, after 200 thousand steps, it starts to incline and oscillate, which means overfitting. Thus, we conducted the technique "early stopping" to prevent overfitting at around 200 thousand steps.

These combined methods enriched our training data as well as improved our model robustness. We doubled our training data and reduced the validation error by about 10% after involving such a method. The validation error is shown in Fig. 14.

D. Simulation

We used the simulator named "Autonomous Driving Simulator" which was developed by the Technical University of Munich. It was realized based on the "Unity" engine and contains a scene with complicated and fast-moving traffic situations such as traffic lights, cars, buses, tracks, etc. This simulator aims to simulate the real-world traffic situation and to test the performance of our model.

In the inference stage, left, center, and right side images are fed into the model parallelly. Then the mean value of the model outputs is calculated as the final output value. This method can prevent the model from outputting extreme values and improve the model's robustness.

In testing, our whole pipeline was able to keep the ego vehicle in its lane and make turns if needed. In addition, we added an automated emergence brake function to our pipeline. Our approach is as follows: [1] Firstly if a vehicle is detected in the center view of our car, then the coordinates of the detection box are returned to our pipeline. [2] Our pipeline will calculate the intersection over the union value between the detection box and our ground truth box. [3] If the IOU value reaches our threshold (0.5 in our pipeline), an automated emergence brake will be performed.

V. Conclusion and Future Works

A. Conclusion

In this paper, we present an autonomous driving pipeline that combines the end-to-end approach as well as the modular approach. We utilized a CNN-based model which was modified from the DAVE2 model for end-to-end controlling and simultaneously implemented YOLO v8[20] model for vehicle detection and automated emergence brake.

As a result, our pipeline can drive ego vehicles and follow the lane in the simulator. Besides, the channel can perform an emergency brake if a crash is estimated.

B. Future works

Although our pipeline can drive our vehicle in the simulator, there still exist several aspects which we can improve.

- **Traffic rules** The first problem of our pipeline is that traffic rules were not followed while driving. We didn't implement the ability to distinguish green and red traffic lights and stop our vehicle in front of the zebra crossing, which could cause severe traffic accidents in real-world scenes. Also, our system was not able to detect the traffic signs such as the vectors on the lane indicating the turning direction. So our vehicle in the simulator was not able to follow the correct route in a crossing.
- **More Training data** Then the second problem of our pipeline is that our CNN-based model was too large, both to train and to use. Even though we have 40k training data, our training data was not able to match the size of our model, so overfitting became inevitable. Additionally, the large model makes the inference stage so time-consuming, that our frames per hour drop to 4 or 5 while running our pipeline in the simulator.
- **More Data Augmentation** To overcome the problem of overfitting and also to enhance the model robustness. We can also apply more data augmentation methods. In our present pipeline, we only implemented the random brightness augmentation. But we can also try methods like image flipping, image scaling, image rotation, and color manipulation.
- **Model Explainability** Our last problem is that our model lacks interpretation. As a black box model, we don't know the exact information flow and the weights of every neuron. Thus we need an explanation of our model and then we can better improve our model. We may use model explaining tools such as LIME (Local Interpretable Model-Agnostic Explanations)[33] and SHAP(SHapley Additive exPlanations)[34] to explain our model and have a better understanding of it.

References

[1] J. McCarthy, "Computer controlled cars," *Obtenido de Universidad de Stan-ford: http://wwwformal. stanford. edu/jmc/progress/cars/cars. html*, 1968.

[2] G. A. Bekey, "Autonomous robots: From biological inspiration to implementation and control," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:60548132

[3] 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, 2020.

[4] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.

[5] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning," *Advances in neural information processing systems*, vol. 18, 2005.

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

[7] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4693–4700.

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

[9] S. Hecker, D. Dai, and L. Van Gool, "End-to-end learning of driving models with surround-view cameras and route planners," in *Proceedings of the european conference on computer vision (eccv)*, 2018, pp. 435–453.

[10] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun, "End-to-end interpretable neural motion planner," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8660–8669.

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

[12] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE intelligent vehicles symposium (IV)*. IEEE, 2011, pp. 163–168.

[13] R. McAllister, Y. Gal, A. Kendall, M. Van Der Wilk, A. Shah, R. Cipolla, and A. Weller, "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 2017.

[14] S. Behere and M. Törngren, "A functional architecture for autonomous driving," in *Proceedings of the First International Workshop on Automotive Software Architecture*, 2015, pp. 3–10.

[15] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 829–846, 2018.

[16] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017.

[17] 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, 2020.

[18] W.-W. Kao, "Integration of gps and dead-reckoning navigation systems," in *Vehicle Navigation and Information Systems Conference, 1991*, vol. 2. IEEE, 1991, pp. 635–643.

[19] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments." in *Robotics: science and systems*, vol. 4, no. Citeseer. Atlanta, GA, USA, 2007, p. 1.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[21] H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Sujiwo, L. Y. Morales, N. Akai, T. Tomizawa, and S. Kato, "Open source integrated planner for autonomous navigation in highly dynamic environments," *Journal of Robotics and Mechatronics*, vol. 29, no. 4, pp. 668–684, 2017.

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

[23] D. Van Vliet, "Improved shortest path algorithms for transport networks," *Transportation Research*, vol. 12, no. 1, pp. 7–20, 1978.

[24] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner, "Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm," *Journal of Experimental Algorithmics (JEA)*, vol. 15, pp. 2–1, 2010.

[25] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck, "Phast: Hardware-accelerated shortest path trees," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 940–952, 2013.

[26] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

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

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

[29] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International conference on machine learning*. PMLR, 2016, pp. 49–58.

[30] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[33] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[34] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.