Real-Time Object Tracking and Path Following for Autonomous Vehicles Using Carla Simulator

Jiarun ZHU
12210212

Qijun CAI
12210214

Jiayang HE
12213023

***Abstract*—We aimed task is 3. CARLA Simulator:Topic 3 : Object Tracking.The ability for autonomous vehicles to track and follow other vehicles in dynamic environments is crucial for ensuring safe and efficient navigation. This research aims to develop a real-time object tracking and path following system using the Carla simulator. The proposed system leverages camera-based object tracking to identify and follow a leading vehicle, enabling the controlled vehicle to replicate its movements, including acceleration, deceleration, turning, and stopping.the github link:https://github.com/Ray0v0/ML_Project_ObjectTracking**

***Index Terms*—Object Tracking, Path Following, Carla**

I. Background and Significance

A. The Rapid Developing Autonomous Driving Technology

With the rapid development of the automotive industry in China, intelligent vehicle assistance systems have achieved large-scale pre-installed mass production applications, and autonomous driving systems are actively being promoted for commercial exploration. According to statistics from the China Intelligent and Connected Vehicle Industry Innovation Alliance[1], in 2023, sales of intelligent and connected passenger vehicles equipped with combined assistance driving functions reached 9.953 million units, with a market penetration rate of 47.3%, which clearly shows that the development of this industry has entered the expressway of rapid growth, and autonomous driving technology becomes the cutting-edge development direction of the automotive industry, which holds significant importance for enhancing traffic safety and travel convenience.

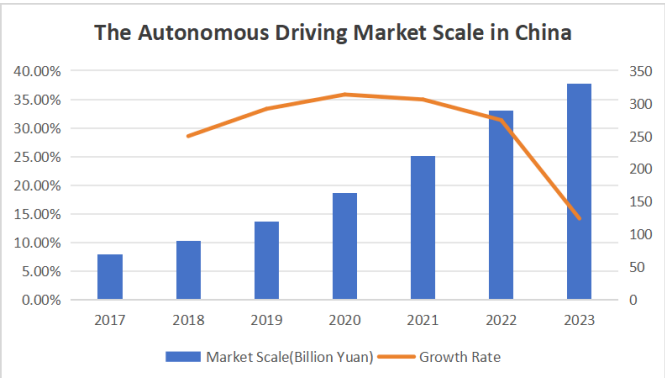


Fig. 1: The Autonomous Driving Market Scale in China from 2017 to 2023, collected and analysised by *www.huaon.com* [2]

B. Introduction to Object Tracking in Autonomous Driving

In the realm of autonomous driving, object tracking is a pivotal task that holds immense importance for the safe and efficient operation of autonomous vehicles. This task involves the continuous monitoring and prediction of the motion trajectories of multiple objects within the vehicle's surroundings. These objects can include pedestrians, other vehicles, bicycles, and any other potential entities that may appear on the road.

In this research project, the image recognition, path planning, and vehicle control algorithms implemented in our vehicle tracking system are closely related to the cutting-edge technologies in autonomous driving. Through the realization of the vehicle tracking task, we aim to explore the frontier technologies in autonomous driving and investigate methods to enhance the safety and reliability of autonomous driving systems.

C. Technologies May Involved in Object Tracking

- Sensor Fusion: Sensor fusion integrates data from multiple sensors to provide a comprehensive and accurate understanding of the environment.

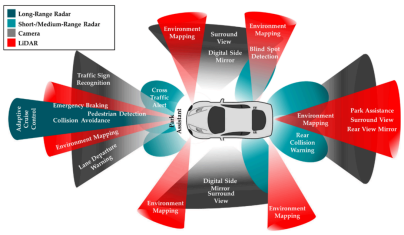


Fig. 2: In autonomous driving, different sensors complete their respective environmental sensing tasks according to their own characteristics and provide multi-dimensional fusion information to the decision-making system. Image from [3].

- Object Detection: Object Detection uses machine learning techniques to identify and classify objects within digital images.

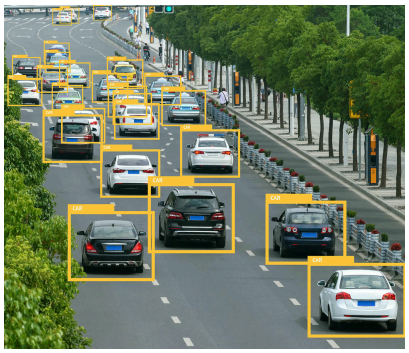


Fig. 3: Image recognition using algorithms like yolo and RCNN

- Path Planning and Control Algorithm: Path planning and control algorithms determine the optimal route and control inputs for a vehicle to achieve specific location and velocity safely and efficiently.

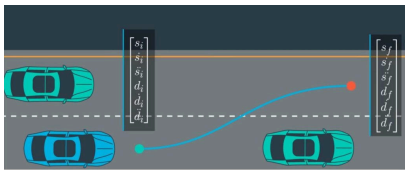


Fig. 4: Path planning demonstration. Image from *https://sanjiv-valsan.com/autonomous-driving-path-planning/*

- Reinforcement Learning: Reinforcement learning, abbreviated as RL, is a field of machine learning that emphasizes how to act based on the environment to maximize expected benefits.

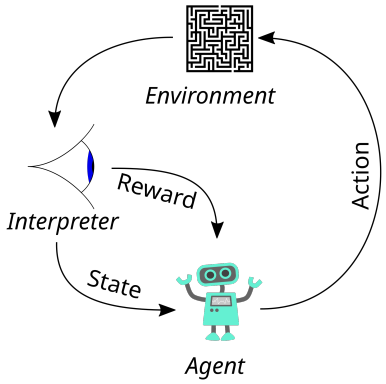


Fig. 5: The typical framework of reinforcement learning: An agent takes an action in an environment, which the environment translates into a reward and a state representation, which are then fed back to the agent. Image from Wikipedia.

D. The Carla Simulator

CARLA (Car Learning to Act)[4] represents a state-of-the-art open-source simulator specifically designed for autonomous driving research. This simulator provides a high-fidelity virtual environment that closely mimics real-world driving conditions, making it an ideal platform for our vehicle tracking system development.

The selection of CARLA for our vehicle tracking system development enables comprehensive testing in a controlled environment while facilitating large-scale data collection and algorithm validation under various conditions. The simulator's architecture supports rapid iteration of reinforcement learning implementations, effectively bridging the gap between simulation and real-world deployment.

The methodologies developed within CARLA can be efficiently transferred to real-world autonomous vehicles, allowing for risk-free validation of tracking algorithms and systematic optimization of system performance. Through Carla's simulation capabilities, our research aims to advance autonomous vehicle tracking systems with direct applications to real-world implementation.

II. Analysis of Current Research Status

A. The Two Approaches

Autonomous driving systems can be broadly categorized into two distinct approaches: the modular approach and the end-to-end approach. [5]

a) ***The Modular Approach***:
The modular approach relies on a structured pipeline where functionalities such as perception, prediction, and planning are developed as separate components and integrated into the system.[6] For instance, the planning module uses the outputs of upstream components, such as detected objects or predicted trajectories, to generate control commands like steering and acceleration.

While this approach benefits from clear division of tasks and interpretability, its reliance on rule-based designs often struggles to handle the immense variability of real-world driving scenarios. As a result, there is a growing shift toward incorporating data-driven methods to enhance the adaptability of modular systems.

b) ***The End-to-End Approach***:
In contrast, the end-to-end approach seeks to simplify the pipeline by treating the autonomous driving system as a fully differentiable program. This method processes raw sensor data directly and produces driving actions or plans as output. Unlike the modular approach, which passes explicit outputs between components, the end-to-end system propagates feature representations across its architecture. [7] Through backpropagation, the entire system is optimized jointly to minimize loss, such as improving planning accuracy.

This paradigm has the potential to achieve superior performance by leveraging large-scale data and optimizing all tasks globally, but it can be less interpretable compared to modular systems.

We created a table to present the differences between the two approaches more clearly.

Comparison Between the Two Approaches

	Modular	End-to-End

Adaptability	Limited	High
Interpretability	High	Low
Development Effort	Exhaustive Expert Rules	Abundant Data with Training
Deployment Readiness	More Mature Widely Deployed	Emerging Under Research

By leveraging large-scale data and advanced deep learning techniques, end-to-end systems offer the potential to simplify the development pipeline, improve adaptability to complex driving scenarios, and achieve superior performance through holistic optimization, end-to-end autonomous driving has become a focal point of research and development for many leading automotive companies, as it represents a promising direction for the future of intelligent vehicles.

B. Analysis of Existing Literatures

Traditional Models:

[8] is noted for performing well across various rankings, partly because it effectively integrates 2D and 3D detection in a multimodal MOT task. The motive considers the complementary characteristics of the two types of observations—3D detection provides depth information but is less sensitive to distant objects, whereas 2D detection does not provide depth information but can detect distant targets.

[9] introduced two motion models, CTRA and Bicycle, suitable for cars, pedestrians, electric bikes, and motorcycles, respectively. These motion models are embedded into a Kalman filter to perform predictions and updates; different standards are applied for different categories of objects, and three types of similarity measures are used, which increases efficiency due to the traditional method employed.

[10] deconstructs the existing tracking-by-detection paradigm under the MOT algorithm into four modules: preprocessing, data association, motion modeling, and lifecycle management. It offers comparisons and analyses of common methods in these modules and suggests improvements.

[11] models the uncertainties in predictions and observations, using uncertainty-aware Mahalanobis distance to construct frame-to-frame matching costs, as opposed to using the intersection over union (IoU) of 3D boxes like in AB3DMOT.

Machine Learning Models:

At the beginning of the project ，we first reviewed an extensive survey on autonomous driving technologies [5], which indicated that object tracking primarily relies on machine learning [12] and deep learning models [13] to process and analyze sensor data. This enables continuous tracking of specific objects in dynamic environments. A common method employed is the use of convolutional neural networks (CNNs) to identify and track objects in video frames. Furthermore, incorporating recurrent neural networks (RNNs) or long short-term memory networks (LSTMs) improves the accuracy of motion predictions, effectively preventing potential conflicts [14].

By examining the reference project architecture [15], we realized the importance of using a monocular RGB camera for real-time object detection and basic image segmentation. This project also introduced a dual-task neural network that can perform both object detection and strategic image segmentation, which inspired us to explore how to handle tracking conflicts, such as when the target object is occluded.

In [16], 3D object tracking was implemented, providing more options for our project approach. However, reproducing the experiment showed limitations in effectively handling lost objects.

[17] proposes a novel end-to-end motion prediction framework (mmTransformer) for multimodal motion prediction.The model effectively mitigates the complexity of motion prediction while ensuring the multimodal outputs.

[[18]] introduced a new integrated framework that consolidates full-stack driving tasks into a single network. It is meticulously designed to utilize the strengths of each module and provide complementary feature abstraction for object interaction from a global perspective. Tasks are communicated through a unified query interface to promote planning. This framework, however, requires substantial computational power, and we have yet to experiment with whether this framework can be successfully replicated.

[19] introduces a novel online deep learning framework for 3D object detection and tracking using a monocular camera ◦ The framework can estimate the complete 3D information from a sequence of 2D images and associate the objects over time. By obtaining continuous frames from the front camera, our network robustly tracks the 3D bounding boxes for each observation and provides its location P with orientation θ, dimension D and 2D projection of its 3D center c.

This comparison sheds light on the different models and performance analyses based on task-specific applications in the existing literature:

Comparison Between the Different Existing Literature

Project Name	Technology Used	Key Features
EagerMOT [8]	Fusion of 2D and 3D Detection	Combines depth information from 3D detection with the broader range detection capabilities of 2D systems to capture distant objects
Ploy-MOT [8]	Fusion of 2D and 3D Detection,Kalman Filter with CTRA and Bicycle Motion Models	Utilizes two motion models for different vehicle types,applies different criteria for efficient traditional method-based MOT
SimpleTrack [10]	Tracking-by-detection Decomposition	Breaks down the MOT algorithm into four modules: preprocessing, data association, motion mod-

		eling, and lifecycle management. Analyzes common methods in each module and suggests improvements
Probabilistic Matching [11]	Uncertainty Modeling in Tracking	Models the uncertainty in predictions and observations, using uncertainty-aware Mahalanobis distance for frame-to-frame matching costs instead of direct IoU of 3D boxes like in AB3DMOT
Machine Learning Based Object Tracking [12], [13], [14]	CNN+LSTM	High accuracy in object recognition and motion prediction
Real-time Object Detection and Image Segmentation [15]	Dual-task Neural Network	Real-time response, combines object detection and image segmentation
3D Object Tracking [16]	3D Reconstruction + Long-term Tracking	recognize 3D object rather than 2D in camera
Multimodal Motion Prediction [17]	Stacked Transformers	Handles multi-source information, reduces prediction complexity,but high cost
Planning-oriented Autonomous Driving [18]	Full Stack Transformer	A brand new structure,cast a series of tasks,no muti-tasks

C. Challenges in Autonomous Driving: A Comprehensive Overview

a) *Human Language as Input:*

Autonomous driving leverages both visual perception and intrinsic knowledge, such as traffic rules and route requirements, to guide causal behavior. Significant progress has been made in robotics and indoor navigation by integrating natural language as granular instructions for controlling visual-motion agents. Technologies like CLIP-MC and LM Nav utilize the CLIP architecture to extract linguistic knowledge from instructions and visual features from images, showcasing the advantages of pretrained visual-language models for complex multimodal tasks. However, the application of large language models like GPT-3 or instructional models like ChatGPT in autonomous driving remains uncertain, with modern LLMs offering potential for handling complex language instructions but facing challenges in interaction modalities due to their lengthy inference times and instability.

b) *World Models and Model-based RL:*

World models and model-based reinforcement learning are crucial for making predictive and strategic decisions in reinforcement learning, especially within end-to-end autonomous driving systems. World models provide an explicit representation of the environment, which is particularly useful given the high complexity and dynamic nature of driving environments. Model-based RL enhances sample efficiency by allowing interaction with a learned model of the world, reducing reliance on actual environment interaction.

c) *Multitask Learning:*

End-to-end multitask learning has proven effective in enhancing performance and interpretability of autonomous driving models. However, finding the optimal combination of auxiliary tasks and their appropriate loss weighting to achieve best performance remains a challenge. Additionally, building large-scale, well-annotated datasets with diverse alignments is crucial.

d) *Knowledge Distillation:*

Despite extensive efforts in designing robust expert models and transferring knowledge from teachers to students of various levels, the teacher-student paradigm still faces efficiency issues. Significant performance gaps have been observed between vision-motion networks and their privileged counterparts, such as real-world access to traffic lights, posing challenges in feature extraction and potential causal confusion.

e) *Interpretability:*

Interpretability plays a crucial role in testing, debugging, and improving systems, offering performance assurances from a societal perspective, increasing user trust, and facilitating public acceptance. Achieving interpretability in end-to-end models, often referred to as "black boxes," is challenging. Techniques like attention visualization offer some interpretability by revealing the model's focus areas, though their reliability and practical utility are limited.

f) *Uncertainty Modeling:*

Modeling uncertainty is a quantitative approach to interpreting the reliability of model outputs. Designers and users need to identify uncertain situations to make necessary improvements or interventions, as planning results are not always accurate or optimal.

These challenges underscore the need for further research to refine models and methodologies in the rapidly evolving field of autonomous driving, ensuring safety, efficiency, and user trust in these advanced systems.

III. Completed Work

We have built a vehicle tracking framework based on the Carla simulator, dividing the vehicle tracking task into two subtasks: perceiver and controller.

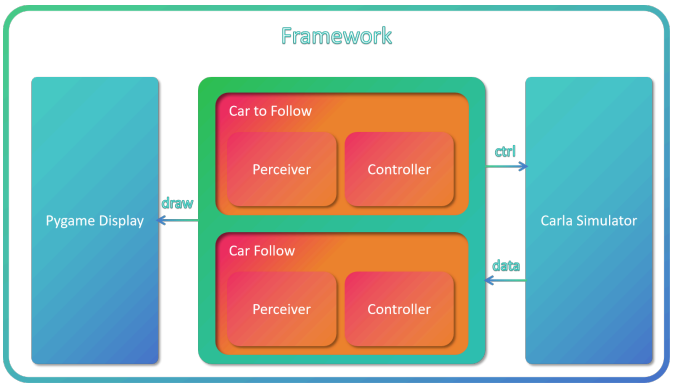


Fig. 6: general work

- Based on this framework, we proposed and tried various implementations of the sub-tasks, including:
- Implementing a real-time vehicle detection and positioning algorithm based on YOLO-v8 and Bayesian posterior.
- Implementing a PID-based vehicle acceleration and steering control scheme.
- Implementing the Distance-Angle-Fusion vehicle control algorithm.
- Implementing historical path tracking control logic based on DAF.
- Implementing navigation control logic based on the navigation data provided by Carla.
- Using the DDPG reinforcement learning algorithm to train an autonomous driving model (training performance was poor, so it was abandoned).
- Implementing a vehicle collision prediction model based on CNN-LSTMs.

The general structure of the entire project is shown in the figure below:

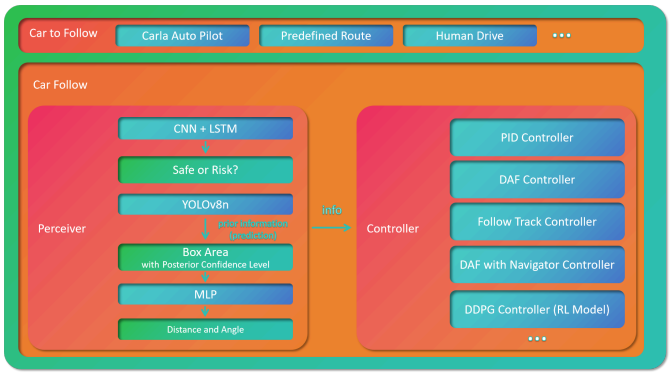


Fig. 7: general work

A. Environment Setup and Control

We successfully deployed Carla simulator version 0.9.8 along with all necessary dependencies and configured the simulation environment for autonomous vehicle control scenarios. We also implemented basic vehicle control functionalities.

B. Data Collection System

In terms of data collection, we established a fundamental sensor data collection framework and implemented video data capture from Carla's camera sensors. Additionally, we developed a pipeline to store the camera feed as structured video data. Finally, we verified the data's integrity and ensured its accessibility for subsequent processing.

```
# 生成后车传感器
bp_collision_sensor = blueprint_library.find('sensor.other.collision')
collision_sensor = world.spawn_actor(
    bp_collision_sensor,
    carla.Transform(),
    attach_to=vehicle_follow
)
sensor_list.append(collision_sensor)
collision_sensor.listen(lambda data: evaluator.collision_occured([velocity_follow]))

bp_camera_rgb = blueprint_library.find('sensor.camera.rgb')
bp_camera_rgb.set_attribute('image_size_x', '800')
bp_camera_rgb.set_attribute('image_size_y', '600')
bp_camera_rgb.set_attribute('fov', '90')
camera_rgb = world.spawn_actor(
    bp_camera_rgb,
    carla.Transform(carla.Location(x=1.5, z=1.4, y=0), carla.Rotation(pitch=0)),
    attach_to=vehicle_follow
)
sensor_list.append(camera_rgb)
```

Fig. 8: Read real-time image data from the Carla world

```
# 后车感知
info_follow, box = perceiver_follow.perceive(
    velocity_follow=velocity_follow,
    pose_follow=pose_follow,
    map=map,
    camera_image=image_rgb  # 传入数组的副本

    # velocity_follow=velocity_follow,
    # pose_follow=pose_follow,
    # velocity_to_follow=velocity_to_follow,
    # pose_to_follow=pose_to_follow,
    # map=None
)
```

Fig. 9: Pass to the rear vehicle perception processor

C. Model Choice

The structure of YOLOv8 is as follows：

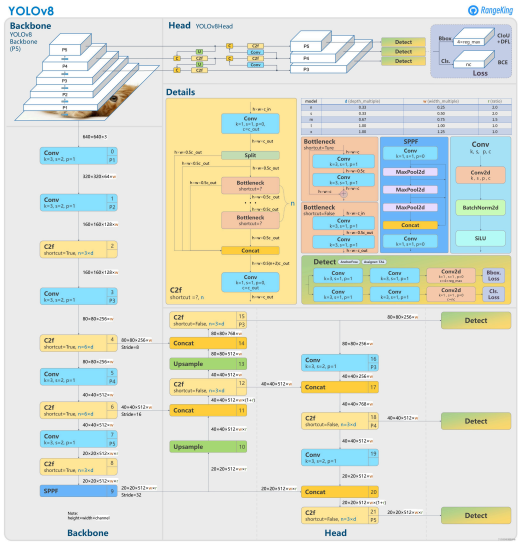


Fig. 10: yolov8 structure

a research comparing faster R-CNN, SSD, YOLOv4, YOLOv5 and YOLOv8 suggests that YOLOv8 outperforms all of the other models.[20]

Method	Backbone	Model Size	Accuracy for mAP 0.5	FPS	Average Inference Time	Batch Size	Input Resolution
Faster R-CNN	ResNet101	72.8 MB	100%	19	0.0528 seconds	8	600x600
SSD	Mobilenet-v2	9.2 MB	86%	433	0.00023 seconds	8	320x320
YOLOv4	CSPDarknet53	162.2 MB	96%	1083	0.0009 seconds	64	416x416
YOLOv4-Tiny	CSPDarknet53-Tiny	22.5 MB	97%	1015	0.0009 seconds	64	416x416
YOLOv5 small	yolov5s	14.1 MB	99%	1002	0.001 seconds	16	416x416
YOLOv8 nano	EfficientNet	6.2 MB	100%	1100	0.0009 seconds	16	640x640

Figure 7: Comparison between Mainstreaam Models

Fig. 11: Comparison between Mainstreaam Models.

After reviewing the results above, Our choice for the baseline is YOLOv8n. Not only it is open-source and easy to modify, but also it demonstrated proficiency in detection and recognition.

D. Model Improvement

- In practical tests, it was found that directly using the YOLOv8n pretrained model had poor recognition performance. We trained a MLP to process the YOLOv8n output (from box to distance and angle) to obtain more accurate information.

The structure of the regression model is shown in the figure below:

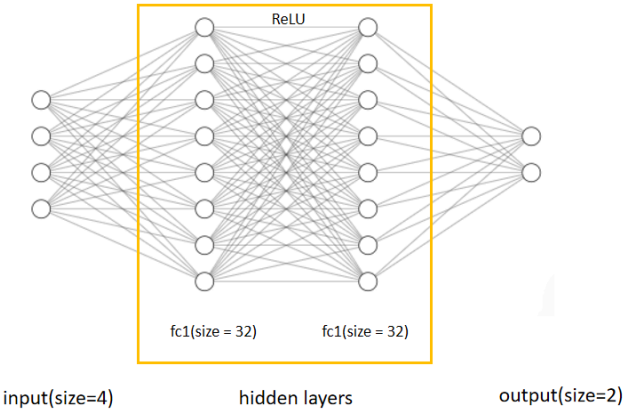


Fig. 12: Regression Model Structure.

- The improved model file: box_to_distance_and_angle_model.pth

E. Perceiver Design

By interacting with Carla in a synchronized mode, we can acquire real-time image data. This image data is then fed into the model for processing, enabling the recognition of the front vehicle. The model outputs include the addition of segmentation bounding boxes on the image, along with estimated information such as relative distance and relative angle.

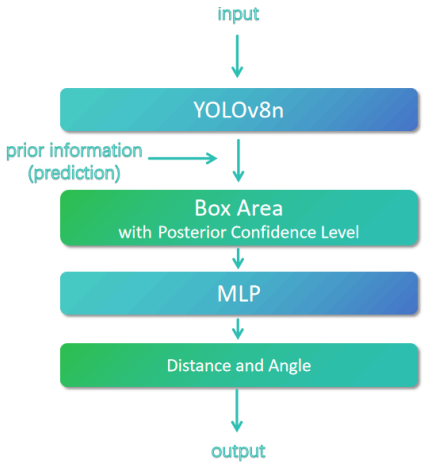


Fig. 13: Perceiver Design Graph

F. The Vehicle Control Algorithm Design

In this task, we considered two approaches:

a) *Tracking the trajectory based on the historical positions of the front vehicle:*

- This approach involves keeping track of the front vehicle's past positions and using that information to predict its future trajectory, aiding in more accurate tracking.

b) *Directly following the vehicle based on the current relative distance and angle:*

- In this approach, we directly use the current relative distance and angle between the vehicle and the front vehicle to adjust the movement, allowing for more immediate and reactive tracking.

Detailed information is below:

1.PID-based Longitudinal and Lateral Vehicle Control:

The mathematical expression of the PID control algorithm is as follows: $\text{Output} = P + I + D$

The proportional, integral, and derivative components are as follows:

$$P = K_p \cdot \text{error} \tag{1}$$

$$I = K_i \cdot \sum \text{error} \tag{2}$$

$$D = K_d \cdot \partial \frac{\text{error}}{\partial t} \tag{3}$$

The final control output is the weighted sum of the three components.

$$\text{Output} = K_p \cdot \text{error} + K_i \cdot \sum \text{error} + K_d \cdot \partial \frac{\text{error}}{\partial t} \tag{4}$$

In this task, we used PID to implement a basic controller for smooth vehicle control.

2.DAF(Distance-Angle-Fusion) Algorithm:

DAF is an autonomous driving control algorithm based on the fusion of distance and angle. It utilizes the relative position, speed, and angle information of the vehicle to adjust throttle, brake, and steering signals in real-time. This ensures the vehicle maintains a safe distance from the front vehicle while minimizing the risk of collision.

Based on this approach, we designed three types of DAF controllers:

- DAFController: A basic controller based on the DAF algorithm.This controller uses the DAF algorithm to adjust the vehicle's speed and steering, ensuring safe distance and avoiding collisions.
- DAFWithNavigatorController： This controller ensures that when the front vehicle follows the route, it uses Carla's navigation package to establish a navigation path between the front and rear vehicle coordinates. The rear vehicle then follows this navigation path. As a result, the rear vehicle adheres to traffic rules (it won't drive in the wrong direction, won't run red lights, and even if all traffic lights are set to green, following the front vehicle can still be challenging).
- DAFFollowTrackController： This controller stores the historical driving trajectory of the front vehicle based on sensor input. It controls the vehicle by following the tracked trajectory of the front vehicle.

```
def predict_control(self, info):  4 个用法 (4 个动态)
    if info.trust_worth is False:
        return carla.VehicleControl(throttle=self.throttle, steer=self.steer, brake=self.brake)

    distance = info.distance
    angle_between_cars = info.angle

    print("distance: %.2f \tangle: %.2f" % (distance, angle_between_cars))
```

```
def predict_control(self, info):  4 个用法 (4 个动态)

    if self.dao is None:
        self.dao = GlobalRoutePlannerDAO(info.map, 1)
        self.nav = GlobalRoutePlanner(self.dao)
        self.nav.setup()

    nav_points = self.nav.trace_route(info.pose_follow.location, info.pose_to_follow.location)

    nav_way_points = []
    for nav_point in nav_points:
        nav_way_points.append(nav_point[0])

    if self.debug_mode:
        draw_waypoints(self.world, nav_way_points)
```

```
def predict_control(self, info):  4 个用法 (4 个动态)
    self.checkpoints.append(info.pose_to_follow) # 根据导航获取前车绝对位置信息

    while len(self.checkpoints) > 1:
        if PoseManager.get_distance(info.pose_follow, self.checkpoints[0]) < self.check_distance:
            self.checkpoints.pop(0)
        else:
            break
```

Fig. 14: the difference between the 3 Controllers,top to bottom : DAF,Navigator,Track

We will compare the implementation results of different methods in the evaluation.

IV. Results Evaluation

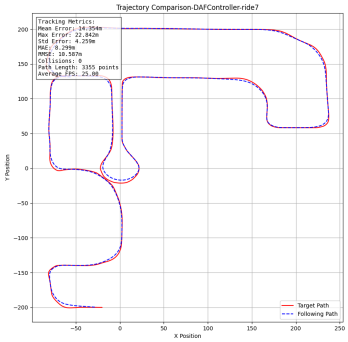
To evaluate the effectiveness of our designed algorithms and models, we will have the front vehicle autonomously drive along a specified route, while the rear vehicle performs the car-following task using our designed algorithm.

- There is a database of 20 different drives with varying difficulty: 10 easy, and 10 challenging rides. The rides are about 1 minute long on average. We used the Tesla Model 3 and manually drove it around the city using connected steering wheel and pedals Hama uRage GripZ. As the car was driven, its location and orientation were recorded at every frame. This way, we have collected a database of 20 different drives with varying difficulty (in a sense of chasing the car). We used this database to evaluate the autonomous driving system. At the beginning of the following experiments, a chasing car was placed half of a meter behind the chased car. Then, we updated every frame the location and orientation of the chased car based on the saved coordinates in the dataset. Meanwhile, the chasing vehicle was driven autonomously and trying to maintain a desired distance from the pursued car.
- Here, we choose some typical routes and plot the driving trajectories of both vehicles on a coordinate graph from a top-down view. We use the following metrics to evaluate the performance of the current algorithm and model: Mean Error, Max Error, Std Error, MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and the number of collisions.

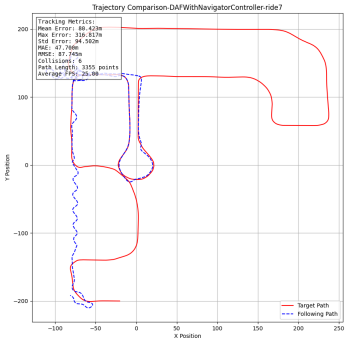
A. *using the **ride7.p** path to evaluate*

ride7.p features a long path with **rapid acceleration** of the front vehicle and many **sharp turns**, making it suitable for testing the car-following performance and its limits.

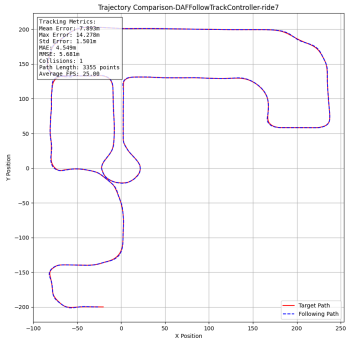
a) *DAFController with Perceiver:*

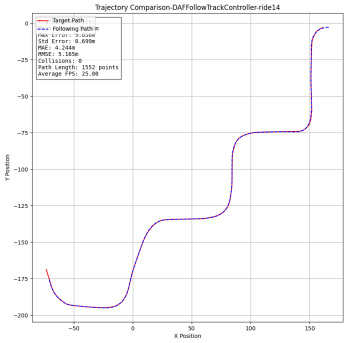


b) *DAFWithNavigatorController with Navigation:*



c) *DAFFollowTrackController with Map data:*





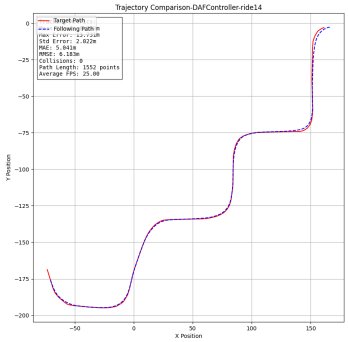
	MAE(M)	RMSE(M)	Collisions
DAF	8.299	10.587	0
Navigator	47.700	87.745	6
Track	4.549	5.681	1

	MAE(M)	RMSE(M)	Collisions
DAF	5.051	6.183	0
Navigator	79.821	105.371	712
Track	4.224	5.165	0

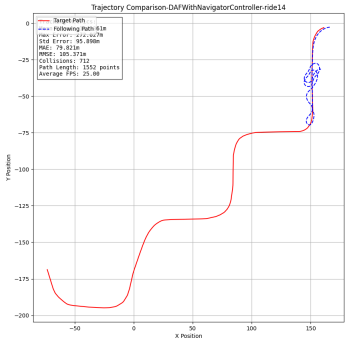
*B. using the **ride14.p** path to evaluate*

The ride14.p path is characterized by many consecutive sharp turns, making it suitable for testing the limits of car-following performance.

a) *DAFController with Perceiver:*



b) *DAFWithNavigatorController with Navigation:*



c) *DAFFollowTrackController with Map data:*

Result Analysis：

- We can consider the **track** method as the **ideal car-following state**, as it follows the path of the front vehicle closely, resulting in small MAE and RMSE values, although one collision occurred. On the other hand, the three evaluation metrics obtained using the **Navigator** method are higher, and, in fact, the images show that the car-following task was not successfully completed throughout.
- When comparing our designed **DAF+Perceiver** with the above two methods, we found that the MAE and RMSE values of the **DAF** algorithm are close to those of the ideal track car-following, indicating that our model and algorithm perform well. Additionally, the **DAF** method successfully **avoided collisions** by controlling the distance, and multiple experiments showed good results.

V. Vehicle_Collision_Prediction Model (Advanced Work)

Based on the CNN+LSTM architecture in the [14] , we built and trained a classification model for vehicle collision prediction using the dataset provided in the project. The model has been integrated into the perception system.

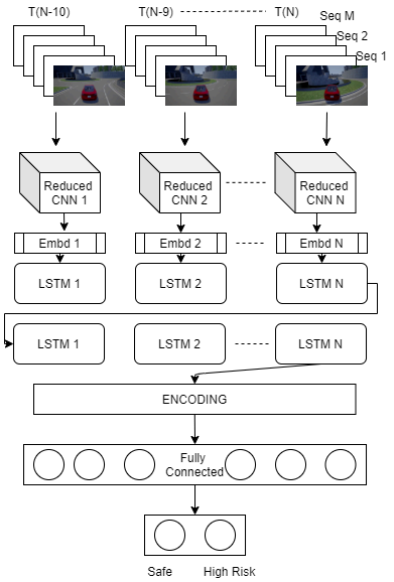


Fig. 15: CNN+LSTMs Network

After different experiments with various architectures and hyper parameters, the final model consists of a very diluted GoogleNet like network with two Inception modules for the CNN part and contains two LSTM layers with 32 hidden units each. The CNN is then wrapped into a time distributed layer at each step to make it available for each of the time steps. By this way, we don't have to learn the image 8 different time. Time distributed layer makes the same network avaialble for all the time steps, reducing the time, complexity and size of the network.

At the end of LSTM layers, we also have some fully connected layers with dropout to learn the classification task. Model Architecture.

This model outputs probabilities for **"safe"** and **"risk"** based on the current image input, and we select the higher value as the prediction for the current situation. The model has been loaded into the sensor for predicting the **likelihood** of a collision during interactions with the environment.

```
r = self.network.predict(np_image_seqs)
self.stat = ['safe', 'collision'][np.argmax(r, axis: 1)[0]]  # 获取预测结果
```

a) *Show Results:*

After training the model, we first use the provided valid set to evaluate the model's performance:

Model Evaluation

	origin->	**training finished**
loss	0.69425404071->	0.1850231587
accuracy	0.55147057->	0.93382352
val_loss	0.69675540924->	0.07657913863
val_accuracy	0.5->	0.90625

The accuracy reached over 93%. To enable practical use, we integrated the model into the perceiver. We selected the "ride7.p" route for experimental demonstration, as it features rapid acceleration and many sharp turns, making it suitable for testing the limits of following the front vehicle and effectively showcasing the collision prediction. We recorded a demo video demonstrating the collision prediction, and in the report, the results are shown in images with the current safety status displayed in the top-left corner.

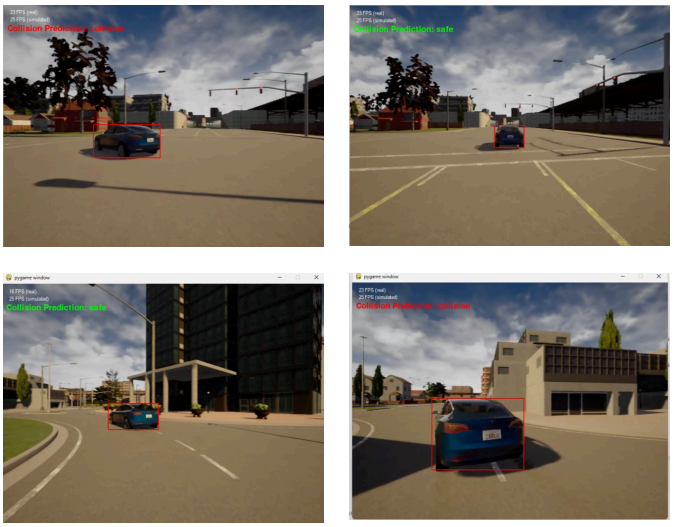


Fig. 16: Vehicle_Collision_Prediction results show1(chasing car)

- As shown in **Fig. 14**, during the car-following task on the **ride7** path, the model was able to predict collisions. This is particularly evident when the front vehicle makes sharp turns, and the ego vehicle is traveling at a high speed with too close a following distance.

Fig. 17: Vehicle_Collision_Prediction results show2(obstacles)

- **Fig. 15** shows the prediction when the vehicle is about to collide with an obstacle.

Due to the large number of parameters in the model：

```
================================================================================
Total params: 14,814,338
Trainable params: 14,811,202
Non-trainable params: 3,136
--------------------------------------------------------------------------------
```

the computational power required for predictions is relatively high. When running locally, frame drops may occur. To address this, we chose asynchronous prediction, performing a prediction approximately every 50 frames (around 2 seconds). In most test rounds, this approach worked well.

VI. Future Work

A. Some attempts (DDQN for RL)

- using the DDQN architecture to train a reinforcement learning model for executing controller tasks.The DDQN

model architecture has been built, and it is a deep reinforcement learning model. You can see it in **Controller/DDPG.py** However, due to limited time and slow training, it could not be showcased in this report. If given the opportunity, we plan to complete it in the future.

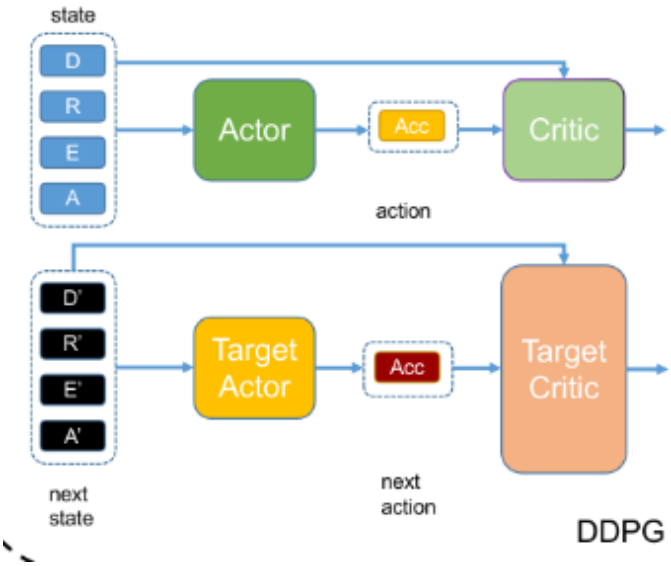


Fig. 18: DDQN Network

In the DDPG algorithm:

- Actor Network: Generates continuous, deterministic actions.

```
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim, action_bound):
        super(Actor, self).__init__()
        self.action_bound = torch.tensor(action_bound).to(device)

        # layer
        self.layer_1 = nn.Linear(state_dim,  out_features: 100)
        self.layer_2 = nn.Linear( in_features: 100,  out_features: 100)
        self.output = nn.Linear( in_features: 100, action_dim)
```

- Critic Network: Evaluates the quality of the actions by calculating the Q-value, which guides the Actor Network.

```
class Critic(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(Critic, self).__init__()

        # layer
        self.layer_1 = nn.Linear(state_dim+action_dim,  out_features: 100)
        self.layer_2 = nn.Linear( in_features: 100,  out_features: 100)
        self.output = nn.Linear( in_features: 100,  out_features: 1)
```

- Target Networks: Used to stabilize the training process and prevent the parameters from updating too quickly.

```
# Deep Deterministic Policy Gradient
class DDPG(object):
    def __init__(self, state_dim, action_dim, action_bound, replacement, memory_capacticy=10000, gamma=0.99, lr_a=0.00001,
                 lr_c=0.0001, batch_size=64):
        super(DDPG, self).__init__()
        self.state_dim = state_dim
        self.action_dim = action_dim
        self.memory_capacticy = memory_capacticy
        self.replacement = replacement
        self.t_replace_counter = 0
        self.gamma = gamma
        self.lr_a = lr_a
        self.lr_c = lr_c
        self.batch_size = batch_size
        self.episode = 0
        self.best_loss = 0

        # 记忆库
        self.memory = np.zeros((memory_capacticy, state_dim * 2 + action_dim + 1))
        self.pointer = 0
        # 定义 Actor 网络
        self.actor = Actor(state_dim, action_dim, action_bound).to(device)
        self.actor_target = Actor(state_dim, action_dim, action_bound).to(device)
        # 定义 Critic 网络
        self.critic = Critic(state_dim, action_dim).to(device)
        self.critic_target = Critic(state_dim, action_dim).to(device)
        # 定义优化器
        self.aopt = torch.optim.Adam(self.actor.parameters(), lr=lr_a)
        self.copt = torch.optim.Adam(self.critic.parameters(), lr=lr_c)
        # 选取损失函数
        self.mse_loss = nn.MSELoss()
```

B. Hardware Limitations

To better validate the effectiveness of the framework we designed, we should test the project on a platform with more computational power.

C. Model Optimization

At the same time, we should attempt to optimize the model architecture and parameters to improve both the inference speed and performance, in order to better accomplish the car-following task.

VII. Conclusion

We have developed a vehicle tracking framework based on the Carla simulator, dividing the vehicle tracking task into two sub-tasks: perceiver and controller.

Based on this framework, we proposed and tested various implementations of the sub-tasks, including:

- Implementing a real-time vehicle detection and positioning algorithm based on YOLOv8 and Bayesian posterior.
- Implementing a PID-based vehicle acceleration and steering control scheme.
- Implementing the Distance-Angle-Fusion vehicle control algorithm.
- Implementing historical path tracking control logic based on DAF.
- Implementing navigation control logic using navigation data provided by Carla.
- Train an autonomous driving model using the DDPG reinforcement learning algorithm (training performance was poor, so it was abandoned).
- Implementing a vehicle collision prediction model based on CNN-LSTMs.

We tested the project on the Carla platform with good results, but there were still some edge cases where tracking the vehicle in extreme following scenarios was difficult.

We consider this acceptable.We are eager to try the effect of deep reinforcement learning on this task and hope to complete it in the future.

References

[1] B. Li, W. Gong, H. Li, and Z. Zhang, "Development and Trends of Autonomous Driving in 2023," *China Smart Internet Development Report (2024)*. Social Sciences Academic Press (China), Beijing, pp. 160–171, Jun. 2024.

[2] P. Liu, "Analysis of the Status Quo and Development Trends of China's Autonomous Driving (Driverless) Industry in 2023." Accessed: Dec. 07, 2024. [Online]. Available: https://www.huaon.com/channel/trend/953114.html

[3] 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, 2021, doi: 10.3390/s21062140.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on robot learning*, 2017, pp. 1–16.

[5] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end Autonomous Driving: Challenges and Frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[6] S. Casas, A. Sadat, and R. Urtasun, "MP3: A Unified Model to Map, Perceive, Predict and Plan," *arXiv preprint arXiv:2101.06806*, 2021.

[7] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016.

[8] O. A. Kim Aleksandr and L. Leal-Taix'e, "EagerMOT: 3D Multi-Object Tracking via Sensor Fusion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[9] X. Li *et al.*, "Poly-MOT: A Polyhedral Framework For 3D Multi-Object Tracking." 2023.

[10] Z. Pang, Z. Li, and N. Wang, "SimpleTrack: Understanding and Rethinking 3D Multi-object Tracking," *arXiv preprint arXiv:2111.09621*, 2021.

[11] H.-k. Chiu, A. Prioletti, J. Li, and J. Bohg, "Probabilistic 3D Multi-Object Tracking for Autonomous Driving," *arXiv preprint arXiv:2001.05673*, 2020.

[12] M. R. K. Akanda, J. Reynolds, T. Jackson, and M. Gray, "Machine Learning Based Object Tracking," *arXiv e-prints*, p. arXiv:2401.07929, Jan. 2024, doi: 10.48550/arXiv.2401.07929.

[13] M. J. Hossain Faruk and M. Asadur Rahman, "Object Detection and Tracking: Deep Learning based Novel Tools to Generate Robust Human and Machine-Annotated Ground Truth Data for Training AI Models," in *2022 4th International Conference on Sustainable Technologies for Industry 4.0 (STI)*, 2022, pp. 1–6. doi: 10.1109/STI56238.2022.10103294.

[14] G. user: perseus784, "Predict Vehicle Collision Moments Before It Happens in Carla!." GitHub, 2024.

[15] P. Jahoda, J. Cech, and J. Matas, "Autonomous Car Chasing," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2020.

[16] Y. Zhang, "CARLA 3D Tracking: A Framework for Monocular 3D Vehicle Detection and Tracking." GitHub, 2024.

[17] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, "Multimodal Motion Prediction with Stacked Transformers," *Computer Vision and Pattern Recognition*, 2021.

[18] Y. Hu *et al.*, "Planning-oriented Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[19] Y. Zhang, J. Song, and S. Li, "3D Object Detection and Tracking Using Monocular Camera in CARLA," in *2021 IEEE International Conference on Electro Information Technology (EIT)*, May 2021, pp. 67–72. doi: 10.1109/EIT51626.2021.9491905.

[20] E. Koçak and T. Soylu, "A performance comparison of YOLOv8 models for traffic sign detection in the Robotaxi-full scale autonomous vehicle competition," *Multimedia Tools and Applications*, vol. 83, pp. 1–31, 2023, doi: 10.1007/s11042-023-16451-1.