DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Semester Thesis

# Autonomous Driving Simulator and Benchmark on Neurorobotics Platform

**Ma Liang**

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Semester Thesis

# Autonomous Driving Simulator and Benchmark on Neurorobotics Platform

# Autonomes Fahren Simulator und Benchmark auf Neurorobotics Plattform

| | |
|---|---|
| Author: | Ma Liang |
| Supervisor: | Prof. Dr.-Ing. habil. Alois C. Knoll |
| Advisor: | Liguo Zhou |
| Submission Date: | 20.12.2022 |

I confirm that this thesis type (bachelor's thesis in informatics, master's thesis in informatics: games engineering, ...) is my own work and I have documented all sources and material used.

Munich, Submission date: Dec .20,2022 Author: Ma Liang

# Abstract

Today's automotive industry is moving towards the direction of electrification and intelligence. With the development of machine learning and deep learning different solutions for autonomous driving are emerging. In order to test autonomous driving solutions or to train deep learning neural network models, a large number of real-world scenarios are often required. But deploying and testing autonomous driving solutions in real-world environments usually requires various approval documents and consumes a lot of time and money. That's why we are working to develop a Unity-based autonomous driving simulation platform. In this project we used unity's state-of-the-art HDPR rendering pipeline and created a high-resolution map based on Garching bei Muenchen using modeling software such as Blender. At the same time we used the NWH (Vehicle Physics 2)Unity package to simulate realistic car physics on our simulated vehicle. In order to be able to deploy autonomous driving algorithms we use ROS2 nodes as communication medium to build different interfaces. In the experimental phase we trained and deployed the most current autonomous driving algorithms, such as YOLOV5 and Pointpillars, and finally we were able to make our simulated car follow the specified route in the virtual city and achieve obstacle avoidance and overtaking. The core of the whole project is to realistically reproduce the real self-driving car environment.

# Kurzfassung

Die heutige Automobilindustrie entwickelt sich immer mehr in Richtung Elektrifizierung und Intelligenz. Mit der Entwicklung von maschinellem Lernen und Deep Learning entstehen verschiedene Lösungen für das autonome Fahren. Um Lösungen für autonomes Fahren zu testen oder Deep-Learning-Modelle für neuronale Netze zu trainieren, ist oft eine große Anzahl von realen Szenarien erforderlich. Der Einsatz und die Erprobung von Lösungen für autonomes Fahren in realen Umgebungen erfordert jedoch in der Regel verschiedene Genehmigungsdokumente und ist mit einem hohen Zeit- und Kostenaufwand verbunden. Aus diesem Grund arbeiten wir an der Entwicklung einer Unity-basierten Simulationsplattform für autonomes Fahren. In diesem Projekt haben wir die neueste HDPR-Rendering-Pipeline von Unity verwendet und eine hochauflösende Karte von Garching bei München mit Hilfe von Modellierungssoftware wie Blender erstellt. Gleichzeitig haben wir das Unity-Paket NWH (Vehicle Physics 2) verwendet, um eine realistische Fahrzeugphysik auf unserem simulierten Fahrzeug zu simulieren. Um autonome Fahralgorithmen einsetzen zu können, nutzen wir ROS2-Knoten als Kommunikationsmedium, um verschiedene Schnittstellen aufzubauen. In der experimentellen Phase haben wir die gängigsten autonomen Fahralgorithmen wie YOLOV5 und Pointpillars trainiert und eingesetzt. Schließlich konnten wir unser simuliertes Auto dazu bringen, der vorgegebenen Route in der virtuellen Stadt zu folgen und Hindernisse zu umfahren und Fahrzeuge zu überholen. Der Kern des gesamten Projekts ist die realistische Nachbildung der realen Umgebung für selbstfahrende Autos.

# Contents

# 1 Introduction

In this chapter I will describe the need for our project and the contribution that this project makes.

## 1.1 Motivation

Nowadays autonomous driving is already an undisputed trend in the automotive industry. An important part of the autonomous driving development process is to validate and test autonomous driving algorithms in real road environments. However, testing in the real road environment has a high probability of causing traffic accidents and even casualties due to algorithm errors or corner cases on the road. In order to ensure the reliability and stability of autonomous driving algorithms, real road testing of about 8.8billion miles must be conducted. So the testing process will cost a huge amount of money and time, which is a problem for both companies and research institutes.
The autonomous driving simulation platform provides powerful algorithmic simulation capabilities through high fidelity maps and 3d digital twin scenes. The self-driving simulation platform has multiple built-in sensors and provides rich interfaces to deploy different autonomous driving algorithms. The built-in auto simulation program enables fully automated simulation to maximize time savings. No need for real roads and cars to avoid unnecessary traffic accidents. The rich interfaces and adaptability make it easier for developers to debugging, so that the development and testing of algorithms based on autonomous driving simulation platform will be more popular.

## 1.2 Contribution

Our project uses the latest unity high-definition rendering pipeline (HDRP) to build 3D twin digital scenes, and we build traffic and road environments based on real satellite navigation maps so we can recreate real-world road conditions.See Figure 1.1. We also used NWH for the first time in an autonomous driving simulation platform to adjust the mechanical parameters of the car to simulate real car dynamics. These contributions aim to create more realistic simulation scenarios.
In addition, to be compatible with different autonomous driving algorithms we use ROS2 nodes to perform the following functions: See Figure 1.2.Send the collected images, point clouds from virtual sensors in Unity to ROS2 in the corresponding ROS message format; Implement object detection and stereo depth estimation algorithms in the corresponding ROS execution units (i.e. nodes), process the data published from Unity, and obtain the

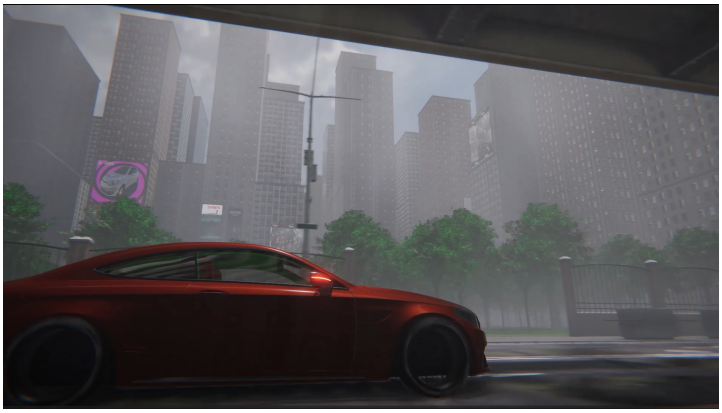

Figure 1.1: Unity HDRP 3D Highly Realistic Scene

information we need: the detected objects in the image and their distances to our ego agent in the simulation environment; Finally publish and visualize them back in Unity.In order to test our autonomous driving platform memory for better autonomous driving functionality, in this project we use the Kitti dataset to train the YOLOV5 algorithm to improve the accuracy of visual image processing. At the same time we learn and use MMdetection3D platform (Pointpillars) to predict 3D-boundingbox based on point cloud information.

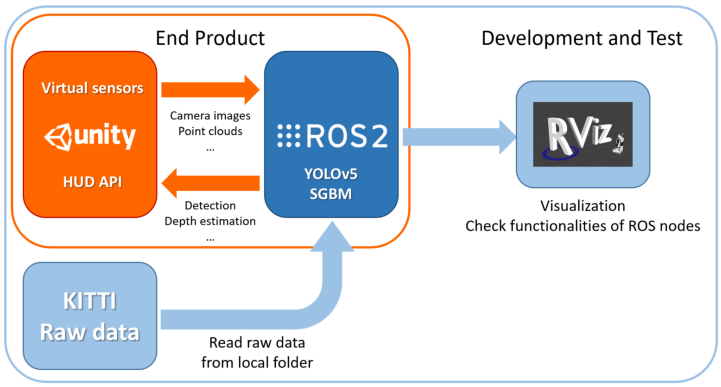


Figure 1.2: Interfaces and Information transmission paths

# 2 Background

In this chapter we present the tools, algorithms, frameworks and software used in this project.

## 2.1 Yolo(You Only Look Once)-V5

The current deep learning methods in the field of target detection are mainly divided into two categories: two-stage target detection algorithms; one-stage target detection algorithms. In the former, the algorithm first generates a series of candidate frames as samples, and then classifies the samples by convolutional neural network[1]; in the latter, the problem of target border localization is directly transformed into a regression problem without generating candidate frames[2]. It is due to the difference between the two methods that there is also a difference in performance, with the former having the advantage in detection accuracy and localization precision and the latter in algorithm speed[2].
YOLO (You Only Look Once) is a one-stage object detection algorithm introduced by Redmon et al. in the year 2016[2]. In comparison to two-stage algorithms, YOLO reframes object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities[2].The YOLOv5 repository [3]was created on 2020-05-18 and has been iterated for many major versions up to today, we use version v6.1 released on 2022-3-19.YOLOv5 has been updated and improved in terms of network structure, data augmentation, training strategy, and loss function to achieve faster inference speed and higher prediction accuracy (COCO dataset) than V4.See Figure 2.1[2].

| Model | size (pixels) | mAP$^{val}$ 0.5:0.95 | mAP$^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.0 | 45.7 | **45** | **6.3** | **0.6** | **1.9** | **4.5** |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | 0.9 | 7.2 | 16.5 |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | 1.7 | 21.2 | 49.0 |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | 2.7 | 46.5 | 109.1 |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 |
| YOLOv5n6 | 1280 | 36.0 | 54.4 | 153 | 8.1 | 2.1 | 3.2 | 4.6 |
| YOLOv5s6 | 1280 | 44.8 | 63.7 | 385 | 8.2 | 3.6 | 12.6 | 16.8 |
| YOLOv5m6 | 1280 | 51.3 | 69.3 | 887 | 11.1 | 6.8 | 35.7 | 50.0 |
| YOLOv5l6 | 1280 | 53.7 | 71.3 | 1784 | 15.8 | 10.5 | 76.8 | 111.4 |
| YOLOv5x6 | 1280 | 55.0 | 72.7 | 3136 | 26.2 | 19.4 | 140.7 | 209.8 |
| + TTA | 1536 | **55.8** | **72.7** | – | – | – | – | – |

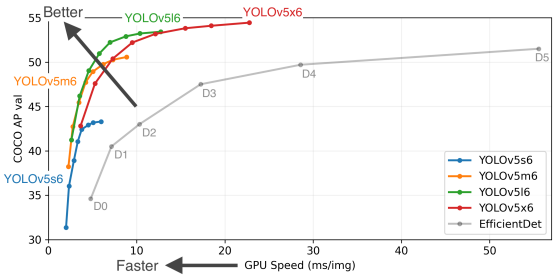


Figure 2.1: YOLO-V5 performance comparison

### 2.1.1 Input: Mosaic data augmentation, Adaptive Anchor calculation, Adaptive Image Scaling

(a) Mosaic data augmentation: Mosaic is a reference to the CutMix data enhancement proposed at the end of 2019, but while CutMix uses only two images for stitching, Mosaic data enhancement uses four images, randomly scaled, randomly cropped, and randomly aligned for splicing.See Figure 2.2.

(b) adaptive Anchor calculation: In the Yolo algorithm, for different data sets, there are Anchor with initial set length and width. In the network training, the network outputs the prediction bounding-box based on the initial Anchor, and then compares it with the groundtruth, calculates the gap between them, and then updates it backwards to iterate the network parameters.Yolov5 embeds this function into the code and adaptively calculates the best Anchor value for different training sets each time it is trained.

(c) Adaptive Image Scaling: In common target detection algorithms, different images have different lengths and widths, so the common way is to uniformly scale the original image to a standard size and then feed it into the detection network. The authors believe that in the actual use of the project, many images have different aspect ratios, so after scaling and filling, the size of the black edges at both ends are different, and if there is more filling, there is information redundancy, which affects the inference speed. Therefore, the letterbox function in datasets.py in Yolov5's code is modified to add the least amount of black borders to the original image adaptively. The black edges at both ends of the image height become less, and the computational effort is reduced during inference, i.e., the target detection speed is improved.See Figure 2.2.

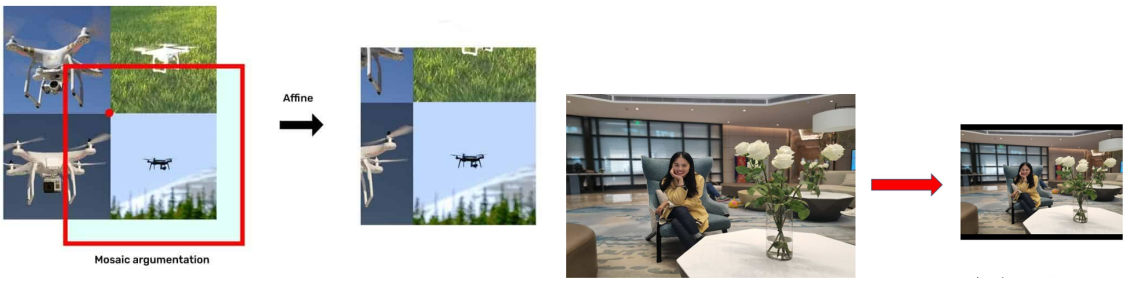


Figure 2.2: Data augmentation (left) and Adaptive Image Scaling(right)

### 2.1.2 Backbone: New CSP-Darknet53[4, 5]

YOLOv5 has a small change since v6.0 compared to the previous version, replacing the first layer of the network (which was the Focus module) with a 6x6 sized convolutional layer. The two are theoretically equivalent, but for some existing GPU devices (and the corresponding optimization algorithms) using a 6x6 convolutional layer is more efficient than using the

Focus module. The following Figure 2.3 shows the original Focus module (similar to the previous Patch Merging in Swin Transformer), which divides each 2x2 neighboring pixel into a patch, and then puts the same position (same color) pixels in each patch together to get 4 feature maps, and then adds a 3x3 sized convolution layer. This is equivalent to using a 6x6 convolutional layer directly.
Yolo-V5 borrows the design idea of CSPNet(See Figure 2.4) and designs the CSP structure in Backbone and Neck of the backbone network.The CSP module consists of a CBL module, a Res unint module, and a convolutional layer, Concate operation[4].Compared with the previous network CSP structure can :

(1) enhance the learning ability of CNN

(2) making it possible to maintain accuracy while being lightweight

(3) Reduced computational bottlenecks reduce memory costs

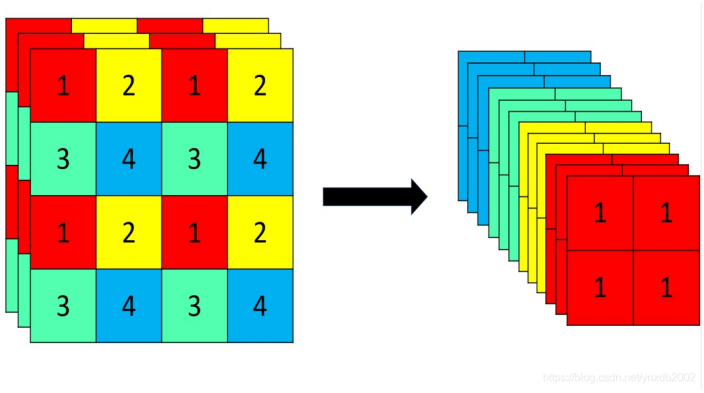


Figure 2.3: Interfaces and Information transmission paths

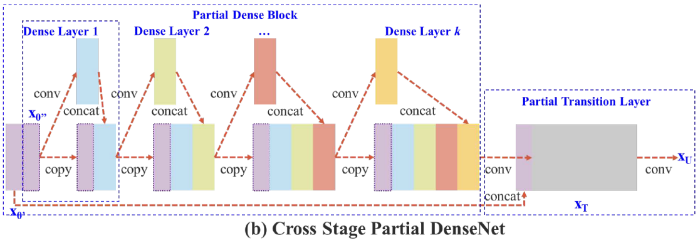


Figure 2.4: CSP(Cross Stage Partial Densenet)[4]

### 2.1.3 Neck:SPPF , New CSP-PAN

The first is to replace the SPP [6] with the SPPF (designed by Glenn Jocher himself), both serve the same purpose, but the latter is more efficient. the SPP structure, shown in the

Figure 2.5 below, is to pass the input through several MaxPool of different sizes in parallel, and then do further fusion, which can solve the target multi-scale problem to some extent[3]. The SPPF structure is to serialize the input through multiple MaxPool layers of size 5x5. Here it should be noted that two MaxPool layers of size 5x5 serially are the same as a MaxPool layer of size 9x9, and three MaxPool layers of size 5x5 serially are the same as a MaxPool layer of size 13x13[3].
Another difference in the Neck part is the New CSP-PAN[4]. In YOLOv4, the PAN structure of Neck was not introduced in the CSP structure, but in YOLOv5 the author added the CSP in the PAN structure.

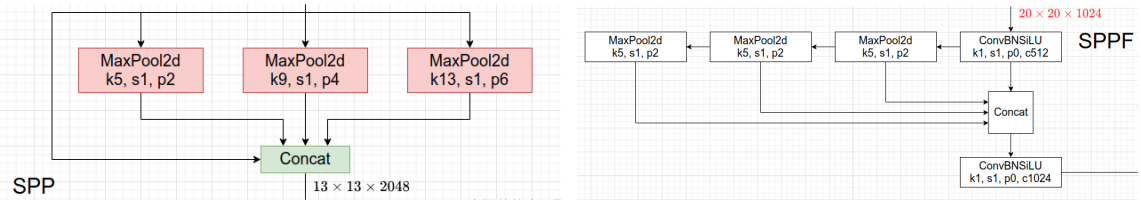


Figure 2.5: SPP(left) and SPPF(right) sturcture

### 2.1.4 Loss Function

The loss function for the target detection task generally consists of two parts, Classificition Loss (classification loss function) and Bounding Box Regeression Loss (regression loss function).The loss of YOLOv5 consists of three main components.

(1) Classes loss: BCE loss is used, and note that only the classification loss of positive samples is calculated.

(2) Objectness loss: still BCE loss is used, note that the obj here refers to the CIoU of the target bounding box and GT Box of the network prediction. obj loss is calculated here for all samples.

(3) Location loss: the CIoU loss is used, and only the location loss of positive samples is calculated.

The development process of Loss of Bounding Box Regeression in recent years is: Smooth L1 Loss -> IoU Loss (2016) -> GIoU Loss (2019) -> DIoU Loss (2020) -> CIoU Loss (2020).Let's start with the most commonly used IOU_Loss,see Figure 2.6, and perform a comparative teardown analysis to see why Yolov5 chooses C_IoU Loss[3].

$$CIoU = IoU - (\frac{\rho^2(b, b^{gt})}{c^2} + \alpha v)$$

$$v = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h})^2$$

$$\alpha = \frac{v}{(1 - IoU) + v}$$

$$L_{CIoU} = 1 - CIoU$$

where $v$ is the parameter that measures the consistency of the aspect ratio . $\rho(b, b^{gt})$ is the Euclidean distance between the two centroids. $c$ is the diagonal distance of the minimum outer rectangle[3].IOU_Loss mainly considers the overlapping area of the detection frame and the target frame. CIOU_Loss: On the basis of IOU, it solves the problem when the bounding boxes do not overlap, considering the information of the distance from the center point of the bounding box and the scale of the bounding box aspect ratio. the regression method of CIOU_Loss is adopted in Yolov5, which makes the speed and accuracy of the prediction box regression a bit higher.

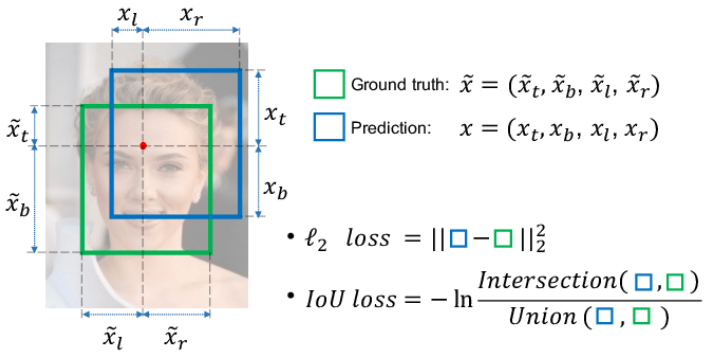


Figure 2.6: IoU_Loss

## 2.2 MMDetection3D

Along with the rapid development of autonomous driving technology and the popularity of LiDAR, 3D target detection has gradually become a research hotspot in the industry and academia in recent years. The release of MMDetection3D (MMDet3D for short) fills this gap.Major features[7]:

- It directly supports multi-modality/single-modality detectors including MVXNet,etc.It can also directly use all 300+ models and 40+ algorithms[8] inside the training MMDetection, supporting the most number of algorithms and coverage directions for the 3D detection code base.Most of the multiple codebases in the 3D detection field focus on point cloud-based unimodal detection, and the codebase for multimodal detection does not support much unimodal detection. Due to the increasing importance of multimodal 3D detection tasks, we have supported multimodal (image + point cloud) 3D detection inside MMDetection3D, which now supports MVX-Net's model at KITTI. From now on, new work can be based on direct comparison between MMDetection3D and other unimodal or multimodal methods, alleviating much of the burden caused by codebase migration and article reproduction, etc[7].

- MMDetection3D supports 5 major datasets, including SUN RGB-D, ScanNet, nuScenes, Lyft, and KITTI[9], making it the largest 3D inspection code base.Due to the difference between indoor and outdoor 3D inspection datasets, few projects have been experimenting and comparing on both types of datasets, so there is no codebase that can support both indoor and outdoor datasets and related methods well. MMDetection3D abstracts the pre-processing process of indoor and outdoor datasets into a set of data pipeline, and at the model level, it has realized that part of the model is independent of the coordinate system[7].
- MMDetection3D has the fastest training speed and supports pip install with one click, easy to use. In the case of 8-card distributed training and using the same superparameter, the training speed (number of samples/second) is compared with the following Figure 2.7; we have typed x for the corresponding models not supported in other codebase[7].
- Currently, no 3D detection codebase can directly support 2D detection SOTA in its own codebase, but MMDetection3D does so. MMDetection3D uses a lot of MMDetection code (e.g., training-related hooks are implemented in MMCV, and functions such as train_detector are not written in MMDetection3D because they are all detection), so with the right config file, 300+ models and 40+ algorithms in MMDetection model zoo can be used in MMDetection3D. MMDetection3D can be used normally in MMDetection3D[7].

| Methods | MMDetection3D | OpenPCDet | votenet | Det3D |
|---|---|---|---|---|
| VoteNet | 358 | × | 77 | × |
| PointPillars-car | 141 | × | × | 140 |
| PointPillars-3class | 107 | 44 | × | × |
| SECOND | 40 | 30 | × | × |
| Part-A2 | 17 | 14 | × | × |

Figure 2.7: IoU_Loss

## 2.3 Neurorobotics Platform(NRP)[10]

Neurorobotics is an emerging transdisciplinary field of research at the interface between neuroscience, artificial intelligence and robotics. The Neurorobotics Platform (NRP) [10]is the service for embodied simulation developed in the context of the Human Brain Project - a FET Flagship funded by the European Commission

- Neurorobotics Platform, NRP[10], is a simulation platform that connects brain models (ranging from spiking neural networks to deep networks) to body models. It enables its users to simulate agents interacting in closed-loop with their virtual environment,

thus connecting brain dynamics to behavioural outputs. This approach sheds a new light on the brain mechanism involved in the perception-cognition-action loop, and will eventually support the creation of a new generation of robots displaying situational awareness, adaptation to evolving experimental parameters, ability to plan complex sequences of actions, and more.

- Neurorobotics Platform, NRP[10], is a simulation platform that connects brain models (ranging from spiking neural networks to deep networks) to body models. It enables its users to simulate agents interacting in closed-loop with their virtual environment, thus connecting brain dynamics to behavioural outputs. This approach sheds a new light on the brain mechanism involved in the perception-cognition-action loop, and will eventually support the creation of a new generation of robots displaying situational awareness, adaptation to evolving experimental parameters, ability to plan complex sequences of actions, and more.

- The NRP [10]is an open access and open source integrative simulation framework that provides opportunities for innovation in both the robotics and neuroscience spaces. It facilitates experiments in cognitive research by implementing experimental workflows that make it possible for researchers to test and compare brain models in embodied settings. The NRP allows its users to create scenarios where a brain model controls a virtual agent interacting with a realistic simulated environment. Testing procedures that can otherwise be difficult to implement on a physical setup - like testing of safety-critical situations or scenarios that could result in damage to the robot - can now be performed in the NRP without incurring any risk to a costly physical plant. By providing extensive control over experimental conditions, scripting capabilities and physically realistic interactions with the simulated environment, the NRP offers opportunities for closed-loop neuroscience that are unmatched by any competing simulation framework.

- Roboticists using the NRP can further their ideas of bio-inspired, safe and adaptable robotic systems. Novel robotic controllers can easily be designed and tested on the NRP, paving the way for enhanced cognitive abilities such as contextual awareness and decision-making[10].

- With the NRP, neuroscientists can observe and analyse emergent behavioural patterns in virtual agents controlled by models of brain architecture and functions. This provides the unprecedented ability to test and refine these models iteratively, without the many approximations entailed by simulating the brain decoupled from the body[10].

- Thanks to its direct connection to both Neuroscience research in HBP and the EBRAINS HPC infrastructure, the NRP is the best available platform for scientific exploration and development of neuromorphic computing applications[10].

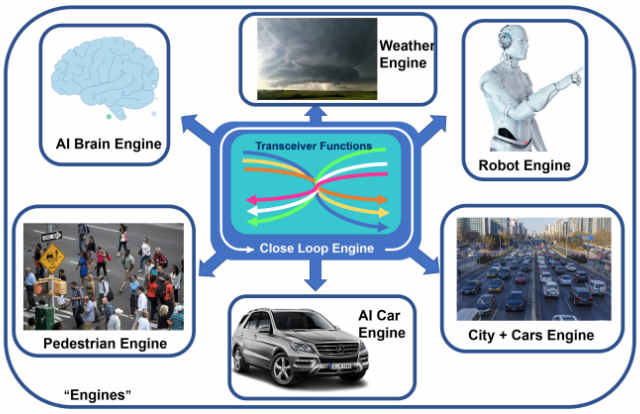


Figure 2.8: NRP basic model[10]

## 2.4 Robot Operation System (ROS)

Robot Operating System (ROS or ros) is an open-source robotics middleware suite. It is not an operating system (OS) but a set of software frameworks for robot software development. Sometime before 2007, the first pieces of what eventually would become ROS began coalescing at Stanford University. Later on Willow Garage began developing the PR2 robot as a follow-up to the PR1, and ROS as the software to run it. Groups from more than twenty institutions made contributions to ROS, both the core software and the growing number of packages which worked with ROS to form a greater software ecosystem. ROS has provided the robot community with a relatively complete set of intermediate layers, tools, software and even common interfaces and standards. It can be said that with ROS, developers in the field of robotics industry can quickly develop system prototypes and do testing and verification without reinventing wheels. Software in the ROS Ecosystem can be separated into three groups:

- language- and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as roscpp;
- Packages containing application-related code which uses one or more ROS client libraries.

ROS was designed to be open source, intending that users would be able to choose the configuration of tools and libraries which interacted with the core of ROS so that users could shift their software stacks to fit their robot and application area. As such, there is very little which is core to ROS, beyond the general structure within which programs must exist and communicate. In one sense, ROS is the underlying plumbing behind nodes and message passing. However, in reality, ROS is not only that plumbing, but a rich and mature set of tools, a wide-ranging set of robot-agnostic abilities provided by packages, and a greater ecosystem

of additions to ROS. As one of the most popular projects in the robot-related open source community, there are already a large number of well-developed open source applications based on ROS, covering perception, planning, control, positioning, SLAM and mapping, visualization and almost all robot fields.

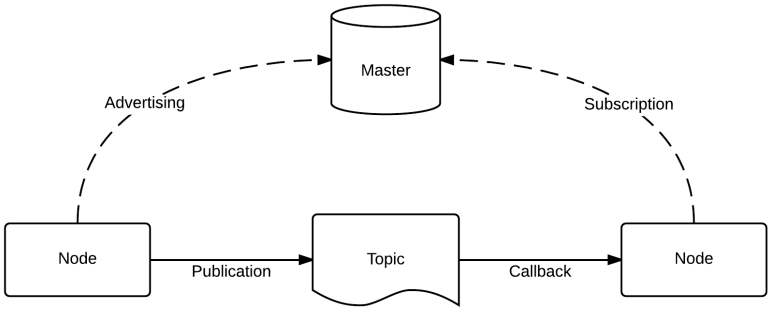


Figure 2.9: Exchange Data with ROS Publishers and Subscribers

## 2.5 Unity3D

### 2.5.1 History of Unity3D

Unity3D is a comprehensive multi-platform game development tool developed by Unity Technologies that allows you to easily create types of interactive content such as 3D video games, architectural visualizations, real-time 3D animations, and is a fully integrated professional game engine.There are countless commercial game engines and free game engines in the industry, among which the most representative commercial game engines are UnReal, CryENGINE, Havok Physics, Game Bryo, Source Engine, etc. However, these game engines are expensive, which makes the game development cost increase greatly. Unity has proposed the slogan of "Democratizing Development" to provide a good game engine that anyone can develop easily, so that developers no longer worry about the price. Unity3D has the following features:

- visual programming interface to complete a variety of development work, efficient script editing, easy development.
- Automatic instant import, Unity supports most 3D models, bones and animations directly imported, mapping materials automatically converted to U3D format.
- multi-platform development and deployment of works with just one click.
- underlying support for OpenGL and Direct11, simple and practical physics engine, high-quality particle system, easy to get started, realistic effects.
- support for Java Script, C, Boo scripting language.

- Unity performance is excellent, the development efficiency is outstanding, very cost-effective advantages.
- support from single-player applications to large-scale multiplayer network game development.

### 2.5.2 High Definition Render Pipeline(HDRP)

The HDRP [11]is a high-fidelity programmable render pipeline developed by Unity for modern (compute shader compatible) platforms.HDRP leverages physics-based lighting techniques, linear lighting, HDR lighting, and a configurable hybrid tiling/clustering delay/forward lighting architecture to give you the tools you need to create games, tech demos, animations, and other applications that meet high graphics standards. HDRP is oriented to the field of PC games, console games, VR products, and may support IOS phones using Matal and Android phones using Vulkan in the future. Its most notable feature is physics-based, so HDRP makes more adjustments to the three aspects of lighting, materials, and cameras compared to the build-in built-in pipeline and URP (Universal Render Pipeline). In addition, at a more fundamental level, HDRP uses hybrid Tile/Cluster deferred/Forward and other rendering algorithms to mix and configure to bring higher performance and a variety of configuration methods. Unlike Build-in Pipeline and URP, HDRP is render path-independent, meaning that whether you choose Forward or Deferred, you can use the non-rich feature set provided by HDRP. HDRP supports Forward and Deferred materials. The deferred materials are more efficient than the forward materials, but support a slightly smaller set of properties. Forward materials have no property restrictions, but are less efficient.
HDRP has the following relevant features: The built-in Shader-Lit, which is derived from the build-in pipeline standard, is based on the PBR lighting model, but Lit is more powerful and can handle more complex and variable situations, such as standard (for simulating general hard surfaces such as metal, wood, etc.), Subsurface scattering (used to simulate skin, jade, leaves), Anisotropy anisotropy (used to create those surfaces where the highlights change when viewed from different angles, such as brushed metal or velvet), Iridescence iridescence (used to create those surfaces where the colors change when viewed from different angles, such as soap bubbles or insect wings), Specular Color mirror color (used to create those who want to have a specific color highlights of the surface), Clear Coat clear coat (used to simulate a thin layer of semi-transparent objects attached to the standard surface, such as car paint, glazed objects).Built-in Shader-LayeredLit, mix up to four materials to get a rich and realistic blend effect, mix intensity by a mask map of the RBGA channel control, in FontainebleauDemo,see Figure 2.10 can see the specific application.Built-in Shader-Decal, decal, which is heavily used in almost all HDRP projects, maps Decal to walls, floors, machinery, and can get a lot of details such as dirty old, blood, and water stains.Built-in Shader Graph Master Node-Fabric for simulating real fabrics.Built-in Shader Graph Master Node-Hair , this node uses the Kajiya-Kay lighting model for simulating realistic hair effects.The built-in Shader Graph Master Node-StackLit, which allows the same material to have multiple lighting characteristics, may bring performance degradation, but

its simulation of physical characteristics is better than Lit, for example, you can use it to use both subsurface scattering and anisotropy on the same material.Built-in Shader Graph Master Node-Eye for eye simulation.Using physics-based light, HDRP light uses physical lighting units to help you light your scene in the most realistic way possible. When using physical light units, you need to follow HDRP's unit conventions (one Unity unit equals one meter) in order for the light to behave correctly. In addition, there are additional types of tube light sources, disk light sources, color temperatures, color-tunable coockie, and other new features.Three sky types, Physically Based Sky, HDRI Sky, and Gradient Sky, Physically Based Sky is used to simulate the real atmosphere, HDRI uses a High Dynamic Range Image map that does not change to simulate the sky, and Gradient Sky uses different colors at the top, middle, and bottom as a simple simulation of the sky.Fog, unity provides a lot of built-in and easy to configure features, such as local fog, height fog, index fog, volume fog, here volume fog to say, volume fog is also called volume light, God's light, used to simulate the Tyndall effect, very out of effect, for example, deep in the mountains of the headlights, fog under the street lights, smoke and dust in the light source, the site of the sun Figure 2.10.HDRP supports cubemap and planar GPU Reflection Probe planar GPU reflection probes to help you produce realistic reflections in real-time scenes. HDRP supports screen-space reflections and refractions. Based on the screen space light and the environment around the object, it uses the screen depth buffer and color buffer to simulate the path of light propagation to the camera, so reflections and refraction can be accurately calculated.

Figure 2.10: HDPR Demo[12]

### 2.5.3 NWH Vehicle Physics 2 Package

NWH Vehicle Physics 2 [13]is a complete vehicle simulation package for Unity game engine. Realistic, easy to use and heavily customizable.

#### Physics Features

- Model based on inertia, torque and angular velocity ensures realistic behavior in all conditions. E.g. vehicle can be started by rolling it downhill, given that it has ignition and the clutch is engaged.

- Drivetrain solver runs at multiple steps per FixedUpdate ensuring immediate response and physical accuracy, while heavily optimized code makes it very fast.
- Output of individual drivetrain components can be set to any other component, e.g. connecting clutch directly to the wheel is possible.
- Engine with adjustable power curve, losses, rev limiter, forced induction (turbocharger or supercharger) and simulated fuel consumption (module). Engine can be stalled. Support for ICE and electric engines/motors.
- Clutch can be either manual or automatic, with adjustable slip torque and engagement. Releasing the clutch too fast will stall the engine.
- Transmission can be Automatic, Automatic Sequential, Manual, CVT or External. Realistic gear ratios with adjustable shifting behavior make it suitable for any type of vehicle. Shift duration, variable shift points, incline effect, etc. are all adjustable at runtime. Supports unlimited number of forward and reverse gears.
- Differentials - Open, Locked, VLSD, HLSD or External. Any number and configuration of differentials is possible, be it FWD, RWD, AWD, 6×6, 8×8, etc. Due to solver using sub-stepping locked differentials can be extremely stiff.
- Wheels and Suspension are handled by WheelController3D which not only gives them 3D ground detection but also makes them adjustable in all the ways possible. Spring and damper curves with bump and rebound damping, adjustable spring curves for progressive suspension, camber, anti-squat geometry, rim offset, inertia, drag, surface friction presets using modified Pacejka or completely custom curves, individually adjustable longitudinal and lateral friction, load/grip curve, adjustable ground detection resolution, etc.
- Axles - each axle can have one or more wheels. Each axle has adjustable steering (steer coefficient, Ackerman, toe angle), adjustable brake and handbrake strength, caster and camber angles and also ARB (Anti-roll bar). Solid axles are supported, check Monster Truck example in the demo.
- Simulated aerodynamic drag and downforce.

**Effects**

- Damage affects vehicle performance and handling. Optimized queue-based mesh deformation that spreads processing over multiple frames.
- Sound system with master settings, AudioMixer and 15 different sound effects. AudioSources are set up automatically at runtime and no manual setup is required. Persistent, surface dependent, procedural mesh skidmarks.

- Vehicle lights with low beam, high beam, tail and brake lights and blinkers supported. Can be used with any number of lights of any type and/or emissive materials. Surface particle effects that are set up automatically and depend on the surface type. Exhaust smoke and exhaust flames.

**Modules**

External module system makes adding extra functionality(ABS,TCS,ESC...) to the fast and easy, with the code being similar to that of a typical MonoBehavior with familiar functions such as Update() and FixedUpdate().


Figure 2.11: Applicatuon of NWH_Package in HDRP

### 2.5.4 Fantastic-City-Generator(FCG) Package[14]

FCG is an application package under unity asset store. Using the fantasy city generator, you can create cities of all sizes, with rivers, tunnels and bridges, just like reality. The FCG package includes 236 unique buildings and a variety of 3D objects, and it's easy to add third-party buildings. Also included in the FCG package is a vehicle traffic system with low polygon cars. It is possible to resize and change the city dimensions as needed and to add other 3d car models.

# 3 My Work

## 3.1 Train YOLO_V5 Algorithm with Kitti Dataset

### 3.1.1 Dataset and label pre-processing

The KITTI dataset[9], founded by the Karlsruhe Institute of Technology and the Toyota Institute of Technology, is the largest international dataset for evaluating computer vision algorithms in autonomous driving scenarios. The KITTI dataset is used to evaluate the performance of computer vision technologies such as stereo, optical flow, visual odometry, 3D object detection and 3D tracking in an in-vehicle environment. The data consists of up to 15 vehicles and 30 pedestrians per image, with various levels of occlusion and truncation. The entire dataset consists of 389 pairs of stereo images and optical flow maps, a 39.2 km visual ranging sequence, and over 200k images of 3D labeled objects, sampled and synchronized at 10 Hz.The KITTI data acquisition platform includes 2 grayscale cameras, 2 color cameras, a Velodyne 3D LIDAR, 4 optical lenses, and a GPS navigation system. According to the previous introduction, the YOLO algorithm is a 2d image detection. Therefore, we download the color image dataset from the official website KITTI[9], a total of 12G images. Then we download the label information from the official KITTI website according to the corresponding images. Thus, we obtained the training dataset and the corresponding Groundtruth. txt files of the KITTI dataset record the labels for each frame (each image). txt files of the KITTI labels[9] are stored in the KITTI format,Table 3.1. where [9]:

(1) the first string: represents the object category 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare', for a total of eight categories. One note: the 'DontCare' label indicates that the area is not labeled, for example because the target object is too far away from the LIDAR. To prevent false positives in the evaluation process (mainly in the calculation of the precision), the evaluation script automatically ignores the prediction results for the 'DontCare' region in order to prevent the region that was originally the target object but was not labeled for some reason.

(2) The second number: represents whether the object is truncated or not floating from 0 (non-truncated) to 1 (truncated), where truncated refers to objects that leave the image boundary.

(3) The third number: whether the object is occluded or not. The integers 0, 1, 2, 3 indicate the degree of occlusion, where 0: fully visible 1: partially occluded 2: mostly occluded 3: completely occluded (unknown).

(4) The fourth number: alpha, the observation angle of the object, the angle between the vertical line of the line between the center of the camera and the center of the object and the direction of the object, range: -$\pi \sim \pi$. is in the camera coordinate system, with the camera origin as the center, the line from the camera origin to the center of the object as the radius, rotate the object around the camera y-axis to the camera z-axis, at this time, the angle between the direction of the object and the camera x-axis $r\_y + \pi/2 - \theta = \alpha + \pi/2$ (i.e., the purple angle in the figure is equal), so alpha

(5) The 4 numbers from 5 to 8: 2-dimensional bounding box of the object xmin, ymin, xmax, ymax.

(6) The 3 numbers from 9 to 11: the dimensional height, width, and length (in meters) of the 3-dimensional object.

(7) The 3 numbers from 12 to 14: the position of the 3-dimensional object x,y,z (in the camera coordinate system, in meters)

(8) The 15th number: the spatial orientation of the 3-dimensional object: rotation_y (should be the above r_y), the global orientation angle of the object in the camera coordinate system (the angle between the forward direction of the object and the x-axis of the camera coordinate system), range: $-\pi \sim \pi$.

(9) 16th number: confidence of detection (not useful for data before training, ONLY for result)

And we need to convert the KITTI format labels into the form of YOLO format labels that the YOLO network can use directly. The YOLO format label information format is the first column for the target category, followed by four numbers for the coordinates of the upper left and lower right corners of the box, which can be seen as numbers less than 1, because the proportion of the corresponding whole picture, so even if the image is stretched and deflated, this txt format label can also find the corresponding target,Table 3.2. Therefore, we only need to use the first digit for the category when training YOLOV5 with KITTI format labels, and the fifth to eighth digits represent the four corner positions of the prediction bounding box. And we need to convert the KITTI format tags into the form of YOLO format labels that the YOLO network can use directly. The YOLO format label information format is the first column for the target category, followed by four numbers for the coordinates of the upper left and lower right corners of the box, which can be seen as numbers less than 1, because the proportion of the corresponding whole picture, so even if the image is stretched and deflated, this txt format label can also find the corresponding target. Therefore, we only need to use the first digit for the category when training YOLOV5 with KITTI format labels, and the 4 digits from the 5th to the 8th digit represent the four corner positions of the prediction frame. To implement the label conversion, we first convert the format of the KITTI dataset ending in txt to an xml format file, which uses only the label information we need. We create a new file and then use it to store the converted xml file. xml files are extensible markup languages that allow you to define and store data in a sharable way. xml supports the exchange of information between computer systems such as websites, databases,

and third-party applications. Predefined rules simplify the process of transferring data as XML files over any network, and recipients can use these rules to read data accurately and efficiently. The converted xml file stores tag information in the form of a tree of nodes. We then create a script to finally convert the converted xml file into a txt file. We get the required dataset (images) and labels (groundtruth).

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Truck | 0.00 | 0 | -1.57 | 599.41 | 156.40 | 629.75 | 189.25 | 2.85 | 2.63 | 12.34 | 0.47 | 1.49 | 69.44 | -1.56 |
| Car | 0.00 | 0 | 1.85 | 387.63 | 181.54 | 423.81 | 203.12 | 1.67 | 1.87 | 3.69 | -16.53 | 2.39 | 58.49 | 1.57 |
| Cyclist | 0.00 | 3 | -1.65 | 676.60 | 163.95 | 688.98 | 193.93 | 1.86 | 0.60 | 2.02 | 4.59 | 1.32 | 45.84 | -1.55 |
| DontCare | -1 | -1 | -10 | 503.89 | 169.71 | 590.61 | 190.13 | -1 | -1 | -1 | -1000 | -1000 | -1000 | -10 |

Table 3.1: Kitti format labels

| | | | | |
|---|---|---|---|---|
| 0 | 0.479492 | 0.688771 | 0.955609 | 0.595500 |
| 1 | 0.736516 | 0.247188 | 0.498875 | 0.476417 |
| 1 | 0.637063 | 0.732938 | 0.494125 | 0.510583 |
| 0 | 0.339438 | 0.418896 | 0.678875 | 0.781500 |

Table 3.2: Yolo format labels

### 3.1.2 Neural network structure and Training preparation

To be able to train the neural network directly using YOLOV5's train.py script[3]. We first need to change the configuration file. Create the kitti.yaml file in the data folder and create relative paths for training and validation based on where the images and labels are stored. Subsequently the categories to be predicted need to be written in the script, for the KITTI dataset there are 8 categories in total, list the names of the categories.To be able to train the neural network directly using YOLOV5's train.py script. We first need to change the configuration file. Create the kitti.yaml file in the data folder and create relative paths for training and validation based on where the images and labels are stored. Subsequently the categories to be predicted need to be written in the script, for the KITTI dataset there are 8 categories in total, list the names of the categories. To train the KITTI dataset we used the YOLO_V5x model[3], where the YOLO_V5x model [3]is the largest model with the highest accuracy. . To achieve higher accuracy we used the initial weights as the official YOLO_V5x pre-training weights[3].

### 3.1.3 Start training

To start training, we use a GTX1080 graphics card, select Batchsize as 8 according to the size of the graphics memory, if you select 16, the card will be prompted to explode the graphics memory. Go to the root directory of YOLOV5 and open a command window and enter the following code.

$$train.py\ --batchsize\ 8\ --epochs\ 300\ --data\ data/kitti.yaml\ --cfg models/yolov5x.yaml$$

– – *weightsyolov5x.pt*

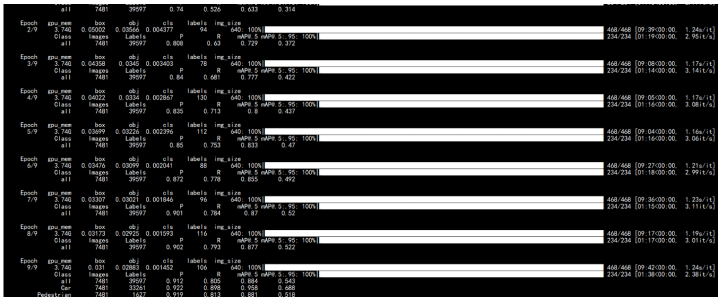

Figure 3.1: Training process

### 3.1.4 Result

The whole training process took 2.5 days. The final result is shown in the Figure 3.2 below, where we are also able to see the distribution information of the labels,see Figure 3.3.At the same time we can also see the P-curve, R-curve and PR-curve of the whole training processsee Figure 3.4. After 300 epochs of training we were able to see that the best training accuracy

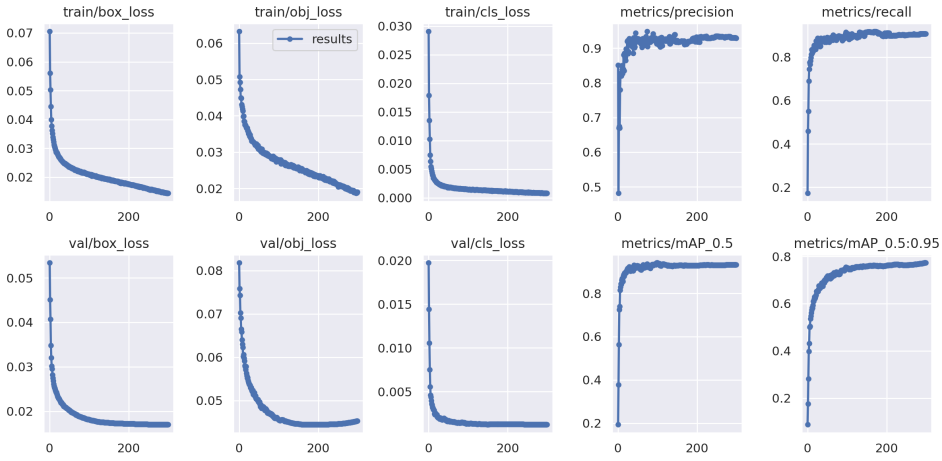


Figure 3.2: The result after 300epoch

reached 94.385% at epoch 46.We also show the results( Figure 3.5) of the prediction bounding box obtained from the validation dataset after 300 epoch training.

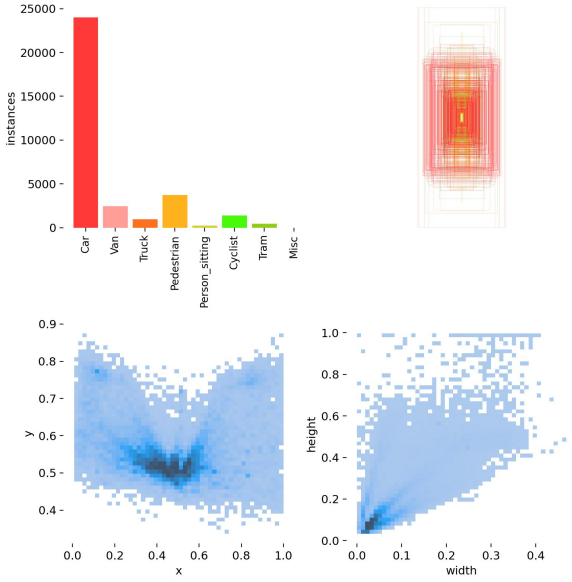


Figure 3.3: Distribution information of the labels

## 3.2 Autonomous driving development based on Unity platform

### 3.2.1 Build highly realistic Unity scenes in HDRP

In order to build an autonomous driving platform based on Unity, we first need to build a realistic enough scenario. Therefore we intend to use HD maps and subsequently build a city traffic scene using Blender and Roadrunner. The resulting 3d city traffic scene is imported into Unity. Since the imported scene does not yet have complete mapping and material sphere issues, we need to map it according to Unity's needs to simulate a more realistic city traffic scene. At the same time we also need to add the weather factor, through the built-in package of Unity HDRP. Then import traffic flow and traffic rules through FCG-Unity package. The specific process of importing traffic flow is that we need to add waypoints [15] to our city. Each section of road forms a traffic flow based on the waypoints. Roads in different directions form two-way lanes according to the order in which the waypoints are added. See Figure 3.6.Traffic rules are then set at each intersection based on the FCG built-in traffic light prefab. In order to achieve a more realistic self-driving testbed, the traffic vehicles in our scenarios are all modeled with a new high fidelity model under Unity HDRP. The models that had already been added to the vehicle physics were added to our traffic flow by adding third-party vehicles through FCG. At this moment these cars are able to drive in a

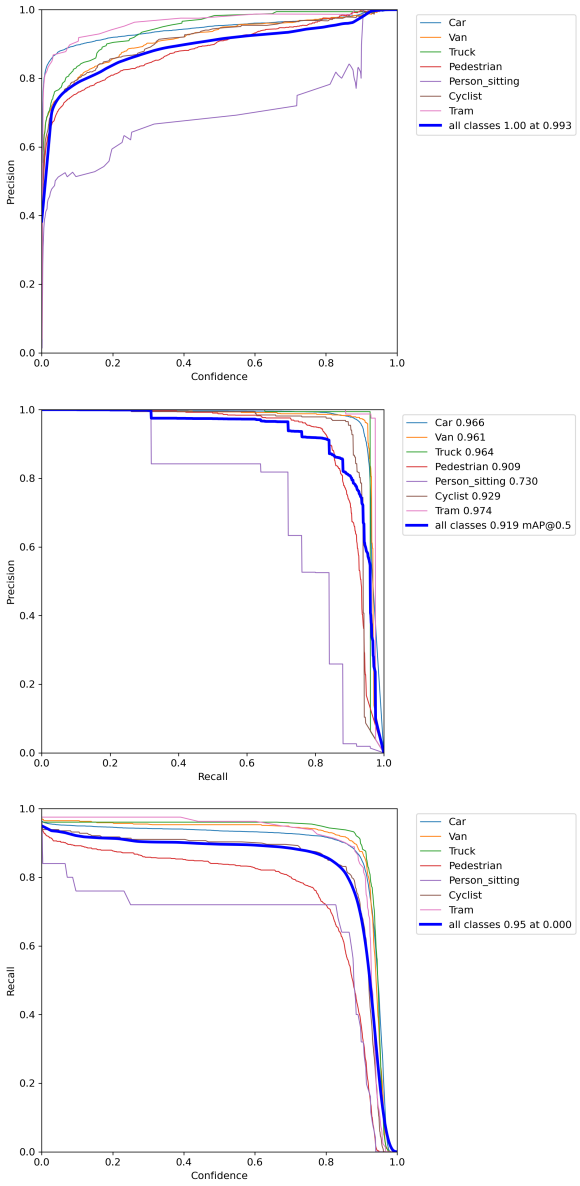


Figure 3.4: From top to bottom P-curve , PR-curve,R-curve

Figure 3.5: Distribution information of the labels

highly realistic city traffic scene while obeying traffic rules.See Figure 3.7.

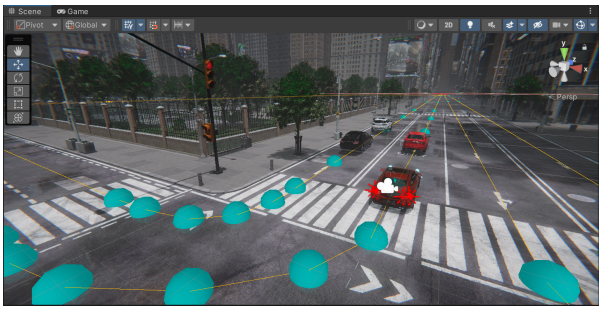

Figure 3.6: Waypoints

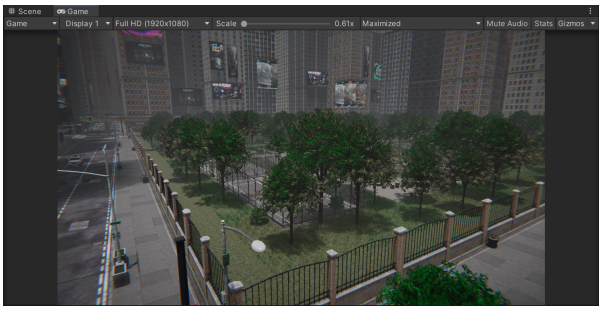

Figure 3.7: Highly realistic city traffic scene

### 3.2.2 Add the most realistic vehicle physics to the test vehicle

In order to achieve a more realistic self-driving simulation, we need to make our test vehicles drive according to real physical characteristics. Therefore, we chose to add the NWH vehicle physics package to our test vehicle, which simulates a real fuel vehicle drivetrain as well as an electric vehicle powertrain and power cell. This package can also detect the road surface and then simulate the vehicle's slip and sway under different power parameters.In order to achieve a more realistic self-driving simulation, we need to make our test vehicles drive according to real physical characteristics. Therefore, we chose to add the NWH vehicle physical package to our test vehicle, which simulates a real fuel vehicle drivetrain as well as an electric vehicle powertrain and power cell. This package can also detect the road surface and then simulate the vehicle's slip and sway under different power parameters. In order to add the NWH package, all crash bodies are first removed according to our chosen test vehicle model. Then search for vehicle setup wizard in the vehicle prefab in the inspector. specify the vehicle type and add each wheel in Wheel Game Objects. finally click Run Setup to add vehicle physics characteristics to this test vehicle model.
Since NWH uses two types of input, one is Input System and the other is an additional script control for autopilot. To use Input System add a new empty Game Object to the test

vehicle prefab and add Input System Vehicle Input Provider and Input System Scene Input Provider to the inspector of the empty Game Object.After running the game through the mouse keyboard or gamepad can control the vehicle start, shift and forward.
For the second control method we need to set AutoSetInput to False in the script so that the default Input System is not activated. By setting a new autopilot script for the Input state to control the vehicle's autopilot. I control the forward speed of the vehicle via a PID controller [16],see Figure 3.8,and change the forward gear of the vehicle based on the current speed and load. Where the differential equation of the PID controller is:

$$u(t) = K_p[e(t) + \frac{1}{T_i}\int_0^t E(t)dt + T_d\frac{de(t)}{dt}]$$

where $K_p$ is the scale factor, $T_d$ is differential time constant, $T_i$ is integral time constant, $e(t)$ is input and $u(t)$ is output.

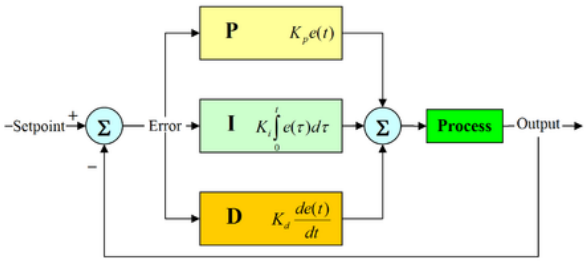


Figure 3.8: PID Controller

### 3.2.3 Integration of vehicle physics and FCG

After adding NWH vehic physics 2 to the test vehicle, changes must be made to the vehicle settings in order for the vehicle to drive normally under the traffic flow constructed by the FCG package. Otherwise, the vehicles in the traffic flow cannot recognize the test vehicle and cause a crash. The reason is that the vehicle body collider is hidden in order to avoid the collision of the tire collider and the body collider when adding NWH vehic physics 2 to the test vehicle. thus vehicle body cannot be identified. To solve this problem we need[17]:

(1) Select all the colldiers on the vehicle (BoxColliders, SphereColliders, MeshColliders, etc.) and assign a custom layer to them - e.g. VehicleLayer.

(2) Untick Auto Setup Layer Mask on the WheelControllers attached to the vehicle. A layer mask dropdown will appear. Make sure to untick the VehicleLayer on that list.

# List of Figures

# List of Tables

# Bibliography


[1] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. DOI: 10.48550/ARXIV.1506.01497. URL: https://arxiv.org/abs/1506.01497.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640.

[3] G. Jocher. *YOLO-V5*. https://github.com/ultralytics/yolov5/blob/master/README.zh-CN.md.

[4] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh. *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. 2019. DOI: 10.48550/ARXIV.1911.11929. URL: https://arxiv.org/abs/1911.11929.

[5] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. "Scaled-YOLOv4: Scaling Cross Stage Partial Network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 13029–13038.

[6] P. Purkait, C. Zhao, and C. Zach. "SPP-Net: Deep absolute pose regression with synthetic views". In: *arXiv preprint arXiv:1712.03452* (2017).

[7] JingweiZhang12 and ZwwWayne. *open-mmlab/mmdetection3d*. https://github.com/open-mmlab/mmdetection3d.

[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. *PointPillars: Fast Encoders for Object Detection from Point Clouds*. 2018. DOI: 10.48550/ARXIV.1812.05784. URL: https://arxiv.org/abs/1812.05784.

[9] A. G. |. P. L. |. C. S. |. R. Urtasun. *KITTI-Dataset*. https://www.cvlibs.net/datasets/kitti/.

[10] A. I. Prof. Dr.-Ing. habil. Alois Knoll Robotics and T. U. M. Real-Time Systems. *Neurorobotics Platform*. https://www.neurorobotics.net/.

[11] *Unity-High Definition Render Pipeline(HDRP)*. https://docs.unity3d.com/cn/Packages/com.unity.render-pipelines.high-definition@10.4/manual/index.html.

[12] *Unity-HDRP demo*. https://unity.com/de/releases/2019-3/graphics.

[13] *NWH Vehicle Physics 2 Unity Package*. http://nwhvehiclephysics.com/doku.php/index.

[14] *Fantastic City Generator Unity Package*. https://assetstore.unity.com/packages/3d/environments/urban/fantastic-city-generator-157625.

[15] *FCG - Adding Vehicles on Traffic System - 02*. https://www.youtube.com/@masterpixel3d667.

[16] C. Knospe. "PID control". In: *IEEE Control Systems Magazine* 26.1 (2006), pp. 30–31.

[17] *NWH Vehicle Physics 2 Documentation _ Troubleshooting*. http://nwhvehiclephysics.com/doku.php/Troubleshooting.