ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету)

Кафедра прикладної математики та інформатики
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри ПМІ
_____ Наталія Маслова
(підпис) (ініціали, прізвище)
«____»_____2023 р.

Випускна кваліфікаційна робота

магістра
(освітньо-кваліфікаційний рівень)

на тему: «Симулятор системи нейромережевого керування автомобіля»

Виконав: студент 2 курсу, групи ІПЗм-22
(шифр групи)

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва напряму підготовки)

Цибулько О. І. _____
(прізвище та ініціали) (підпис)

Керівник проф. каф. ПМІ, д.т.н., Башков Є. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент зав. каф. АТ, к.т.н., доц. Поцепаєв В.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент в.о. зав. каф. ЕТ, к.т.н., доц. Ковальов С.О _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій випускній кваліфікаційній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Луцьк – 2023

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь магістр
Спеціальність (напрям підготовки) 121 Інженерія програмного забезпечення

(шифр і назва)

ЗАТВЕРДЖУЮ
в.о. завідувача кафедри ПМІ
_____ /Н. О. Маслова/
«____»_____2023 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Цибульку Олексію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Симулятор системи нейромережевого керування автомобіля
керівник роботи проф. каф. ПМІ, д.т.н., Башков Євген Олександрович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від « 28 » вересня 20 23 року № 446
2. Строк подання студентом роботи 10 грудня 2023 року
3. Вихідні дані до роботи результати переддипломної практики
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) аналіз сучасних систем автономного керування; 2) вибір інструментів для розробки дослідницького зразка і вбудовування моделей ШІ; 3) визначення вимог до додатка; 4) проєктування системи автономного керування автомобіля; 5) розробка і опис системи генерації тренувальних даних; 6) розробка і опис системи комунікації симулятора та модуля автономного керування; 7) розробка симулятора; 8) розробка модуля автономного керування; 9) тестування і оцінка розробки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) рисунки, таблиці, блоки програмного коду, діаграми.

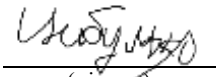
6. Консультанти розділів роботи

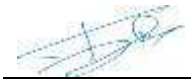
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Назарова І.А.		

7. Дата видачі завдання 4 жовтня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналітичний огляд предметної області, дослідження теоретичної бази, вивчення джерел інформації	04.10.23-17.10.23	
2	Аналіз сучасних систем автономного керування, вибір інструментів розробки	18.10.23-31.10.23	
3	Проєктування і розробка експериментального зразка програмної системи нейромережевого керування автомобіля	01.11.23-14.11.23	
4	Тестування та оцінка розробленої системи	15.11.23-30.11.23	
5	Оформлення пояснювальної записки та опис створеної системи. Проходження нормоконтролю	01.12.23-06.12.23	
6	Перевірка пояснювальної записки антиплагіатною системою. Оформлення презентації до роботи, графічного матеріалу та рецензування роботи	07.12.23-25.12.23	
7	Захист роботи	26.12.23	

Студент  **Цибулько О. І.**
(підпис) (прізвище та ініціали)

Керівник роботи  **Башков Є. О.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Цибулько О. І. Симулятор системи нейромережевого керування автомобіля / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 121 Інженерія програмного забезпечення. – ДВНЗ ДонНТУ, Луцьк, 2023.

Об’єктом дослідження – процес автоматичного керування рухомими засобами.

Предметом дослідження – системи автоматичного керування автомобіля на базі нейронних мереж.

Метою роботи є розробка симулятора системи нейромережевого керування автомобіля, яка дозволить моделювати процес руху автомобілю в невизначених раніше умовах.

Практичне значення розробки симулятора системи автоматичного керування автомобіля за допомогою нейронних мереж полягає в створенні функціональної нейронної мережі керування автомобілем для її подальшого використання як основи повноцінної системи керування автомобілем.

Методи дослідження базуються на основних положеннях теорії нейронних мереж, теорії обчислювальних систем, теорії цифрової обробки зображень.

Ключові слова: нейронна мережа, згорткова нейронна мережа, цифрова обробка зображень, симулятор, нейрон, шар, модель, Python, Tensor Flow

ABSTRACT

Tsybulko O. O. I. Tsibulko Simulator of the neural network control system of the car / Graduation qualification work for obtaining an educational degree “Master” in specialty 121 “Software Engineering”. – DonNTU, Lutsk, 2023.

The object of research is the process of autonomous control of moving vehicles.

The subject of research is autonomous car control systems based on neural networks.

The purpose of the work is to develop a simulator of the neural network control system of the car, which will allow simulating the process of driving a car in previously undefined conditions.

The practical significance of developing a simulator of an automatic car control system using neural networks is to create a functional car control neural network for its further use as the basis of a full-fledged car control system.

Research methods are based on the basic principles of the theory of neural networks, the theory of computer systems, and the theory of digital image processing.

Keywords: neural network, convolutional neural network, digital image processing, simulator, neuron, layer, model, Python, Tensor Flow

ЗМІСТ

ВСТУП 9

РОЗДІЛ 1 АНАЛІЗ ПІДХОДІВ ДО ПРОЄКТУВАННЯ НЕЙРОННИХ МЕРЕЖ КЕРУВННЯ АВТОМОБІЛЕМ 10

1.1 Огляд проблеми 10

1.2 Сучасні підходи до розробки автономних транспортних засобів 12

1.2.1 Системи безпілотного керування на основі нейронних мереж 12

1.2.2 Системи безпілотного керування без нейронних мереж 14

1.3 Симулятори систем автоматичного керування автомобілем 15

1.4 Висновки за розділом 16

РОЗДІЛ 2 ОГЛЯД АРХІТЕКТУР РОЗРОБКИ НЕЙРОННИХ МЕРЕЖ 17

2.1 Згорткові нейронні мережі 17

2.2 Архітектура системи автоматичного керування 21

2.2.1 Існуючі топології нейронних мереж для автономного керування 22

2.3 Keras 26

2.4 Unity 26

2.5 Spider 27

2.6 Visual Studio 28

2.7 Висновки за розділом 29

РОЗДІЛ 3 СИМУЛЯТОР СИСТЕМИ АВТОНОМНОГО КЕРУВАННЯ 30

3.1 Умови симуляції 30

3.2 Проєктування людино-машинний інтерфейс програми симулятора 30

3.3 Побудова програмного застосунку 34

3.3.1 Використані сторонні ресурси 34

3.3.2 Головне меню симулятора 37
3.3.3 Сцени для автономного керування 38
3.3.4 Сцени для генерації тренувального набору 44
3.4 Висновки за розділом 49
РОЗДІЛ 4 МОДУЛЬ АВТОНОМНОГО КЕРУВАННЯ 50
4.1 Система обміну даними 50
4.2 Структура нейронної мережі системи автономного керування 51
4.3 Програмна реалізація модулів автономного керування 56
4.3.1 Програмний модуль для створення Keras моделі 57
4.3.2 Програмний модуль опису структури нейронної мережі 60
4.3.3 Програмний модуль навчання нейронної мережі 60
4.4 Модуль автономного керування автомобілем 65
4.5 Висновки за розділом 67
РОЗДІЛ 5 НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ 68
5.1 Налаштування параметрів симулятора 68
5.2 Формування набору даних 68
5.3 Вибір гіперпараметрів нейронної мережі 69
5.3.1 Розмір пакету 69
5.3.2 Швидкість навчання 71
5.3.3 Кількість епох 75
5.4 Тестування найкращої моделі 77
5.5 Висновки за розділом 84
ВИСНОВКИ 85
ПЕРЕЛІК ПОСИЛАНЬ 87
Додаток А Зауваження нормоконтролера 92

Додаток Б Графічна частина 93

Додаток В.1 Програмний код програми симулятора 103

Додаток В.2 Програмний код тренування нейронної мережі 121

Додаток В.3 Програмний код модуля автономного керування..... 124

ВСТУП

Аналіз статистичних даних показує, що з роками кількість автомобілів на дорогах зростає, що призводить до збільшення кількості дорожньо-транспортних пригод і підвищення загрози для всіх учасників дорожнього руху.

Сьогодні автовиробники оснащують свої автомобілі безліччю комп'ютерних систем для покращення комфорту й безпеки як водія, так і його оточення. Таке програмне забезпечення допомагає в різних аспектах: утримувати увагу, паркування, триматися потрібної смуги руху, будувати найзручніші маршрути тощо. Основні розробки світового ринку мінімізують помилки через людський фактор і роблять водіння більш безпечним:

– driver monitoring system – система з використанням камер для контролю уваги водія;

– collision avoidance – система запобігання зіткненню за допомогою маневрування, гальмування або прискорення;

– highway pilot – автоматичне керування на певних відрізках шосе.

Вже сьогодні компанія Uber використовує безпілотні таксі в певних містах США. Інша компанія Waymo у своїх безпілотних авто використовує архітектуру з глибоким навчанням VectorNet яка прогнозує траєкторії транспортних засобів за допомогою графової нейронної мережі.

З огляду на дану галузь, можна зробити висновок, що застосування в ній нейронних мереж є досить обширним. Дана робота буде сфокусована на розробці нейронної мережі для автономного керування автомобілем.

Об'єкт дослідження – процес проєктування системи автоматичного керування автомобіля розробленої за допомогою нейронних мереж.

Мета дослідження – розробка нейронної мережі для керування автомобілем.

РОЗДІЛ 1
АНАЛІЗ ПІДХОДІВ ДО ПРОЄКТУВАННЯ НЕЙРОННИХ МЕРЕЖ КЕРУВННЯ АВТОМОБІЛЕМ

Автономні транспортні засоби не просто забезпечать комфорт, але і позбавлять нас від аварій, спричинених втомленими, нетверезими або розсіяними водіями. Звільнивши мандрівників від обов'язку бути за кермом, безпілотні автівки підвищать продуктивність пасажирів, дозволяючи їм працювати під час поїздки. Із загальним поширенням безпілотників вони також спростять рух транспорту та скоротять час подорожі, усуваючи транспортні затори, що виникають через ДТП, "тягучки" і нагромадженість. Це можливо завдяки здатності безпілотників передавати інформацію про дорожні умови та рух на дорогах один одному. Повертаючи пасажирам час, який зараз вони витрачаємо на водіння, та гарантуючи безпеку. Безпілотні автомобілі принесуть значний внесок у благополуччя людства. Проте, перш ніж довірити своє життя безпілотним автомобілям, необхідно переконатися, що вони здатні впоратися зі складними викликами реального дорожнього руху.

1.1 Огляд проблеми

Люди успішно керують автомобілями завдяки своїй інтуїції. Людський мозок здатен обробляти великі обсяги інформації, виділяти суттєве і використовувати це для прийняття рішень. Водіння традиційно було вкрай складною задачею для комп'ютерів через широкий спектр можливих сценаріїв і відсутність інтуїтивного розуміння. Проте люди без зусиль можуть виявити перешкоди, розмітку доріг і інші транспортні засоби, що представляє велику складність для комп'ютерів. Проблеми імітації людського сприйняття можуть бути вирішені за допомогою алгоритмів машинного навчання. Навіть якщо алгоритми машинного навчання обіцяють значний прогрес у сфері

автономного водіння, вони все ж мають свої власні труднощі. Однією з таких проблем є необхідний обсяг даних для навчання.

Збирання величезної кількості даних, необхідних для створення ефективного контролера для автомобіля, є складною інженерною задачею. Для ефективного керування автономним транспортним засобом в різних сценаріях. Контролер повинен мати дані, які були зібрані людиною для конкретного сценарію. Саме через це водіння завжди було виключною сферою людей. Люди вражаюче добре адаптуються до різних рівнів освітлення, погодних умов та дорожніх ситуацій. Адаптація до різноманітних умов, які можуть виникати на дорозі, є складною задачею для комп'ютерів, і цю задачу найкраще розв'язувати за допомогою алгоритму, який може навчатися та узагальнювати інформацію так само, як і люди.

Автономні транспортні засоби розробляються з використанням комплексу алгоритмів машинного навчання та інших методів. Ці алгоритми в значній мірі ґрунтуються на даних, які накопичуються водіями-людьми, і потребують багаторівневого підходу для перетворення цих даних на керуючі команди для автомобілів. Такі дані часто включають в себе відеопотоки, зібрані з різних бортових сенсорів, таких як камери, лідар, радар та інфрачервоні датчики. Ця інформація може бути використана для калібрування та навчання алгоритмів, фактично служачи як досвід, з якого можна вчитися. Зміни в навколишньому середовищі, такі як нові типи доріг або погодні умови, потребують збору сотень годин даних, їх обробки та маркування для навчання алгоритмів і повторного тренування системи. Зміни в апаратному забезпеченні, такі як нові сенсори, вимагають адаптації алгоритмів та проміжних рівнів, які перетворюють високорівневі алгоритмічні виходи в низькорівневі команди для керування.

Мета даної роботи полягає у вирішенні викликів автономного водіння, застосовуючи метод навчання на основі реалістичних даних моделювання та застосовуючи всебічний підхід до навчання. Нейронні мережі функціонують як обчислювальні моделі мозку та можуть навчатися аналогічним чином. Для

досягнення навичок керування автомобілем нейронною мережею були використані дані з симуляції. Цей підхід гарантує ефективність методу, підтверджену експериментами на тестовій системі автомобіля, що працює в середовищі, що наближається до реального.

1.2 Сучасні підходи до розробки автономних транспортних засобів

Безпілотні автомобілі користуються рядом пристроїв, включаючи камери, LiDAR, RADAR, GPS і датчики інерції, для збору інформації про своє оточення. Подальше оброблення цих даних здійснюється за допомогою спеціальних алгоритмів, відомих як алгоритми глибокого навчання, для отримання розуміння ситуації навколо автомобіля. На основі цього безпілотні автомобілі приймають важливі рішення, спрямовані на здійснення безпечного та плавного руху.

Згорткові нейронні мережі (CNN) широко використовуються в автономних автомобілях завдяки здатності аналізувати просторову інформацію, зокрема інформацію із зображень. CNN добре розпізнають особливості на зображеннях, через що вони є найкращим вибором для визначення різних аспектів оточуючого середовища. У CNN зі збільшенням кількості шарів мережі різні рівні спеціалізуються на виявленні різних шаблонів. Перші шари спроможні виявляти прості структури, такі як контури, тоді як більш глибокі шари виявляють складніші образи, такі як форми об'єктів. Ця здатність адаптуватися є однією з ключових причин, чому CNN важливі для безпілотних автомобілів.

1.2.1 Системи безпілотного керування на основі нейронних мереж

HydraNet [1, 2], розроблений Tesla, є динамічною архітектурою, яка була представлена Раві та іншими дослідниками в 2018 році, і в основному спроектована для використання в безпілотних автомобілях для семантичної сегментації. Її основною метою є покращення ефективності обчислень під час логічного аналізу. Концепція HydraNet передбачає існування різних мереж

CNN, які називаються філіями і призначені для виконання конкретних завдань. Кожна гілка отримує різні вхідні дані, і мережа може вибірково визначити, які гілки активувати, після чого об'єднати виходи з різних філій для прийняття остаточного рішення. Ця структура зображена на рисунку 1.1.

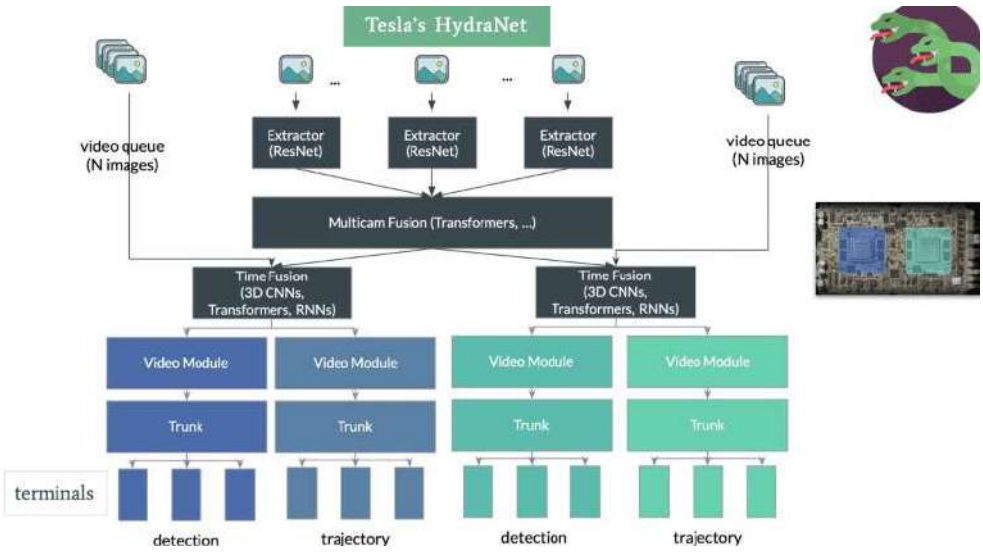


Рисунок 1.1 – HydraNet (Джерело: https://www.thinkautonomous.ai/blog/how-tesla-autopilot-works/)

ChauffeurNet [3] – це нейронна мережа, яка базується на рекурентних нейронних мережах (RNN) і використовується компанією Google Waymo для навчання автономних автомобілів шляхом імітаційного навчання. В основному використовується RNN для генерації траєкторій руху, але ChauffeurNet також містить компонент згорткової нейронної мережі (CNN), який називається FeatureNet. Ця мережа для вилучення контекстних ознак використовується спільно з іншими нейронними мережами сприйняття (рисунок 1.2).

Концепція ChauffeurNet полягає в навчанні самого автономного транспорту за допомогою імітації дій досвідчених водіїв. Для подолання обмежень, пов'язаних з нестачею реальних навчальних даних, автори статті «ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst» використовують синтетичні дані. Ці синтетичні дані містять різні

аномалії, такі як відхилення від заданої траєкторії, додавання перешкод і створення нереалістичних сценаріїв. Навчання автомобіля за допомогою синтетичних даних виявилося більш ефективним, ніж використання лише реальних даних.

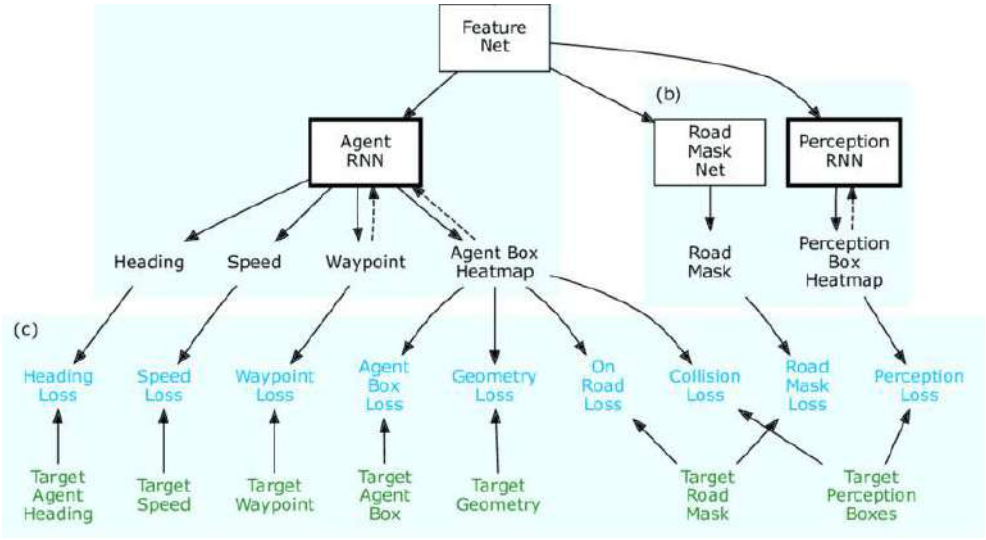


Рисунок 1.2 – Тренування ChauffeurNet (Джерело: https://www.researchgate.net/figure/Training-the-driving-model-a-The-core-ChauffeurNet-model-with-a-FeatureNet-and-an_fig1_329525538)

1.2.2 Системи безпілотного керування без нейронних мереж

Один з найвідоміших прикладів дослідження в області автономного водіння – це DARPA Grand Challenge [4]. Це змагання автомобілів-роботів, яке фінансується урядом Сполучених Штатів. Основною метою цих змагань є створення повністю автономних транспортних засобів. Організатором цього конкурсу є Управління перспективних дослідницьких проектів Міністерства оборони США. Перший DARPA Grand Challenge відбувся в 2004 році і вимагав від учасників створити транспортні засоби, які могли б пройти відстань у пустелі Мохаве повністю автономно. У першому змаганні жоден з учасників не зміг успішно пройти цю дистанцію. Проте в 2005 році DARPA провело другий Grand Challenge [5], який був успішно завершений, і перемогу отримав автомобіль Stanley від Stanford Racing Team.

Ні один з автомобілів не використовував підходи машинного навчання для керування автомобілем. Натомість керування транспортними засобами було реалізовано за допомогою звичайних алгоритмів, використовуючи дані від різних датчиків, таких як камери, GPS і лідар. Зазвичай алгоритми машинного навчання використовуються для обробки зображень у завданнях автономного водіння. У випадку DARPA Grand Challenge, використання камер було менш ефективним, ніж зазвичай, через особливості рельєфу місцевості. У першому DARPA Grand Challenge траса була повністю бездоріжжям і в більшій частині мала нерівну місцевість. Оскільки значна частина дороги і навколишнього ландшафту мала коричневий колір, це ускладнювало розпізнавання кольорових відтінків. Таким чином, надійність даних з камери була меншою, ніж у випадку радару і лідару.

У 2007 році DARPA провело друге велике змагання, DARPA Urban Grand Challenge. Urban Grand Challenge проходив на авіабазі Джордж у Вікторвіллі, штат Каліфорнія, і охоплював 60 миль міської дороги. Команди отримали аерофотознімки регіону змагання. Команда Tartan з Університету Карнегі-Меллона взяла участь у змаганнях з роботом, побудованим на шасі Chevrolet Tahoe. Робот Team Tartan використовував зображення камери, лідар і радар як вхідні дані для своїх алгоритмів керування. Як і в минулих випробуваннях, лідар і радар відіграли велику роль у процесах прийняття навігаційних рішень. Однак у цьому випадку зображення камери використовувалися для базової функції відстеження лінії. Команда Tartan фактично використовувала згорткову нейронну мережу, але лише для обробки аерофотознімків, данні з якої пізніше використовувалися для високорівневого планування шляху.

1.3 Симулятори систем автоматичного керування автомобілем

Загалом, для симуляції систем автоматичного керування автомобілем та самих автомобілів і їх оточення є два підходи.

Перший, створення симулятора власноруч. Цей підхід має очевидні переваги:

– Створення саме тієї системи в якій необхідно провести випробовування;

– Простота зміни параметрів та умов симульованої системи;

– Простий збір та моніторинг будь-яких параметрів.

Головним недоліком є часові витрати на розробку симулятора. Хоча ця проблема може виявитися не такою серйозною при використанні ігрових рушіїв таких, як Unity чи Unreal Engine. Вони містять готові варіанти автомобілів з реалізованою фізикою поведінки та треки для них.

Другий, використати для випробувань існуючу комп'ютерну гру. Комп'ютерні ігри досить різноманітні, тому пошук симуляції з необхідними умовами (населений пункт з живими вулицями чи порожній гоночний трек) не повинен викликати складнощів.

Перевагами даного підходу є:

– Незначні витрати часу на пошук необхідного симулятора;

– Всі аспекти симулятора виконані на високому рівні.

Недоліком такого підходу є відсутність гнучкості в налаштуванні симульованої системи. Популярним вибором гри для симуляції є GTA V через те, що вона надає безліч умов для проведення тестів в наближеному до реального світу оточенні.

1.4 Висновки за розділом

Як висновок можна сказати, що зі своїм розвитком, нейронні мережі мають все більше галузей застосування. У галузі безпілотного керування вони застосовуються як для обробки вхідних даних з камер та різних датчиків, так і для керування транспортом. Для безпечної перевірки працездатності системи, її розміщують в симуляцію розроблену для тестування окремих її частин або системи в цілому.

РОЗДІЛ 2
ОГЛЯД АРХІТЕКТУР РОЗРОБКИ НЕЙРОННИХ МЕРЕЖ

2.1 Згорткові нейронні мережі

Вперше математична модель штучного нейрону була запропонована нейрофізіологом Уореном Маккалохом та математиком Уолтером Пітсом разом з моделлю нейронної мереж в 1943 році [6]. Їх активне використання як складової штучного інтелекту почалося лише протягом останніх двадцяти років. Структурні варіанти нейронних мереж різняться від простих одношарових перцептронів до глибоких згорткових нейронних мереж. Представити нейронну мережу найпростіше у вигляді графу. Нейрон можна розглядати як вузол на графу, який приймає один або декілька числових вхідних сигналів, застосовує до них певну арифметичну функцію та генерує вихідний результат. Функція, яку нейрон застосовує до своїх вхідних даних, називається «функцією активації». Ця назва походить від того, що функція контролює рівень активності нейрона. Одношаровий перцептрон, який показаний на рисунку 2.1, є найпростішою формою штучної нейронної мережі, також відомою як "ШНМ", оскільки він включає лише один нейрон.

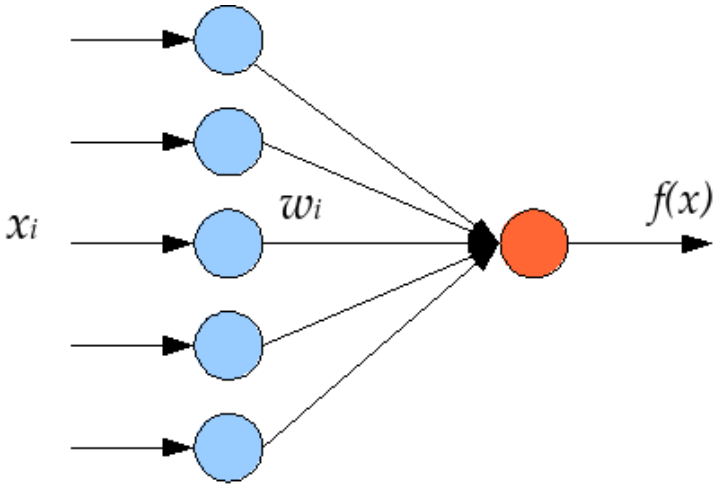


Рисунок 2.1 – Одношаровий перцептрон (Джерело: https://uk.wikipedia.org/wiki/Перцептрон)

Множина X являє собою сигнал на входах нейрона, множина W – ваги виходів. Щоб визначити вихід нейрона, числові значення x_i перемножуються на вагу значення, w_i. Отримані значення потім передаються нейрону, який представлений помаранчевим колом. Нейрон виконає підсумовування зважених вхідних даних, які він отримує.

Один з способів обробки вихідних даних нейронної мережі полягає в установленні кількості нейронів на вихідному шарі, рівної кількості категорій, які мережа повинна ідентифікувати. У цьому випадку кожен окремий нейрон відповідає конкретному класу виходу і активується, якщо мережа вважає, що вхідні дані відносяться до цього класу.

Згорткові нейронні мережі (CNN) є підкласом нейронних мереж із прямим зв'язком, призначеним для обробки двовимірних даних, таких як зображення. Вони схожі на звичайні нейронні мережі, складаються з нейронів, кожен з яких має вагу та зміщення. Нейрони в CNN отримують вхідні дані, виконують точковий добуток, за яким слідує нелінійність, і далі застосовують функцію втрати в шарі класифікації.

Основна відмінність між CNN і звичайними мережами із прямим зв'язком полягає в тому, що CNN ефективніше працюють з двовимірними вхідними даними, тому їх основне використання пов'язане з обробкою та класифікацією зображень. Зазвичай CNN починають роботу зі згорткового шару, який бере вхідні зображення і виділяє різні ознаки об'єктів, такі як краї, лінії, криві тощо (рисунок 2.2).

Рисунок 2.2 – Структура згорткової нейронної мережі (Джерело : https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/)

Згортковий шар є основою структури CNN, і він включає більшість операцій, що застосовуються в мережі CNN. Згортковий шар в основному здійснює математичну операцію, відому як згортка, яка включає у себе множення матриці ядра/фільтру на вхідні дані, після чого зважені вхідні дані підсумовуються, що показано на рисунку 2.3.

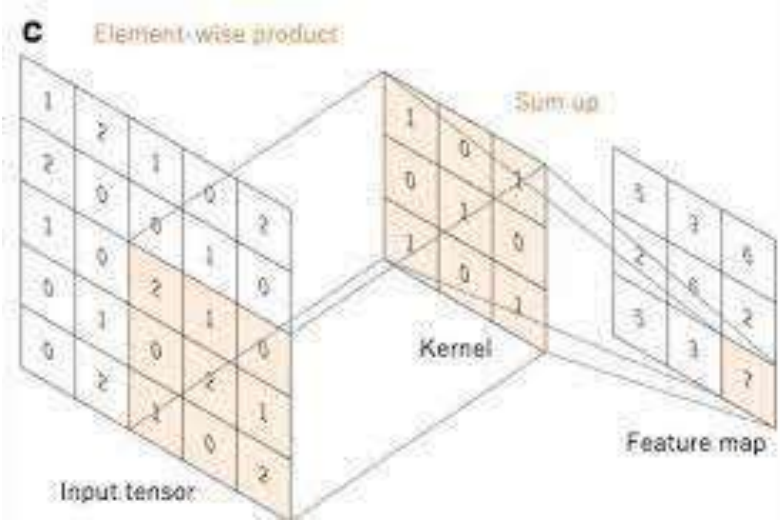


Рисунок 2.3 – Згортка матриці (Джерело: https://ela.kpi.ua/bitstream/123456789/45474/1/Tsuprun_bakalavr.pdf)

Функції активації застосовуються для забезпечення нелінійності та фільтрації необхідної інформації від непотрібної. За допомогою цієї функції обробляється кожен піксель вхідного зображення. Найбільш популярними функціями активації є Sigmoid, TanH, ReLU.

Sigmoid – це неперервно диференційована монотонна нелінійна S-подібна функція, яка часто застосовується для апроксимації значень деякої величини. Сигмоїда – нелінійна за своєю природою, а комбінація таких функцій робить також нелінійну функцію, що дає можливість складати шари. Інша перевага сигмоїди над лінійною функцією полягає в її діапазоні значень [0,1]. Така властивість сигмоїди дуже корисна, тому що не призводить до помилок у разі великих значень активації. Часто під сигмоїдою розуміють логістичну криву (рисунок 2.4), яка визначається формулою $f(x) = \frac{1}{1+e^{-x}}$.

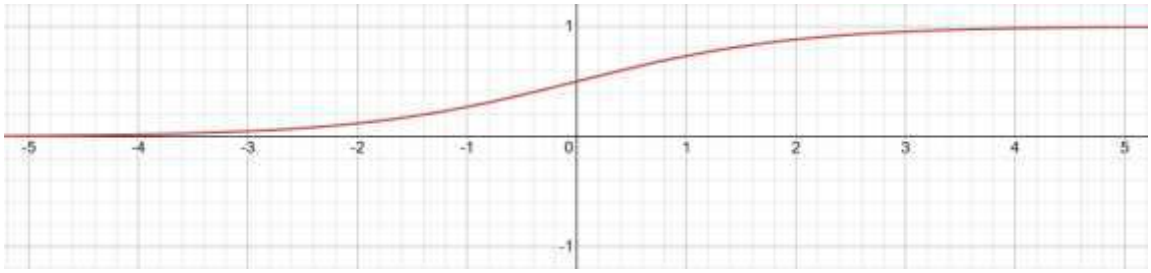


Рисунок 2.4 – Sigmoid

Гіперболічний тангенс (TanH) дуже схожий на сигмоїду, тому вони мають схожі характеристики. Її природа є нелінійною, вона добре підходить для комбінації шарів, а діапазон значень функції (-1, 1). Однак варто відзначити, що градієнт тангенціальної функції більше, ніж у сигмоїди. Гіперболічний тангенс виражається формулою $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (рисунок 2.5).

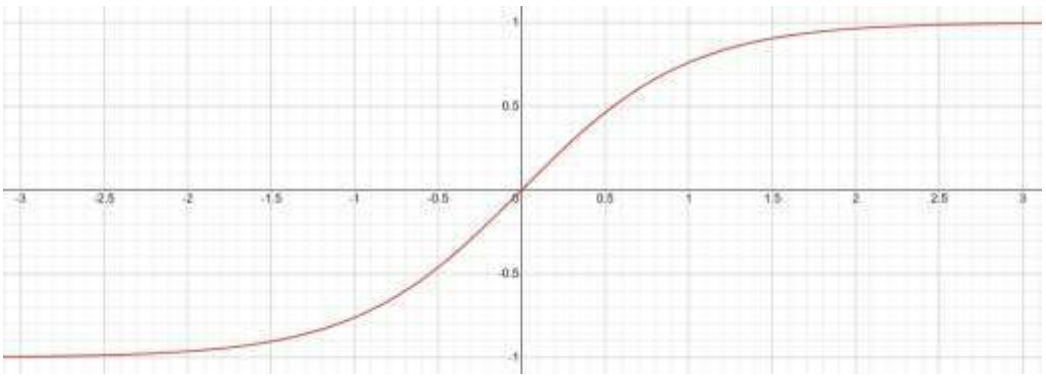


Рисунок 2.5 – TanH

ReLU є хорошим апроксиматором, оскільки будь-яка функція може бути апроксимована комбінацією ReLU. Область допустимих значень ReLU [0, inf), тобто активація може "вибухнути". перевагою цієї функції є розрідженість активації. У великих мережах з безліччю нейронів використання сигмоїди або гіперболічного тангенсу призведе до активації всіх нейронів аналоговим способом. Це означає, що майже всі активації мають бути оброблені для опису виходу мережі. Інакше кажучи, активація щільна, але це затратно. Бажано, щоб

деякі нейрони не були активовані, це зробило б активації розрідженими та ефективними. ReLU виражається формулою $f(x) = \max(0, x)$ (рисунок 2.6).

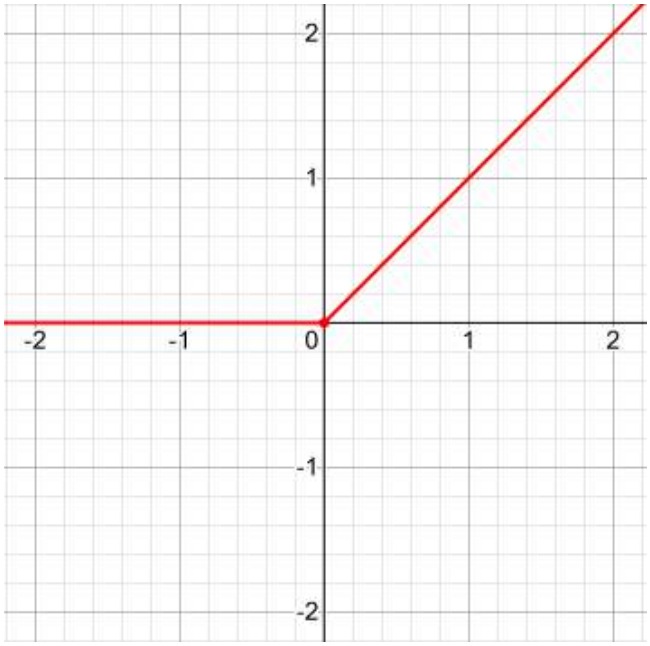


Рисунок 2.6 – ReLU

В подальшому, при розробці нейронної мережі, буде використано саме цю функцію активації через її простоту та швидкість обчислення.

2.2 Архітектура системи автоматичного керування

За останні кілька років було представлено безліч архітектур згорткових нейронних мереж, спрямованих на покращення точності та продуктивності. Класифікація зображень є цікавою і однією з найскладніших задач у галузі комп'ютерного зору. В цій задачі розв'язувач повинен або визначити, що зображено на фотографії, ідентифікувати об'єкти на зображенні, або згрупувати зображення з схожими характеристиками в однорідні категорії. Здається, що для успішного вирішення цих завдань найчастіше використовуються згорткові нейронні мережі. Щорічно проводиться ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [9], де учасники змагаються

з власноруч розробленими алгоритмами згорткових нейронних мереж для класифікації зображень з бази даних ImageNet.

2.2.1 Існуючі топології нейронних мереж для автономного керування

End-to-end (e2e) системи, засновані на архітектурі Deep Neural Network (DNN), використовуються у сфері автономного водіння для моделювання складних взаємозв'язків, отриманих з інформації, отриманої від системи сприйняття автомобіля. Це досягається за допомогою різних видів нейронних блоків, які об'єднуються в рівні (наприклад, згорткові шари, повнозв'язані шари, рекурентні шари і т. д.), з метою безпосередньо контролювати кермо, акселератор і гальмо автомобіля. Ці дії на елементи керування транспортним засобом можуть бути категоричними, наприклад, збільшення або зменшення швидкості, або вони можуть генерувати конкретні команди, такі як поворот на 13,6 градусів або досягнення швидкості 45 км/год. Алгоритми машинного навчання, які використовуються для моделювання керування автомобілем, відносяться до категорії, відомої як «кероване навчання». Ці алгоритми засновані на знаннях, отриманих з набору даних, зазвичай зібраних під час тестів на реальному автомобілі [10] або в симуляторах водіння [11]. Ці набори даних містять інформацію, отриману від систем сприйняття, таких як зображення (кольорові або інфрачервоні), дані з лідару, радару, IMU, а також інформацію про дії водія (кут повороту керма, сила натиску на педаль газу чи гальма).

Архітектура SiD-e2e використовує єдине джерело даних для вхідного рівня з метою генерації значень для керування транспортним засобом. Моделі SiD використовують візуальну інформацію, отриману від однієї або декількох камер транспортного засобу, для створення єдиного зображення поля зору транспортного засобу як вхідний візуальний сигнал для мережі [12]. Перед подальшою обробкою DNN зображення піддаються зменшенню розміру та нормалізації. Далі зображення проходять через згорткові шари із різними параметрами ядра (k × k) та глибиною (d), які автоматично виділяють важливі

характеристики зображення через послідовну обробку в різних шарах. Після проходження згорткових шарів отриманий вектор перетворюється у одновимірний вектор (F-рівень) і поєднується з набором повнозв'язаних шарів (FC), які здатні приймати рішення. Нарешті, FC-шари завершуються кількістю нейронів, яка відповідає кількості змінних, що потрібно передбачити. На рисунку 2.7 наведено приклад структури SiD, де нормалізоване зображення подається через групу згорткових шарів із різними параметрами ядра, за якими слідують набори повнозв'язаних шарів та остаточний вихідний рівень.

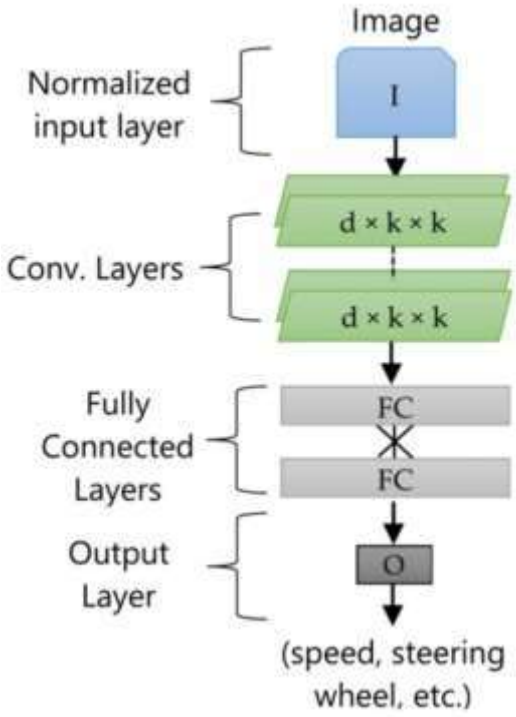


Рисунок 2.7 – Архітектура SiD-e2e (Джерело: https://www.researchgate.net/figure/Single-data-e2e-architecture-SiD_fig2_351856725)

Архітектура MiD-e2e відкриває можливість об'єднання різноманітних джерел даних транспортного засобу, таких як радар, поздовжні та поперечні прискорення, кутові швидкості, карти або GPS, разом із візуальною інформацією з камер автомобіля. Включення більш широкого спектра джерел інформації у DNN спрямоване на покращення продуктивності моделі,

поліпшення прогнозування конкретних ситуацій або ненормального водіння, підвищення стійкості до виникнення помилок [13]. Цей тип архітектури об'єднує результати SiD-e2e і набір FC-шарів, що дозволяє інтегрувати характеристики з інших джерел даних транспортного засобу на один рівень, який об'єднує всю цю інформацію, що зображено на рисунку 2.8.

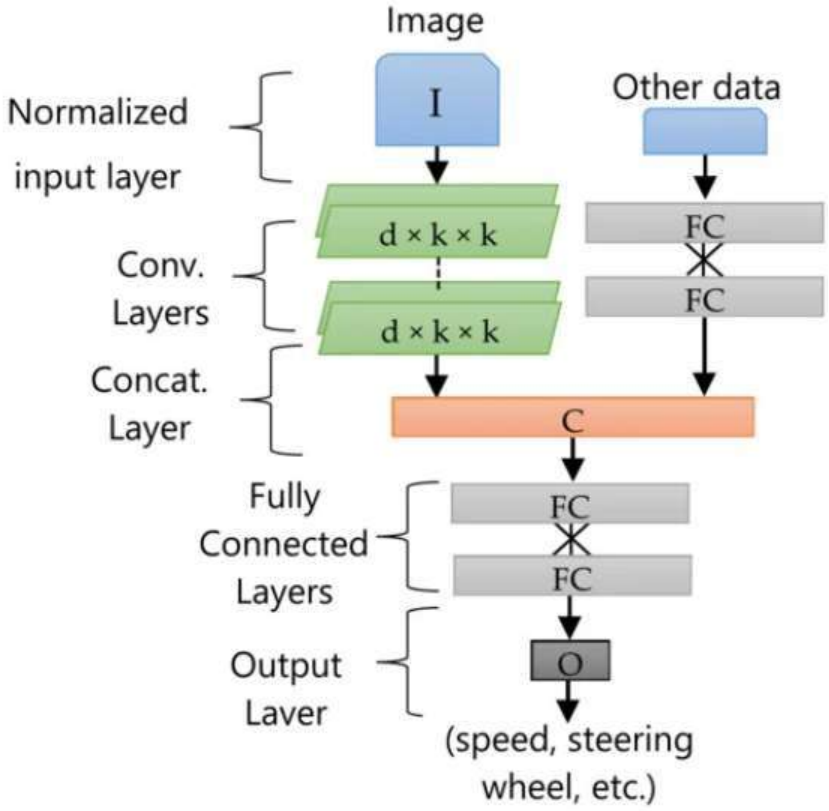


Рисунок 2.8 – Архітектура MiD-e2e (Джерело: https://www.researchgate.net/profile/Pedro-Navarro-Lorente/publication/351856725/viewer/AS:1027626844057606@1622016740905/background/7.png)

Архітектура SeD-e2e, що працює з послідовними даними, спрямована на моделювання часових зв'язків у даних за допомогою нейронних одиниць, які мають зворотний зв'язок. Ці типи нейронних мереж відомі як рекурентні нейронні мережі (RNN). Базові RNN можуть вивчати короткострокові залежності в даних, але вони стикаються з проблемами у захопленні довгострокових залежностей через проблему зі зникаючим градієнтом [14].

Щоб вирішити ці проблеми зі зникаючим градієнтом, були розроблені більш складні архітектури RNN, які використовують функції активації, що базуються на блоках стробування. Блок стробування має здатність умовно визначати, яка інформація буде запам'ятовуватися, забуватися або проходити через нього (рисунок 2.9). Прикладами таких типів архітектур RNN є Long Short-Term Memory (LSTM) і Gated Recurrent Unit (GRU).

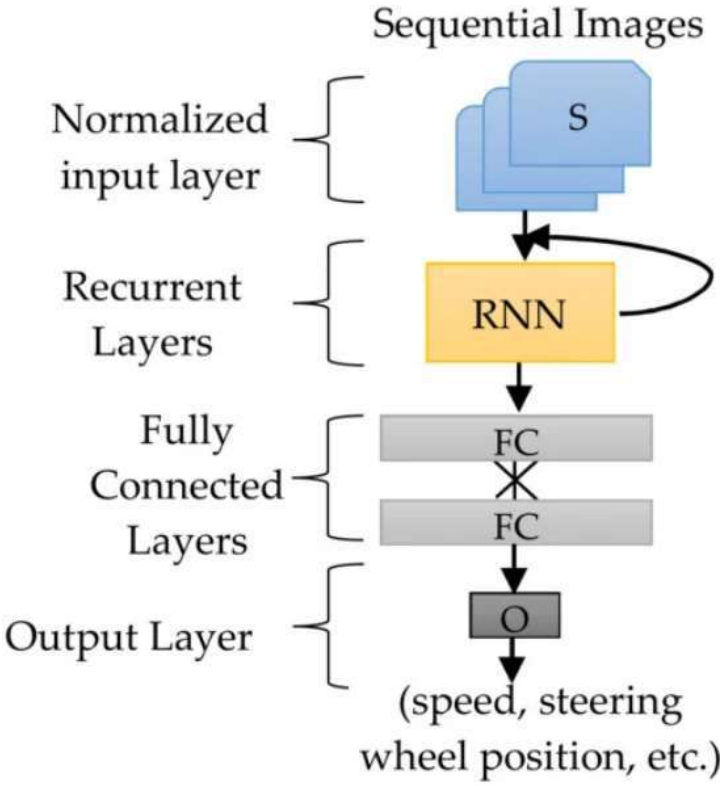


Рисунок 2.9 – Архітектура SeD-e2e (Джерело: https://www.researchgate.net/profile/Pedro-Navarro-Lorente/publication/351856725/viewer/AS:1027626844057606@1622016740905/background/7.png)

2.3 Keras

Для розробки нейронних мереж найкращим варіантом є Keras [7] для TensorFlow – відкрита нейромережна бібліотека, написана мовою Python. Keras призначена для спрощення швидких експериментів з мережами глибокого навчання. Основний акцент робиться на зручності використання, модульності та можливості розширення. У Keras знаходяться різні реалізації ключових будівельних блоків нейромереж, такі як шари, функції втілення, функції втілення передачі, оптимізатори та різноманітні інструменти для полегшення обробки зображень та тексту. Це спрощує процес кодування, необхідний для створення глибоких нейромереж.

Keras надає таку ж функціональність як і Tensorflow з набагато більш доступним та простим у використанні інтерфейсом. Крім того, Keras API включає в себе різноманітні популярні функції активації, такі як "relu" та "linear", а також "sigmoid", "tanh", "softmax" та інші.

2.4 Unity

Unity [8] – універсальний інструмент для створення відеоігор і додатків, а також є рушієм, на якому вони функціонують. Програми, розроблені з використанням Unity, працюють на різних платформах, включаючи настільні комп'ютери, мобільні пристрої та ігрові консолі, і можуть відображати графіку як у двовимірному, так і у тривимірному форматі, а також на пристроях віртуальної чи доповненої реальності. Застосунки, створені в Unity, підтримують технології DirectX та OpenGL. Редактор Unity відзначається простим і настраюваним інтерфейсом, який складається з різних вікон, що дозволяє легко налаштовувати та відлагоджувати гру безпосередньо у редакторі. Двигун також підтримує мову програмування C#, а для розрахунків фізики використовує фізичний двигун PhysX від NVIDIA.

Unity Asset Store - це інтернет-магазин, де можна придбати різноманітні ресурси для Unity, такі як тривимірні моделі, аудіофайли та музика, набори елементів інтерфейсу користувача, шейдери та частинки, набори спрайтів, а

також різноманітні інструменти. Майже у кожному проекті Unity можна знайти ресурси, які полегшують і прискорюють процес розробки, роблячи його більш комфортним.

В даній роботі цей ігровий рушій є вигідним на фоні інших варіантів через свою популярність та простоту в роботі. Вибір саме ігрового рушія, а не готової гри, надає велику гнучкість при налаштування параметрів симулятора. Графічний інтерфейс Unity зображено на рисунку 2.10.

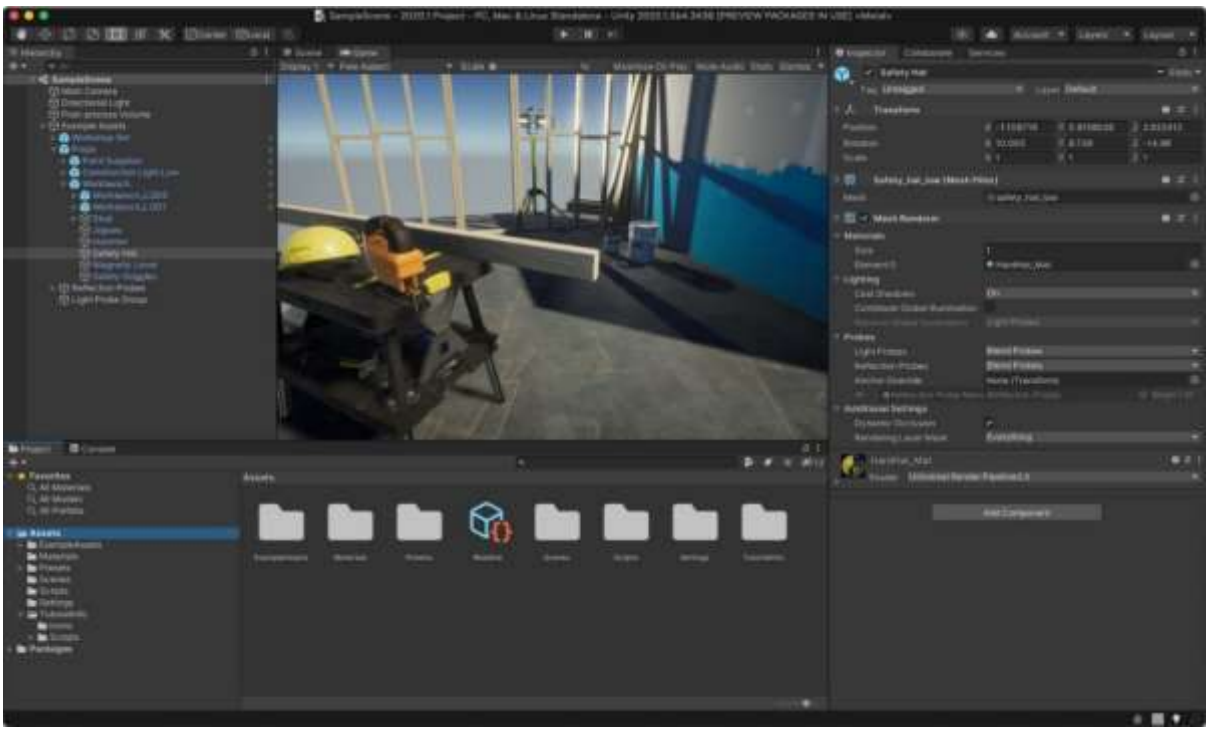


Рисунок 2.10 – Графічний інтерфейс ігрового рушія

2.5 Spider

Spyder — це безкоштовне наукове середовище з відкритим кодом, написане мовою Python для Python і розроблене вченими, інженерами та аналітиками даних. Він характеризується унікальним поєднанням розширених функцій редагування, аналізу, налагодження та профілювання комплексного інструменту розробки з дослідженням даних, інтерактивним виконанням, глибокою перевіркою та чудовими можливостями візуалізації наукового

пакету. Spyder чудово працює з Anaconda, надаючи можливість інтеграції редактора з віртуальним середовищем conda.

Python - високорівнева інтерпретована мова програмування, що визначається своїм простим читабельним синтаксисом. Це робить її популярною серед початківців і забезпечує швидку розробку. Python підтримує різні парадигми програмування та має широкий набір бібліотек. Велика та активна спільнота розробників сприяє обміну досвідом та розв'язанню проблем. Мова широко використовується в різних галузях, включаючи веб-розробку, наукові дослідження, штучний інтелект і обробку даних. На рисунку 2.11 зображено описане середовище.

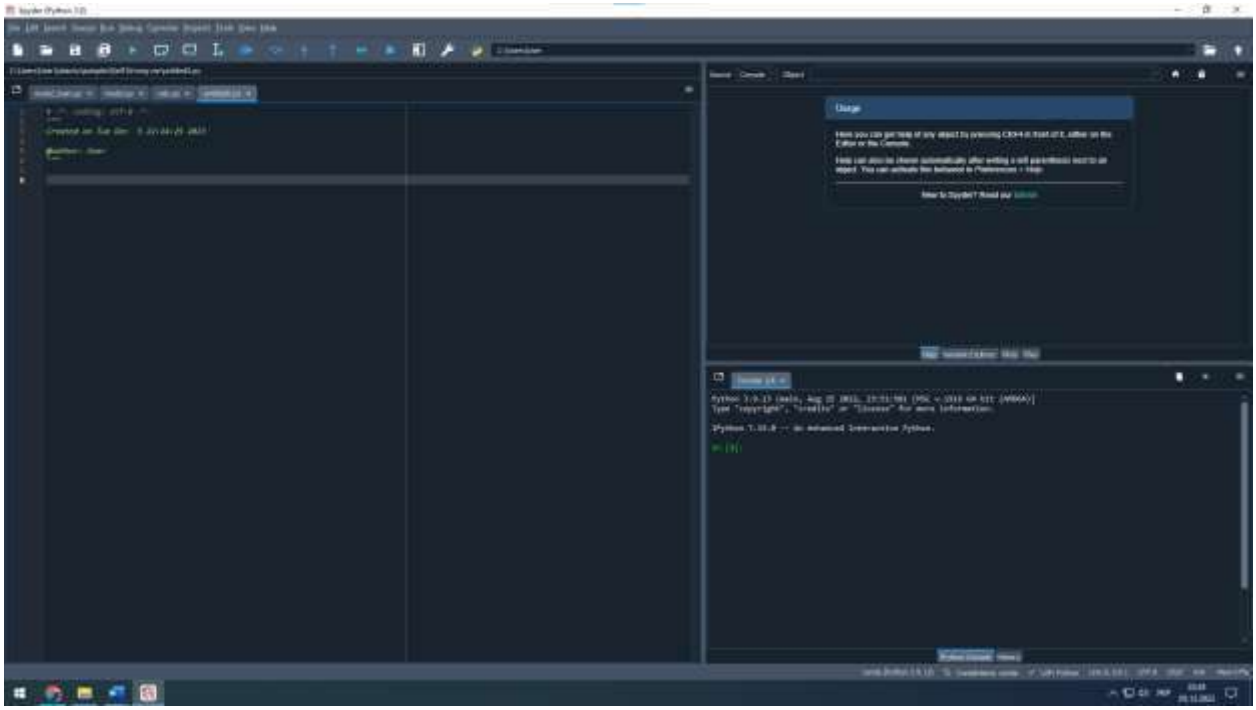


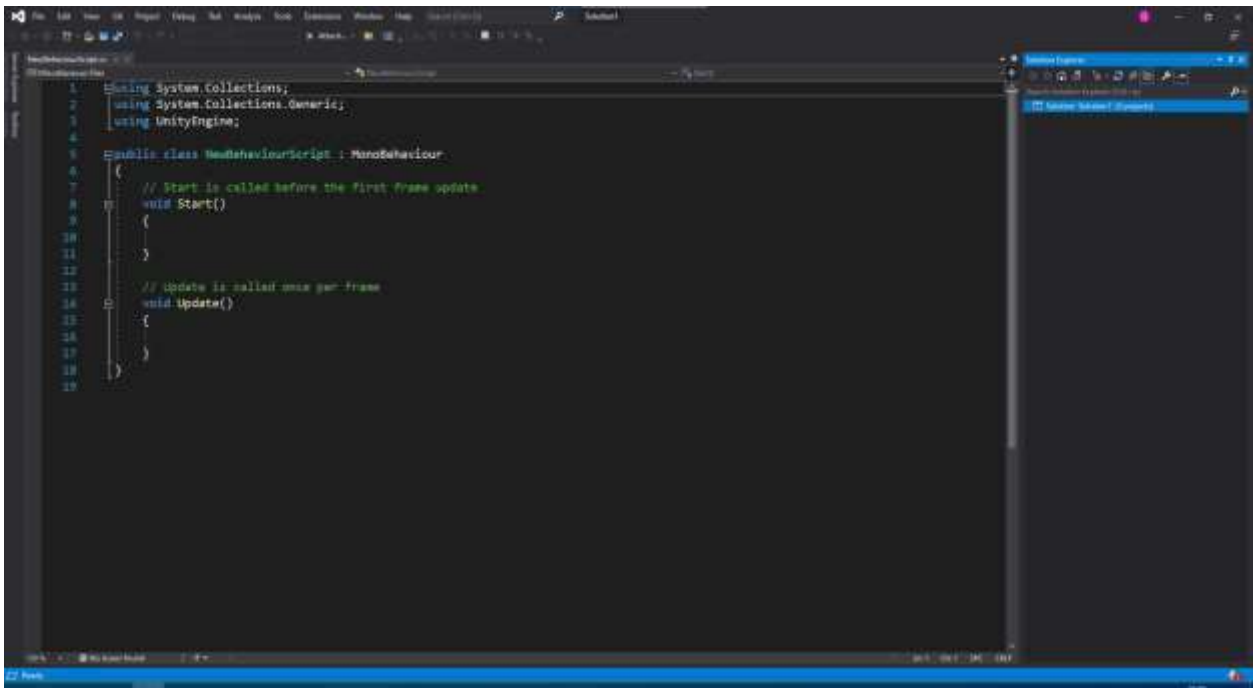
Рисунок 2.10 – Spyder

2.6 Visual Studio

Обраний для даної роботи ігровий рушій Unity, використовує скрипти на мові C# для реалізації функціоналу програми. В якості редактора коду на мові C# обрано Visual Studio. Ця середа розробки інтегрується з Unity,

автоматично підсвічуючи синтаксис і надаючи корисні підказки. На рисунку 2.11 зображено стандартно згенерований код в наведеній середі розробки.

Мова програмування C# є об'єктно-орієнтованою та типобезпечною. Вона входить до сімейства мов C і має вбудовані конструкції, спрямовані на підтримку принципів об'єктно-орієнтованого програмування. Завдяки цьому, C# є відмінним вибором для розробки та впровадження програмних компонентів.



2.11 – Стандартно-згенерований код

2.7 Висновки за розділом

У результаті було проаналізовано підклас нейронних мереж – згорткові нейронні мережі. В даній роботі обробка зображення є значною частиною завдання, тому для його вирішення було обрано згорткову нейронну мережу. Для її розробки оптимальним API є Keras. Описано засоби розробки які будуть використані в даній роботі.

РОЗДІЛ 3
СИМУЛЯТОР СИСТЕМИ АВТОНОМНОГО КЕРУВАННЯ

3.1 Умови симуляції

Метою даної роботи є розробка такої нейронної мережі яка після тренування зможе курувати автомобілем на невідомих їй раніше дорогах в створеному симуляторі. Тобто, автомобіль повинен проїхати на невідомому автодромі, на якому не проводилися тренування, повне коло не сходячи з проїжджої частини.

Конкретна реалізація полягає в створенні пустого автодрому на якому будуть проводитися тренування системи автоматичного керування та іншого автодрому для перевірки працездатності натренованої системи. Візуально, дорога та її оточення повинні бути приближені до реальних. Фізична модель автомобіля також повинна ґрунтуватися та законах реального світу.

Умови симулятора, які є важливими:

– візуальний контраст між проїжджою частиною та узбіччям дороги;

– наявність різносторонніх поворотів;

– повна видимість дороги;

– наявність сили тертя автомобіля та дороги;

– наявність інерції автомобіля;

– данні для системи автоматичного керування збираються з камер, розташованих спереду автомобіля на капоті;

– введення даних про кут повороту та силу прискорення здійснюється за допомогою клавіатури.

В симуляторі не передбачено цикли зміни дня та ночі, динамічні погодні умови, будь-які динамічні об’єкти на сцені, окрім самого авто. Реалізовано лише один тип дорожнього покриття – сухий асфальт.

3.2 Проєктування людино-машинний інтерфейс програми симулятора

Програма симуляції поведінки автомобіля повинна містити функції вибору різних дорожні карт для проведення тренування та тестування навченої нейронної мережі. Через це, в додатку необхідно передбачити можливість вибору з як найменш двох дорожніх карт. Також симулятор повинен надавати функції генерації вхідних даних для тренування нейронної мережі та функції для її тестування. Зважаючи на це доцільним буде схематичний начерк вікна головного меню, зображений на рисунку 3.1.

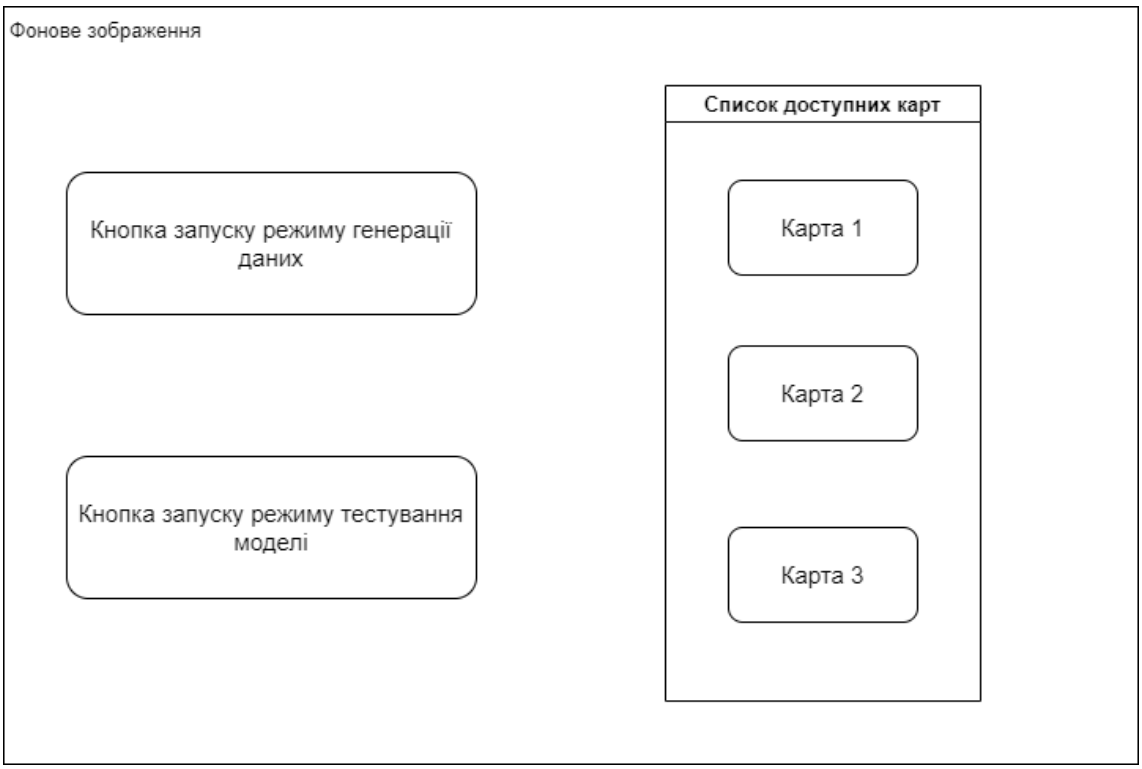


Рисунок 3.1 – Схема головного меню

На самій дорозі, при керуванні транспортним засобом, корисними параметрами для відстежування є швидкість та кут повороту керма авто. У режимі генерації даних необхідними елементами користувацького інтерфейсу є кнопка початку запису тестових даних та шкала виконання даного процесу. Згідно цього доцільним є схематичний начерк користувацького інтерфейсу під час генерації даних, зображений на рисунку 3.2.

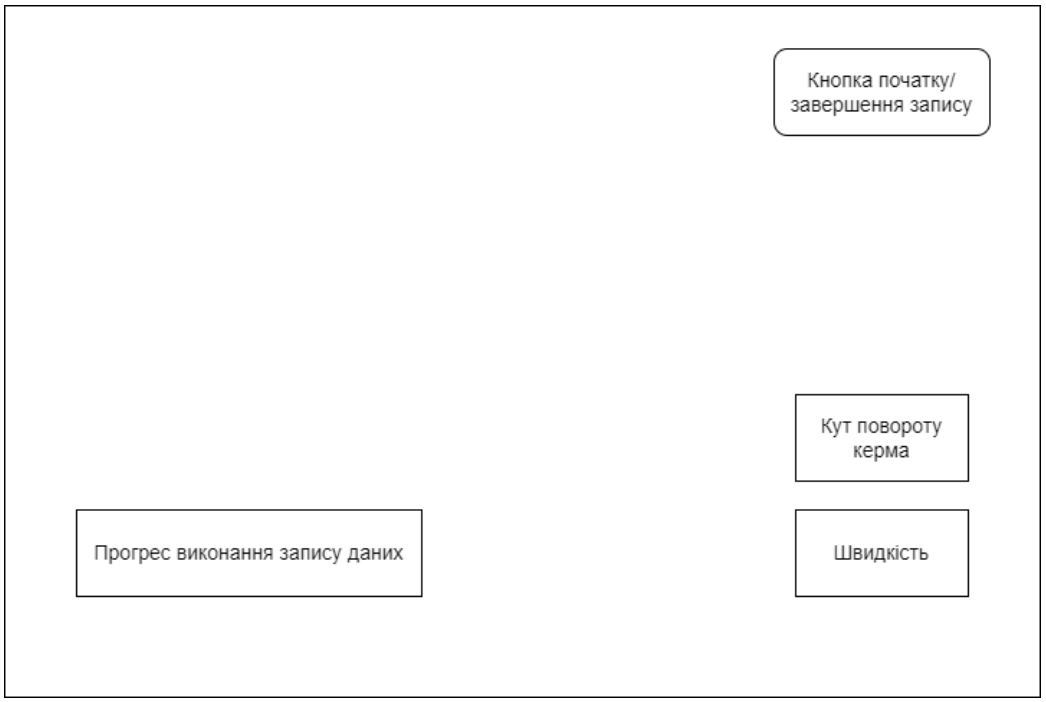


Рисунок 3.2 – Користувацький інтерфейс в режимі генерації даних

У режимі автономного керування, автомобілем все ще можна керувати вручну, тому доцільно буде додати елемент який показує поточний тип керування. Згідно цього доцільним є схематичний начерк користувацького інтерфейсу під час генерації даних, зображений на рисунку 3.3.

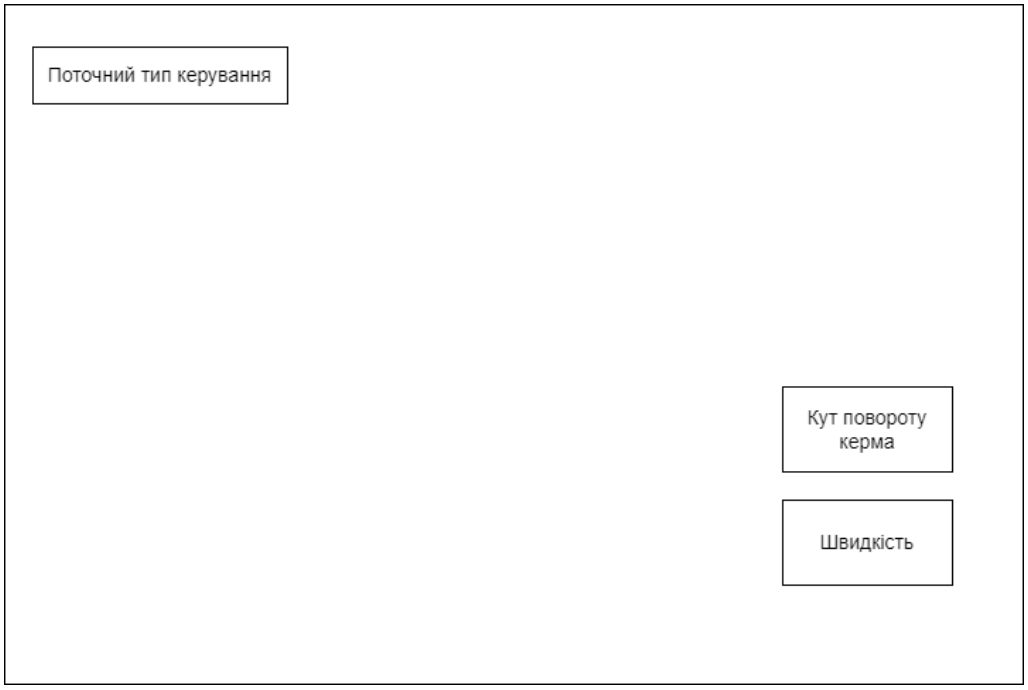


Рисунок 3.3 – Користувацький інтерфейс в режимі автономного керування

Програмний додаток має надавати можливість генерувати дані для тренування нейронної мережі. Алгоритм працює в два етапи: створення семплів, збереження на диск. На першому етапі, при натисканні кнопки початку запису генерується масив даних. Кожен елемент, або семпл, цього масиву містить наступні елементи:

– координати авто в просторі;
– кут повороту керма;
– прискорення;
– сила гальмування;
– швидкість;
– час створення семплу.

При повторному натисканні на кнопку запису, створення семплів зупиняється. На основі кожного семпла з цього масиву формується дані, які записується до файлу. Порядок роботи цього алгоритму відображено за допомогою діаграми станів яку зображено на рисунку 3.4.

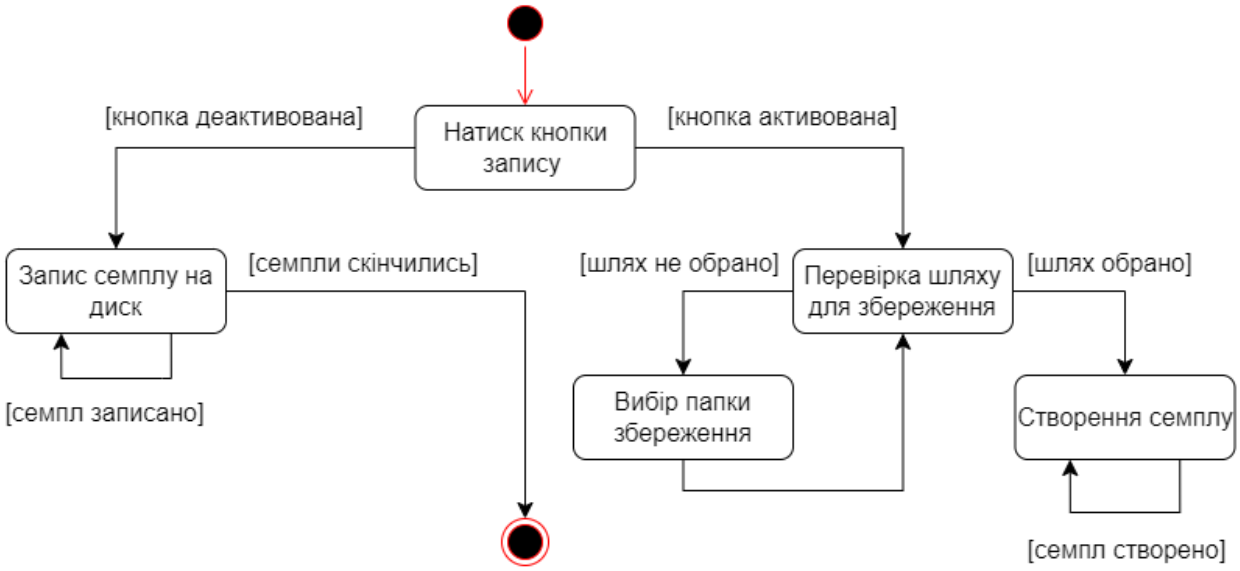


Рисунок 3.4 – Діаграма станів алгоритму генерації даних

3.3 Побудова програмного застосунку

Щоб створити новий проєкт, треба запустити Unity Hub, вибрати "New" та у відкритому вікні обрати тип проєкту, вказати його назву та визначити розташування, що зображено на рисунку 3.5.

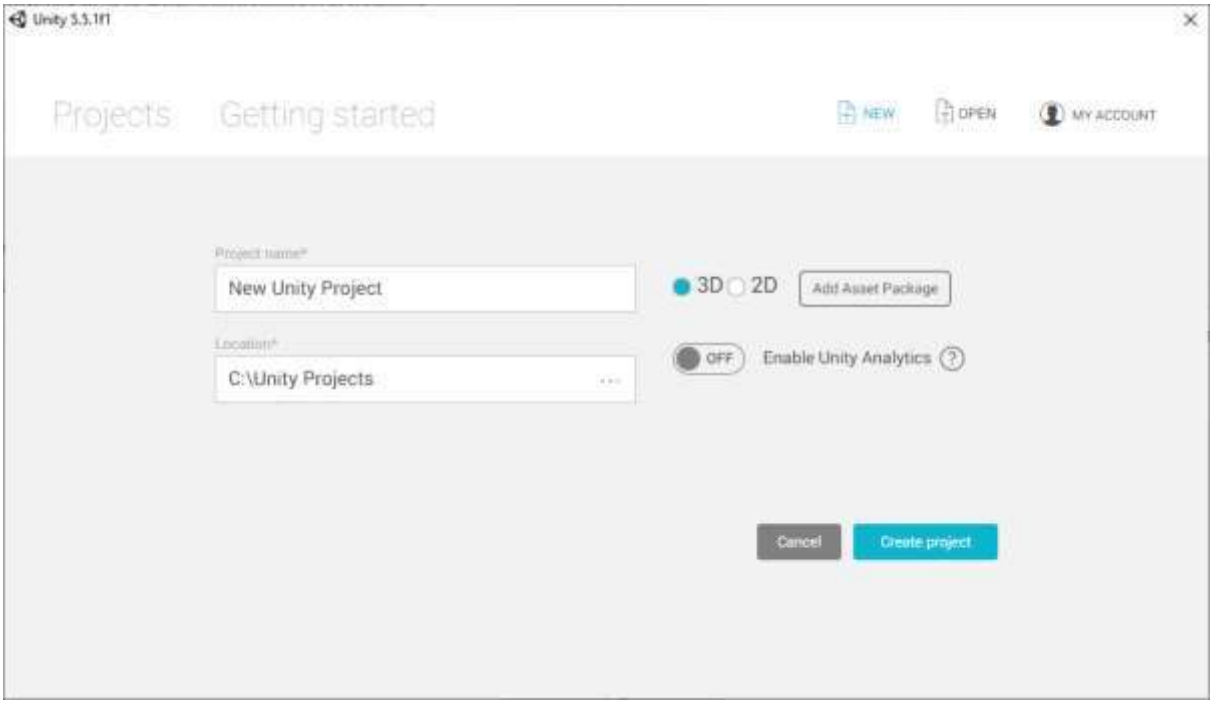


Рисунок 3.5 – Створення нового проєкту Unity

3.3.1 Використані сторонні ресурси

У проекті також використані сторонні ассети. В Unity термін «Asset» використовується для позначення різних ресурсів, таких як моделі, текстури, звуки, скрипти, анімації тощо, які використовуються в проєкті. Assets утримують дані та ресурси, необхідні для побудови ігрових сцен та функцій.

Для побудови дороги та візуального рельєфу використано ресурс «MountainTrack2». Він містить три різні за маршрутом типи доріг. Також представлені збережені сцени які відрізняються типом дороги, часом доби, часом пори року. Кожні сцена містить наступні об’єкти:

– Reflection Probe – містить компонент Reflection Probe, який використовується для симуляції відбиття світла, що дозволяє отримувати

реалістичні відбиття світла від об'єктів, навколишнього середовища та джерел світла;

– Directional light – джерело світла на сцені;

– маршрут – сукупність компонентів Mesh Collider (для реєстрації колізії) та Mesh Renderer (для візуалізації 3D-моделі або сітки) які відповідають за дорожнє полотно;

– оточення – сукупність компонентів Mesh Collider (для реєстрації колізії) та Mesh Renderer (для візуалізації 3D-моделі або сітки) які відповідають за оточуючі декоративні об’єкти.

Обрані сцени чудово підходять для генерації даних та перевірки моделей нейронних мереж, оскільки мають всі типи поворотів та виконані в реалістичному графічному дизайні. На зображеннях 3.6 та 3.7 зображено 3D моделі, які відображають замкнутий кільцевий маршрут, карт «HillTrack» та «SnakeTrack» відповідно.

Рисунок 3.6 – 3D модель кари «HillTrack»

Рисунок 3.7 – 3D модель кари «SnakeTrack»

Модель авто та реалізація фізики його поведінки взято з ресурсу «Standard Assets». Він містить 3D модель автомобіля (рисунок 3.8), скрипти для вводу команд керування та фізики поведінки, які будуть модифіковані.

Рисунок 3.8 - 3D модель автомобіля

Реалізація обміну даними між симулятором та нейронною мережею виконана за допомогою ресурсу SocketIO. Цей плагін дозволяє інтегрувати ваш Unity проєкт із серверною частиною Socket.IO. Під час підключення Socket.IO працює у власному потоці, щоб уникнути блокування основного потоку. Події ставляться в чергу та відправляються в наступному отриманому кадрі.

3.3.2 Головне меню симулятора

Головне меню симулятора складається з фонового зображення та чотирьох кнопок:

- Training mod;
- Auto mod;
- Hill track;
- Snake track.

Скрипт «MenuOptions.cs» реалізує взаємодію з перекисленими кнопками. При натисканні на кнопку групи вибору карти, обраний об’єкт виділиться та зміниться значення змінної «track» на відповідний порядковий номер.

При натисканні на кнопку Training mod завантажується сцена з відповідною картою, налаштована для генерації тренувального набору даних для нейронної мережі. Кнопка Auto mod виконує аналогічну функцію для заданого їй переліку сцен. На рисунку 3.9 зображено головне меню додатка.

Рисунок 3.9 – Головне меню додатка

3.3.3 Сцени для автономного керування

Сцени для автономного керування містять наступні ключові об’єкти:

– автомобіль з компонентами «CarController.cs» та «CarRemoteControl.cs», однією камерою для передачі зображень;

– об’єкт з компонентом «SocetIOComponent.cs» для встановлення зв’язку з програмою автономного керування;

– об’єкт з компонентом «CommandServer.cs» для передачі даних до програми автономного керування;

– об’єкт з компонентом «UISystem» для оновлення користувацького інтерфейсу.

При завантаженні сцени, компонент «CommandServer.cs» шукає інший компонент «SocetIOComponent.cs» в якому зазначені наступні параметри з’єднання:

– string url;

– bool autoConnect;

– int reconnectDelay;

– float ackExpirationTime;

– float pingInterval;

– float pingTimeout.

Налаштування цього компонента показані на рисунку 3.10.

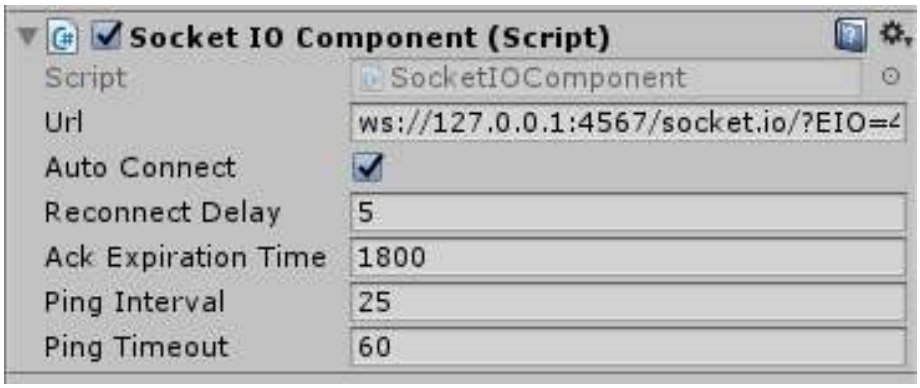


Рисунок 3.10 – Параметри з’єднання

Також створюються події типу SocketIOEvent: OnOpen, OnSteer, onManual. Події OnOpen та onManual лише відсилають дані до програми автоматичного керування використовуючи функцію EmitTelemetry. Ця функція передає інформацію про кут повороту керма, прискорення, швидкість, зображення отримане з камери автомобіля тільки в тому випадку, коли користувач не контролює автомобіль вручну.

Подія OnSteer отримує інформацію від модуля автономного керування у вигляді кута повороту керма та прискорення й записує їх до змінних float CarRemoteControl.SteeringAngle та float CarRemoteControl.Acceleration відповідно, посилаючись на його компонент «CarRemoteControl.cs». Потім відсилає дані, використовуючи EmitTelemetry.

У свою чергу компонент «CarRemoteControl.cs» має поля SteeringAngle та Acceleration, значення яких застосовується до автомобіля. При умові, що користувач не керує тим в ручному режимі, викликається функція Move, яка передає SteeringAngle та Acceleration до «CarController.cs».

Компонент «CarController.cs» є обов’язковим як для автономного автомобіля, так для керованого в ручному режимі. Тут описані методи для

реалізації фізики поведінки авто. Цей компонент оновлює фізичну тривимірну модель автомобіля, пересуваючи її у просторі та керує окремими елементами, такими як колеса. Налаштування даного компонента зображені на рисунку 3.11.

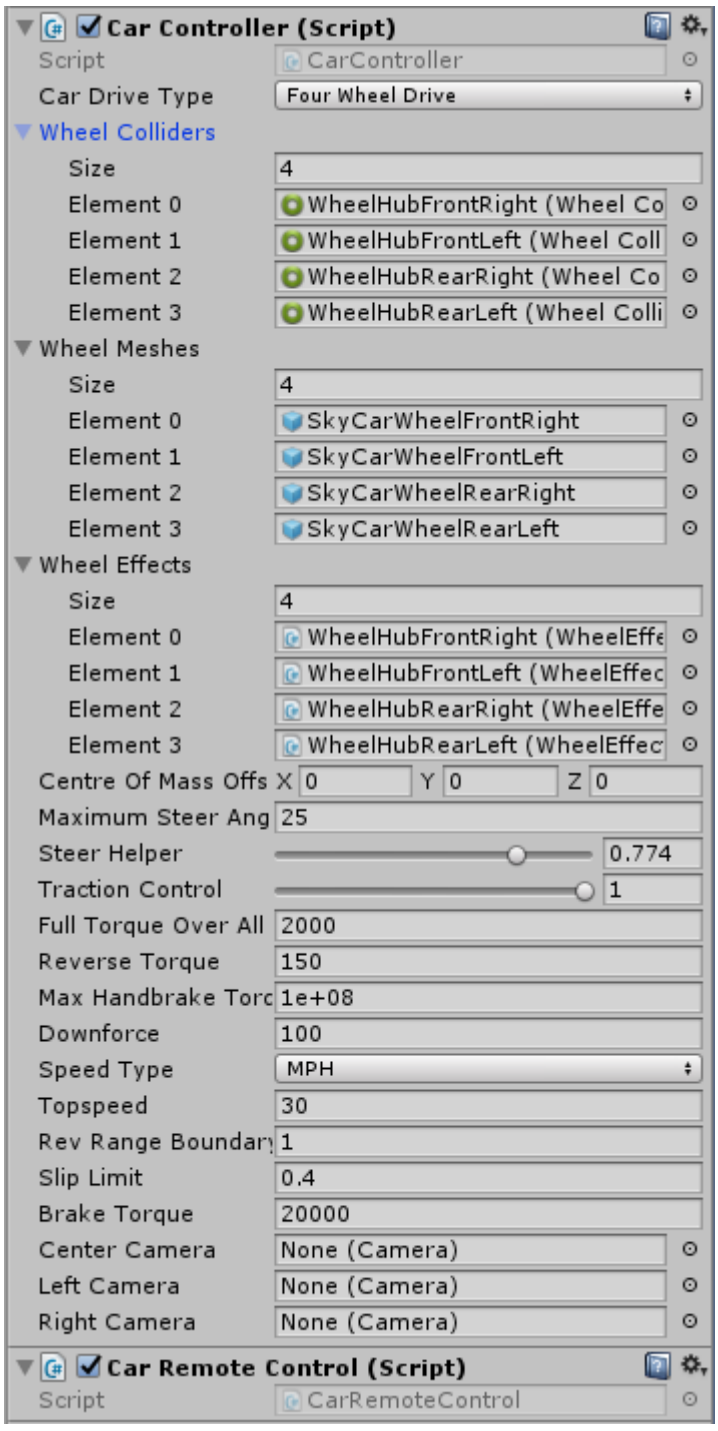


Рисунок 3.11 – Параметри компонента «CarController.cs»

Компонент «UISystem.cs» відповідає за оновлення показників зображених в користувацькому інтерфейсі. В ньому оновлюються значення швидкості та кута повороту, відображається тип керування автомобілем, оновлюються значення прогресу готовності записаних даних. До нього передаються візуальні елементи користувацького інтерфейсу типу Text. Передбачено також поле для посилання на компонент автомобіля, через який здійснена взаємодія двох компонентів. Цей компонент є обов'язковим як для автономного автомобіля, так для керованого в ручному режимі. Має булеву змінну isTraining, яка потрібно зазначати у відповідних сценах. Це змінить поведінку скрипта. При isTraining= false, скрипт працює для сцени з автономним керуванням та забороняє запис даних для генерації тестових наборів. При isTraining= true, скрипт працює для сцени генерації тренувальних даних та приховує надпис з поточним типом керування автомобілем. Налаштовані параметри компонента «UISystem.cs» зображено на рисунку 3.12. Користувацький інтерфейс наведено на рисунку 3.13.

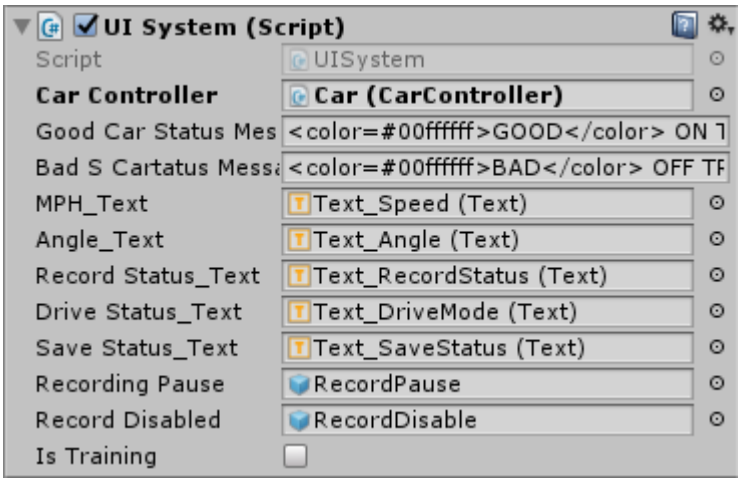


Рисунок 3.12 – Налаштовані параметри компонента «UISystem.cs»

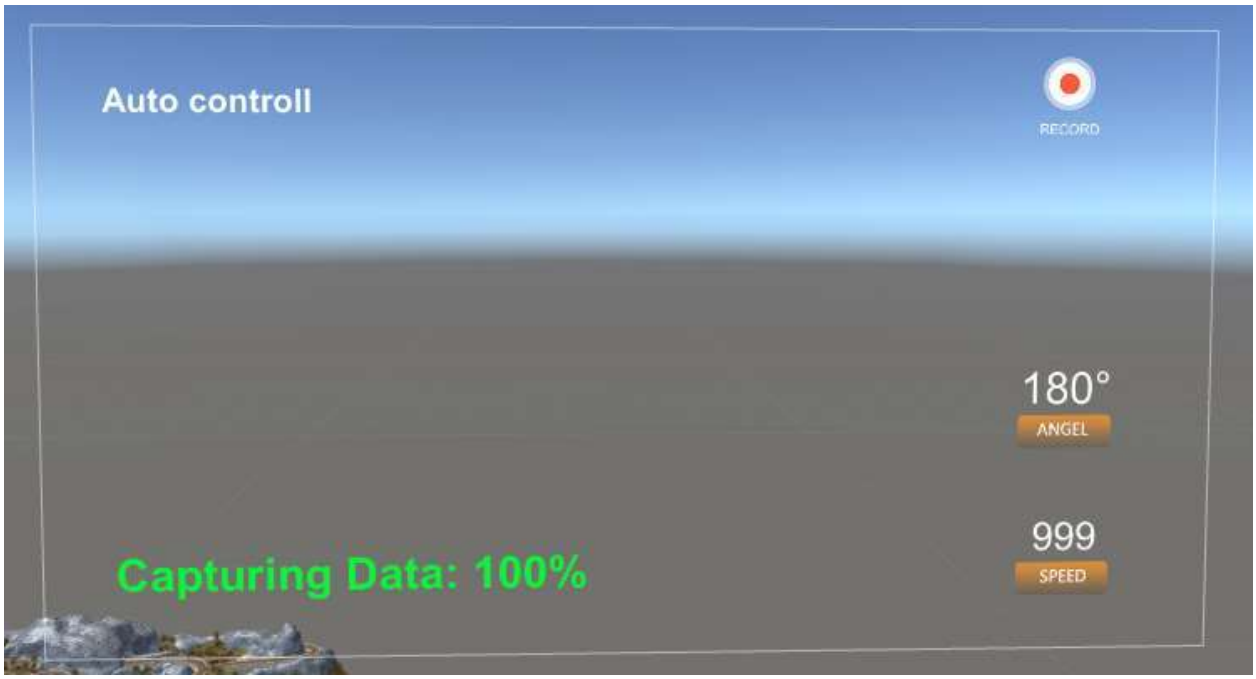


Рисунок 3.13 – Користувацький інтерфейс

На рисунку 3.14 зображено користувацький інтерфейс при вимкнутому записі даних.

Рисунок 3.14 – Користувацький інтерфейс при вимкнутому записі

На рисунку 3.15 зображено користувацький інтерфейс при ввімкнутому записі даних.

Рисунок 3.15 – Користувацький інтерфейс при ввімкнутому записі

На рисунку 3.16 зображено користувацький інтерфейс при автономному керуванні в автоматичному режимі керування.

Рисунок 3.16 – Користувацький інтерфейс при автономному керуванні

На рисунку 3.17 зображено користувацький інтерфейс при ручному керуванні в автоматичному режимі керування.

Рисунок 3.17 – Користувацький інтерфейс при ручному керуванні

3.3.4 Сцени для генерації тренувального набору

Сцени для автономного керування містять наступні ключові об’єкти:

– автомобіль з компонентами «CarController.cs» та «CarUserControl.cs», трьома камерами для передачі зображень;

– об’єкт з компонентом «UISystem» для оновлення користувацького інтерфейсу.

Компонент «CarUserControl.cs» приймає на вхід команди керування автомобілем від користувача та передає їх до компонента «CarController.cs». Ввід команд керування автомобілем реалізовано в окремому класі

«Steering.cs». В ньому описано публічні поля вводу напрямку по горизонтальній та вертикальній осях. Клавіші на клавіатурі «W», «S», «стрілка вгору», «стрілка вниз» відповідають за введення по вертикалі. Клавіші на клавіатурі «A», «D», «стрілка вліво», «стрілка вправо» відповідають за введення по горизонталі. Також реалізована можливість вводу даних по вертикалі за допомогою комп’ютерної мишки, при натиснутій лівій кнопці миші.

Компонент «UISystem» обробляє подію ToggleRecording, яка викликається при натисканні на кнопку користувацького інтерфейсу або кнопку «R» на клавіатурі. При цьому оновлюється відповідний елемент користувацького інтерфейсу та змінюється значення булевої змінної IsRecording в компоненті «CarController.cs», що означає початок запису даних для тренування нейронної мережі.

При першому записі за сесію, необхідно вказати шлях для збереження даних, що зображено на рисунку 3.18.

Рисунок 3.18 – Зазначення шляху збереження файлів

Генерація даних відбувається в два етапи: запис параметрів до оперативної пам’яті під час їзди, збереження даних та створення .csv файлу після завершення збору даних. Цей процес проводиться в два етапи через його ресурсну затратіть при одночасному зборі та зберіганні даних.

При зміні значення IsRecording=true в «CarController.cs», викликається функція Sample типу IEnumerator. В Unity IEnumerator застосовують головним чином для роботи з корутинами. Корутина – спосіб виконання асинхронного коду в Unity, що дозволяє здійснювати паузу та виконувати операції в течії часу без блокування основного потоку виконання. При виклику Sample, створюється новий екземпляр класу CarSample, до якого записується наступні параметри: rotation, position, steeringAngle, throttle, brake, speed, timeStamp. Екземпляри класу CarSample зберігаються до черги carSamples. Цей процес повторюється з періодичністю кожні 0,6 секунди до поки збирання даних не завершиться.

При зміні значення IsRecording=false, корутина Sample завершує своє виконання, зберігаються поточні значення позиції та кутів нахилу автомобіля у просторі для коректного позиціонування після запису даних викликається функція WriteSamplesToDisk типу IEnumerator й виконується поки в черзі carSamples кількість елементів більша нуля. За один цикл з черги вилучається елемент типу CarSample. Для автомобіля призначаються його параметри position та rotation для спостереження прогресу. Далі для кожної з трьох камер викликається функція WriteImage, який зберігає зображення та повертає шлях до нього. Далі формується рядок для запису до файлу .csv. Рядок має наступний формат: string.Format ("{0},{1},{2},{3},{4},{5},{6}\n", centerPath, leftPath, rightPath, sample.steeringAngle, sample.throttle, sample.brake, sample.speed). Після цього рядок записується до файлу та процес повторюється.

Функція WriteImage приймає параметр типу Camera, string з назвою камери, string з часовою міткою. Виконується функція «camera.Render()», щоб змусити подану камеру оновитися. Створюється зміна типу Texture2D з

параметрами камери, зображеними на рисунку 3.19. На її основі, функцією «texture2D.EncodeToJPG()» створюється зображення, а текстура знищується. Отримане зображення зберігається за вказаним раніше шляхом та має ім’я наступного типу: назвою камери + "_" + часова мітка + ".jpg".

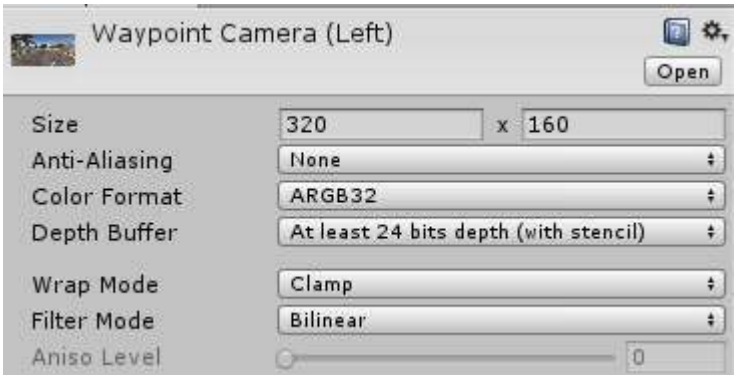


Рисунок 3.19 – Параметри лівої камери

Для тренування нейронної мережі необхідно записати тренувальний набір даних та валідаційний. Ці два набори записуються на одній дорозі та зберігаються в різні папки. В результаті запису даних генерується папка з зображеннями та файл «diving_log.csv», яку зображено на рисунку 3.20.

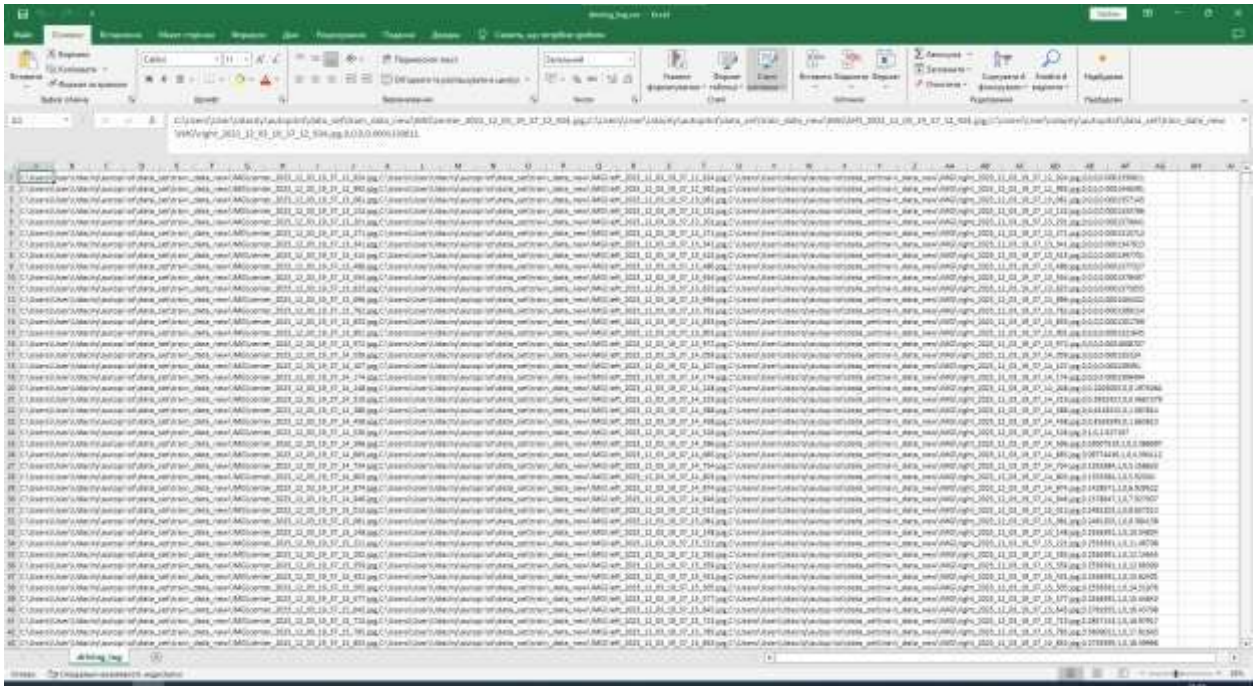


Рисунок 3.20 – Файл «diving_log.csv»

На рисунку 3.21 зображено папку із збереженими зображеннями.

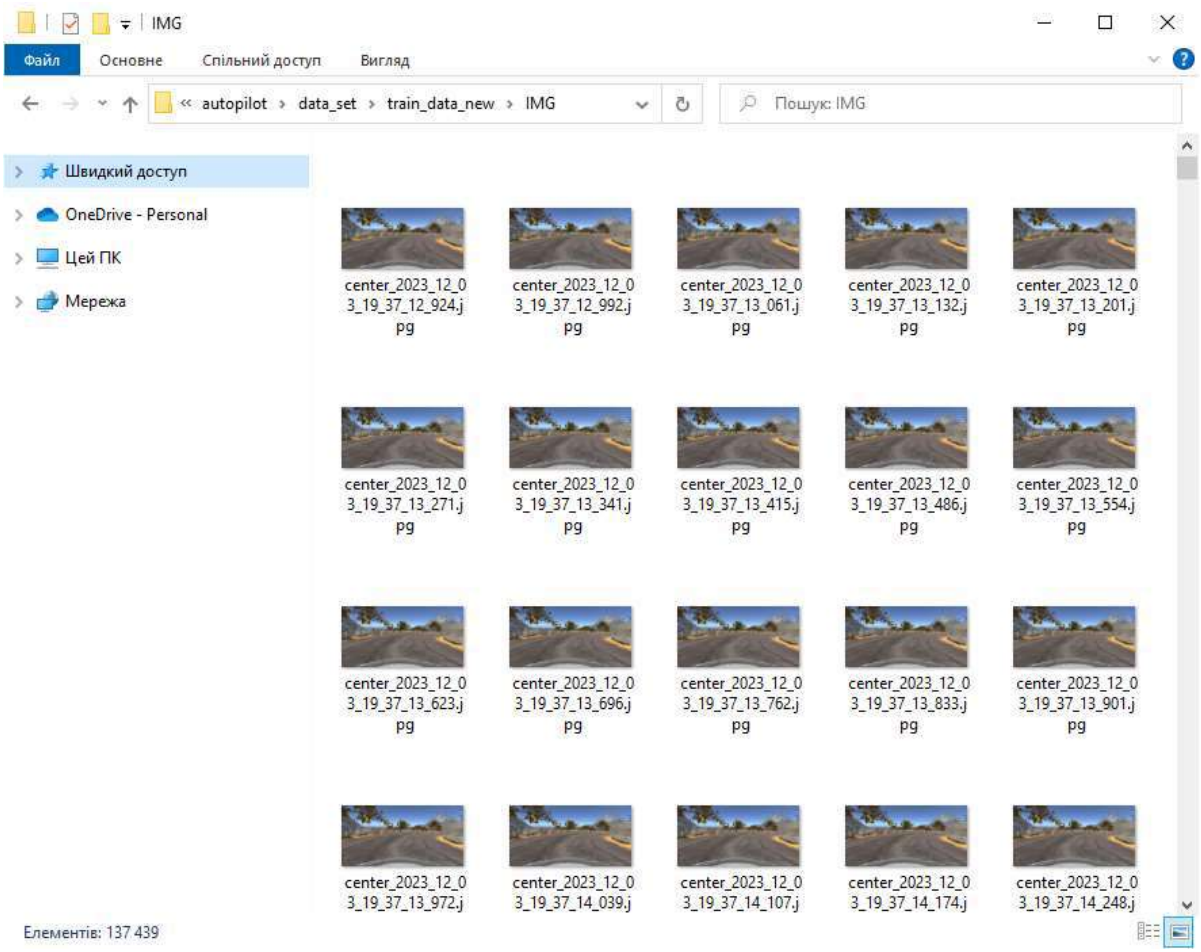


Рисунок 3.21 – Збережені зображення

На рисунку 3.22 зображено приклад збереженого зображення та його властивості на рисунку 3.23.

Рисунок 3.22 – Приклад зображення

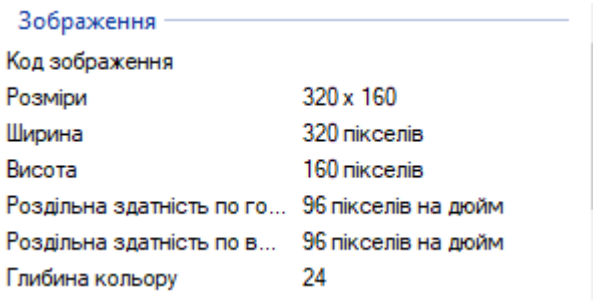


Рисунок 3.23 – Параметри зображення

Описані компоненти збережені як префаб – попередньо створений та налаштований об'єкт або група об'єктів, яку можна використовувати в інших сценах або додатках. Всі створені префаби при бажанні можна використовувати в інших додатках, які сумісні з версією Unity на якій вони були створені.

3.4 Висновки за розділом

Було спроєктовано та реалізовано симулятор для системи автономного керування. Описано обов’язкові пункти , які необхідно реалізувати в програмі. Приведено схеми головного меню та користувацького інтерфейсу на сцені. Описано загальну структуру сцен для автономного керування та сцен генерації навчальних даних. Описано взаємодію програмного застосунка на програми автономного керування. Описано принцип генерації та збереження тестових даних.

РОЗДІЛ 4
МОДУЛЬ АВТОНОМНОГО КЕРУВАННЯ

4.1 Система обміну даними

Спроєктована система для керування автомобілем складається з програмного додатку, розробленого за допомогою ігрового рушія Unity та python програми, яка відповідає за керування автомобілем шляхом генерації таких даних як прискорення та кут повороту керма. Обмін між цими модулями відбувається за рахунок архітектури клієнт-сервер. В цій архітектурі взаємодія між програмами поділяється на дві основні ролі: клієнт тобто python програма та сервер тобто Unity додаток. Клієнт — це програма або пристрій, який надсилає запити на сервер, тоді як сервер — це програма або пристрій, який надає відповіді на ці запити. В даному випадку, клієнт передає прискорення та кути повороту керма до сервера. Сервер в свою чергу, передає клієнту зображення та кут повороту керма. Таким чином, до поки запущено симулятор та python програму, вони безперервно обмінюються повідомленнями, що зображено на рисунку 4.1.

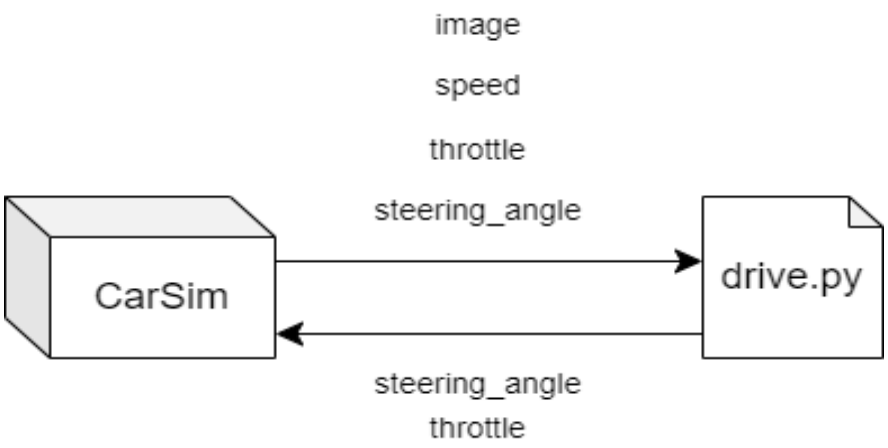


Рисунок 4.1 – Діаграма обміну даними

Обмін даними між модулями відбувається тільки коли користувач запустив рівень з дорогою. Обмін даними припиняється, коли користувач знаходиться в головному меню, але канал обміну не закривається. Таким чином можна змінювати рівні з дорогами без переривання роботи модуля автономного керування.

4.2 Структура нейронної мережі системи автономного керування

Найкращим вибором для даної роботи є архітектура на основі MiD-e2e, яка буде містити модуль з мережею типу SeD.

Загальна нейронна мережа, яка містить модулі: згорткової мережі для обробки зображень, рекурентної мережі для обробки куту повороту і глибинної багатошарової мережі перцептронів для об’єднання зображення та керування. Ця модульна архітектура надає можливість адаптувати систему для різних вхідних даних та вимог. Крім того, ця архітектура легко піддається розширенню для вирішення різних завдань, таких як підтримка світлофорів та інтеграція додаткових датчиків.

Система приймає зображення з камери та поточний кут повороту керма як вхідні дані та видає кут повороту керма. Ця структура мінімально оброблених вхідних даних і вихідних сигналів керування характерна для end-to-end архітектури мережі. Така нейронна мережа – це чорний ящик, який дає обмежене уявлення про те, як вона працюватиме. Структура описаної системи зображена на рисунку 4.2.

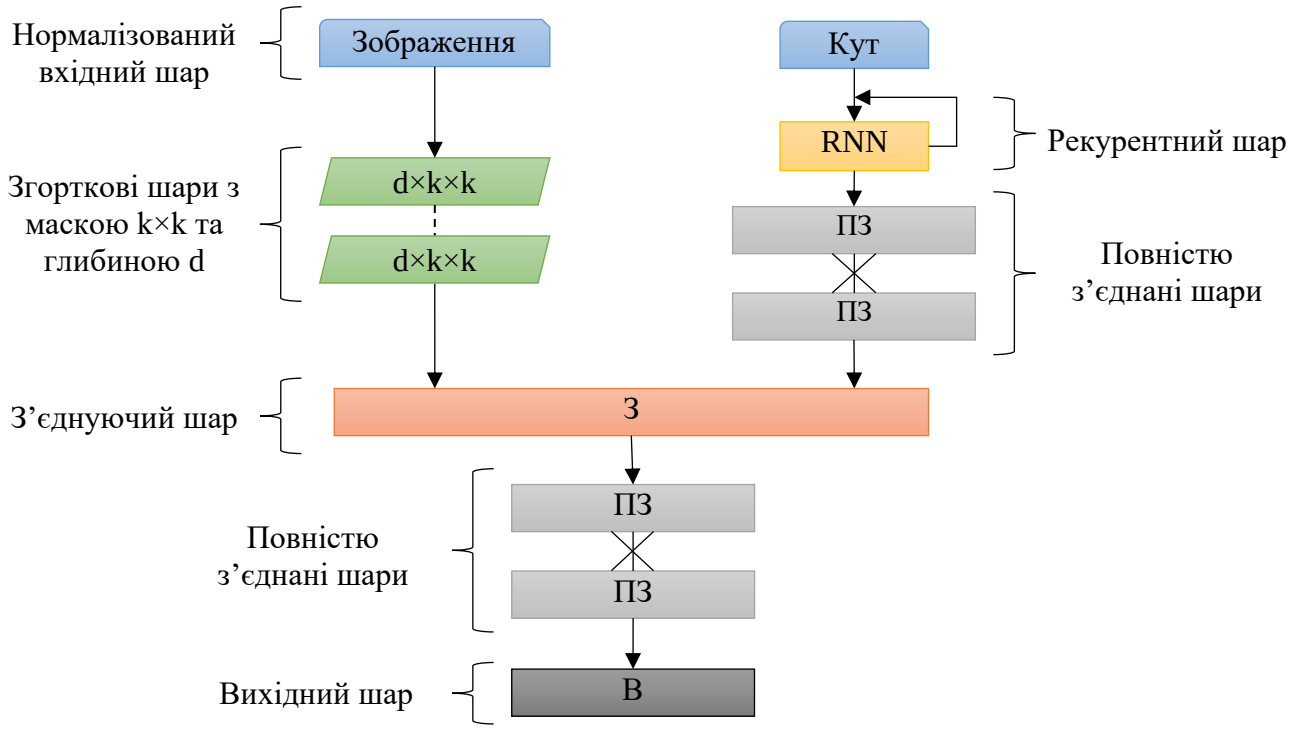


Рисунок 4.2 – Загальна нейронна мережа

Хоча наведена структура є бажаною для вирішення поставленої задачі, вона є складною в реалізації. Простіша нейронна мережа надає перевагу в часі для розробки, її налаштуванні та тестуванні. Зважаючи на це, було спроєктовано єдину нейронну мережу типу SiD, яка приймає на вхід зображення, прискорення та кут повороту керма. В даному випадку мережа приймає всі необхідні дані але втрачає гнучкість налаштування впливу різних груп вхідних параметрів. Структура описаної системи зображена на рисунку 4.3.

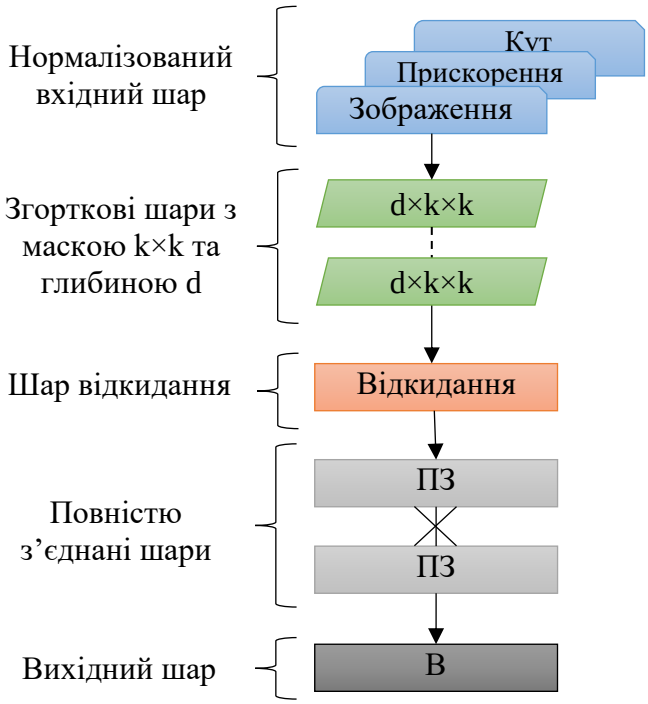


Рисунок 4.3 – Розроблена нейронна мережа

Наведена нейронна мережа складається з одинадцяти шарів. Перший рівень мережі виконує нормалізацію зображення, причому цей рівень жорстко закодований і не піддається налаштуванню під час навчання. Згорткові шари призначені для вилучення ознак і вибираються емпірично через серію експериментів, які змінюють ваги шарів. Використовуються чотири згорткові шари з маскою 5x5 і параметром «strided=2» та ще три з маскою 3x3 і параметром «strided=1». Останні чотири повністю з'єднані шари призначені для остаточного формування вихідного значення кута повороту керма, але точно сказати яку саме роль вони відіграють неможливо через end-to-end систему нейронної мережі. Шари розробленої нейронної мережі зображено на рисунку 4.4.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 160, 320, 3)]	0
rescaling (Rescaling)	(None, 160, 320, 3)	0
conv2d (Conv2D)	(None, 80, 160, 32)	2432
activation (Activation)	(None, 80, 160, 32)	0
conv2d_1 (Conv2D)	(None, 40, 80, 48)	38448
activation_1 (Activation)	(None, 40, 80, 48)	0
conv2d_2 (Conv2D)	(None, 20, 40, 64)	76864
activation_2 (Activation)	(None, 20, 40, 64)	0
conv2d_3 (Conv2D)	(None, 10, 20, 64)	102464
activation_3 (Activation)	(None, 10, 20, 64)	0
conv2d_4 (Conv2D)	(None, 10, 20, 76)	43852
activation_4 (Activation)	(None, 10, 20, 76)	0
conv2d_5 (Conv2D)	(None, 10, 20, 76)	52060
activation_5 (Activation)	(None, 10, 20, 76)	0
conv2d_6 (Conv2D)	(None, 10, 20, 76)	52060
activation_6 (Activation)	(None, 10, 20, 76)	0
dropout (Dropout)	(None, 10, 20, 76)	0
flatten (Flatten)	(None, 15200)	0
dense (Dense)	(None, 100)	1520100
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 20)	1020
dense_3 (Dense)	(None, 10)	210
dense_4 (Dense)	(None, 1)	11

Рисунок 4.4 – Розроблена нейронна мережа

В основі поданої нейронної мережі стоїть алгоритм клонування поведінки. Цей алгоритм є найпростішою формою імітаційного навчання, яке фокусується на вивченні політики експерта за допомогою навчання під наглядом. Як правило, імітаційне навчання корисне, коли експерту легше продемонструвати бажану поведінку, а не вказати функцію винагороди, яка б породила таку ж поведінку, або безпосередньо вивчити політику. Поведінкове

клонування безпосередньо вивчає політику за допомогою контрольованого навчання на парах «спостереження-дія» з експертних демонстрацій. Це простий підхід до вивчення політики, але політика часто погано узагальнює і погано відновлюється після помилок.

Перевагами даного алгоритму є:

– якщо на виконання завдання витрачається багато ресурсів, алгоритм імітаційного навчання може використовувати наявні дані;

– агент вчиться через наслідування, що є біль ефективним використанням інформації.

Недоліками даного алгоритму є:

– незважаючи на те, що агент отримує досвід від експерта, він все ще має шанс відхилитися від його поведінки. Невеликі помилки змусять агента більше відійти від експерта. Якщо агент опинився в ситуації, в якій експерт ніколи не був, агент може зробити фатальну помилку;

– агент лише вчиться відповідати на стан дією, без довгострокового планування.

Дані для навчання нейронної мережі записуються людиною. Під час навчання, мережа порівнює дані отримані від людини та згенеровані програмою на основі тренувальних даних, знаходячи різницю їх значень. Далі використовується метод зворотного поширення помилки оновлюючи ваги моделі на основі значень помилки. Як алгоритм машинного навчання, зворотне поширення є вирішальним кроком у загальному методі, який використовується для ітеративного навчання моделі нейронної мережі. Він використовується для розрахунку необхідних коригувань параметрів, щоб поступово мінімізувати помилку. Зворотне розповсюдження обчислює градієнт функції втрат відносно ваг мережі для окремого прикладу вводу-виводу, і робить це ефективно, обчислюючи градієнт по одному шару за раз, ітеруючи назад від останнього шару, щоб уникнути зайвих обчислень. В якості функції оцінки використано метод найменших квадратів.

4.3 Програмна реалізація модулів автономного керування

Хоч TensorFlow є основною бібліотекою для розробки нейронних мереж, необхідними є ще безліч інших бібліотек. Для комфортного завантаження пакетів, контрою версій та сумісності окремих модулів, встановлення залежностей, було створено віртуальне оточення за допомогою Miniconda. Miniconda — безкоштовний мінімальний інсталятор для conda. Це невелика початкова версія Anaconda, яка включає лише conda, Python, пакети, від яких вони обидва залежать, і невелику кількість інших корисних пакетів. Також за допомогою цього інсталятора можна створити віртуальне оточення, яке містить всі необхідні бібліотеки використовуючи конфігураційний файл .yml або .yaml.

Щоб створити віртуальне оточення за допомогою конфігураційного файлу, було виконано команду «conda env create -f C:\Users\User\source\repos\self_mod_conda_env.yaml» у вікні «Anaconda Prompt», де й необхідно в подальшому запускати виконання python скриптів. Після активації оточення командою «conda activate self_mod», можна переконатися в наявності всіх необхідних пакетів виконавши команду «conda list» (рисунку 4.5).

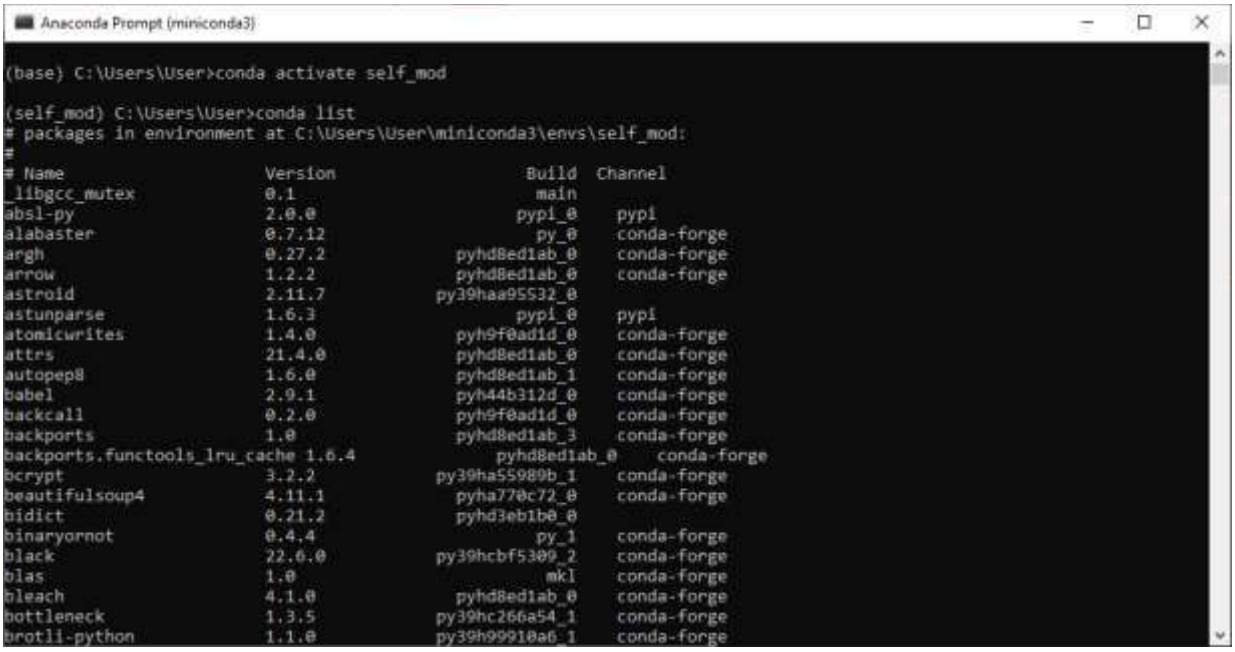


Рисунок 4.5 – Перелік встановлених пакетів

4.3.1 Програмний модуль для створення Keras моделі

Програмний модуль «model.py» містить метод для створення Keras моделі для автономного керування автомобілем. Метод приймає один аргумент «inshape». Тут має бути зазначено висота та ширина вхідного зображення. Вихідні дані моделі будуть єдиним значенням з плаваючою комою у діапазоні від -1 до 1. Це буде кут повороту, який передано симулятору. Метод повертає модель Keras з єдиним виходом регресії. Модель містить:

– Чотири шари Conv2D з різними параметрами «filters» та спільними параметрами: «kernel_size=5», «strides=2», «padding="same"»;

– Три шари Conv2D з різними параметрами «filters» та спільними параметрами: «kernel_size=3», «strides=1», «padding="same"»;

– Шар Dropout(0.5);

– Шар Flatten();

– Чотири шари Dense() з різними параметрами «units».

Conv2D є згортковим шаром введення в бібліотеці Keras, який використовується для обробки зображень. Цей шар використовує згорткові фільтри для вилучення важливих функцій з введеного зображення. У багатьох випадках використовується для виявлення різних візуальних особливостей, таких як границі, текстури та об'єкти. При виклику вказані наступні параметри:

1) filters – кількість фільтрів, що застосовуються до вхідного зображення;

2) kernel_size – розмір згорткового ядра (фільтра);

3) strides – значення, що визначає крок згортки по ширині і висоті;

4) padding: Визначає режим доповнення: 'valid' (без доповнення) або 'same' (доповнення, щоб вивід був того ж розміру, що і вхід);

5) activation – функція активації, яка застосовується після згортки.

Dropout є шаром регулювання в нейронних мережах і використовується для випадкового вимикання деяких виходів під час тренування. Це допомагає уникнути перенавчання, зменшити взаємозалежність між нейронами та поліпшити здатність узальнювати вхідні дані моделі. Параметр «rate» визначає частку виходів, які будуть вимкнені. При «rate=0.5» половина виходів буде вимкнено під час кожного оновлення.

Flatten є шаром в бібліотеці Keras, який використовується для перетворення вхідних даних в одновимірний вектор. Використовується при переході від шарів згортки до повністю з'єднаних шарів. Основна задача цього шару – згладжувати розмірність даних, що надходять з попереднього шару, так щоб їх можна було використовувати в повністю з'єднаних шарах.

Dense є базовим шаром у бібліотеці Keras, який представляє собою повністю з'єднаний шар нейронів. У шарі Dense кожен нейрон пов'язаний з кожним нейроном на попередньому шарі. Це означає, що кожен вхідний сигнал проходить через вагу, яка налаштовується під час тренування моделі. Параметр «units» визначає кількість нейронів у шарі.

Для всіх шарів використано функцію активації ReLU. Структура отриманої моделі зображена на рисунку 4.6.

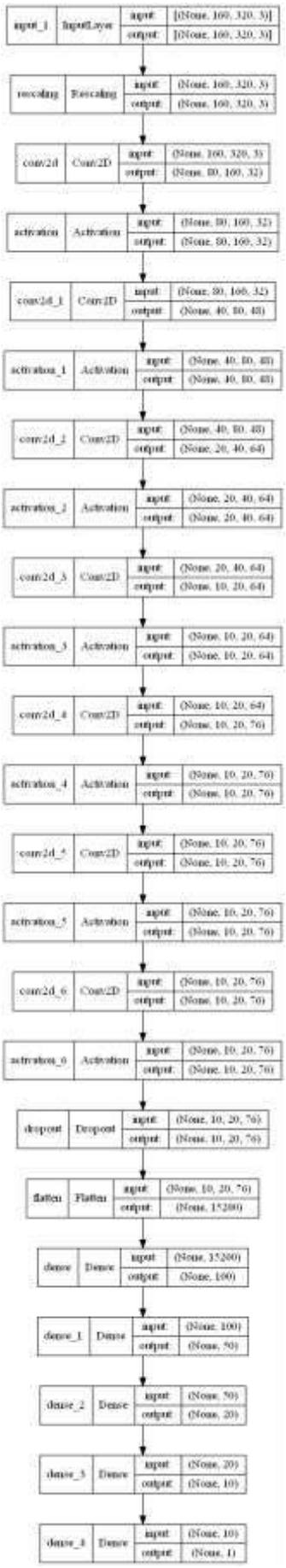


Рисунок 4.6 – Структура отриманої моделі

Програмний модуль «model.py» є допоміжним для «model_train.py», в якому використовуються його методи для побудови та тренування моделі.

4.3.2 Програмний модуль опису структури нейронної мережі

Програмний модуль містить «utils.py» з модифікованим генератором послідовності, або генератором набору даних, створеному на базі стандартного класу у бібліотеці Keras під назвою Sequence. Конструктор цього модифікованого класу приймає наступні параметри:

1) шлях до .csv файлу, який містить згенеровані дані для навчання або тестування;
2) batch size – кількість зразків;
3) count – для вибору діапазону елементів з файлу .csv;
4) imgsize – розмір в якому буде завантажено зображення;

Конструктор модифікованого класу ініціалізує зміні класу вхідними параметрами.

Також модифіковано метод доступу до елемента за індексом «__getitem__». Новий метод знаходить даних в переданому .csv файлі використовуючи передані параметри batch size та індекс. Завантажує дані та форматує, складаючи зображення з трьох камер горизонтально та повертає їх як масив даних «x», знаходить відповідне значення кута повороту керма та повертає його як значення «y».

4.3.3 Програмний модуль навчання нейронної мережі

Даний модуль спроєктованої системи містить код який реалізує навчання створеної нейронної мережі. Приймаючи параметри, вказані при запуску скрипту, виконується формування змінних з даними для тренування та тестування нейронної мережі.

Модуль тренування нейронної мережі «model_train.py» необхідно виконувати використовуючи Anaconda prompt з наступними параметрами:

– train_csv_file – шлях до файлу в якому знаходяться посилання на файл з даними для тестування нейронної мережі;

– test_csv_file – шлях до файлу в якому знаходяться посилання на файл з даними для тренування нейронної мережі;

– batch_size – гіперпараметр, який визначає кількість зразків, які потрібно опрацювати перед оновленням параметрів внутрішньої моделі;

– epochs – гіперпараметр, який вказує на кількість повних проходів алгоритму навчання через весь набір даних.

Параметр " batch_size " процесі навчання подібний до циклу for, який ітераційно обробляє один або кілька зразків та робить прогнози. Після кожного циклу прогнози порівнюються з очікуваними значеннями вихідних змінних, і помилка обчислюється. З цієї помилки алгоритм оновлення використовується для вдосконалення моделі, наприклад, переміщаючись вниз уздовж градієнта помилок. Навчальний набір даних представляє собою розділення на одиничні або кілька "партій". В даному випадку розмір партії перевищує один зразок і менше, ніж розмір навчального набору даних, тому використаний алгоритм можна назвати міні-пакетним градієнтним спуском.

Параметр "epochs" визначає, що кожен елемент у навчальному наборі даних має можливість впливати на оновлення параметрів внутрішньої моделі. Епоха представляє собою період, під час якого відбувається обробка однієї або кількох партій. Можна розглядати цю динаміку, як цикл for, де кількість ітерацій визначається параметром "epochs", і кожна ітерація охоплює обробку навчального набору даних. У межах цього циклу for виявляється ще один вкладений цикл for, який повторює кожну партію зразків.

Всі вхідні параметри записуються до змінної args та присвоюються іншим відповідним змінним. На основі файлів train_csv_file та test_csv_file створено відповідні набори даних за допомогою методу dataset(test_csv_file_path,batchsize) та dataset(train_csv_file_path,batchsize), яку описано в модулі utils.py.

Наступним кроком створено Keras модель з параметром розміру вхідних зображень. За допомогою вбудованого до Keras методу summary(), інформація про створену модель та хід її навчання зберігається до файлу train.log, приклад якого наведено на рисунок 4.7.

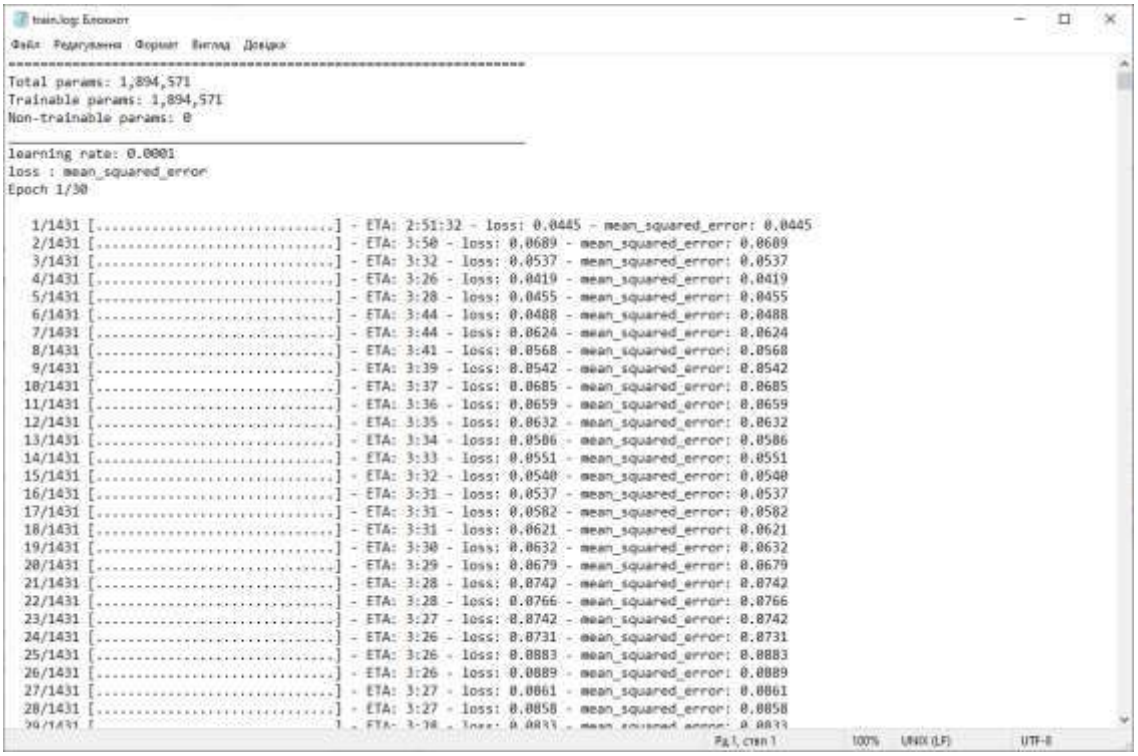


Рисунок 4.7 – Приклад файлу train.log

Метод compile() в бібліотеці Keras використовується для налаштування параметрів моделі перед початком процесу навчання. Цей метод визначає алгоритм оптимізації, функцію втрат та метрики для відстеження під час навчання моделі. Аргументами даного методу є:

– оптимізатор – визначає алгоритм оптимізації, який використовується для оновлення ваг моделі під час навчання;

– функція втрат – визначає, яким чином вимірюється відхилення прогнозів моделі від фактичних значень;

– метрика – вказує метрики, які використовуються для оцінки якості моделі під час навчання.

В якості оптимізатора обрано алгоритм Адама. Оптимізація Адама – метод стохастичного градієнтного спуску, який базується на адаптивній оцінці моментів першого та другого порядку. Цей метод обчислювально ефективний, має невеликі вимоги до пам'яті, інваріантний до діагонального масштабування градієнтів і добре підходить для проблем, які є великими з точки зору даних та параметрів.

Функцією втрат обрано середнє квадратичне відхилення, що вказано як змінна loss="mean_squared_error". Вона вимірює середньоквадратичну різницю між прогнозованими значеннями моделі та фактичними значеннями. Мінімізація цієї функції під час навчання означає, що модель стає кращою в передбаченні числових результатів.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2, \quad (4.1)$$

де, n – номер змінної,

Y_i – вектор спостережених значень,

$\widehat{Y}_i$ – вектор передбачених значень.

Наступним необхідним параметром для тренування нейронної мережі є learning rate, або швидкість навчання. Стохастичний градієнтний спад – алгоритм оптимізації, який оцінює градієнт помилок для поточного стану моделі, використовуючи приклади з навчального набору даних, а потім оновлює вагові коефіцієнти моделі за допомогою алгоритму зворотного поширення помилок, що називається просто зворотним поширенням. Величина, на яку ваги оновлюються під час навчання, називається розміром кроку або «швидкістю навчання». Зокрема, швидкість навчання – гіперпараметр, який використовується для навчання нейронних мереж і має невелике позитивне значення, часто в діапазоні від 0,0 до 1,0.

Швидкість навчання контролює, наскільки швидко модель адаптується до проблеми. Менші значення навчання вимагають більшої кількості епох

навчання, враховуючи менші зміни вагових коефіцієнтів кожного оновлення, тоді як більші значення навчання призводять до швидких змін і вимагають меншої кількості епох навчання. Надто велика швидкість навчання може призвести до того, що модель надто швидко наблизиться до неоптимального рішення, тоді як надто мала швидкість навчання може призвести до зупинки процесу. Тому для тренування моделі обрано наступні значення швидкості навчання lrs=[0.001, 0.0001, 0.00001, 0.01]. Для кожного значення швидкість навчання, програма викликає методи compile() та fit().

Для тренування моделі машинного навчання на навчальних даних в бібліотеці Keras використовується метод fit() . Під час виклику цього методу, модель адаптує свої ваги та параметри так, щоб мінімізувати обрану функцію втрати. При виклику даного методу вказано наступні параметри:

– навчальні дані – вказується вхідний набір даних, на якому модель буде тренуватися;

– валідаційниі дані – дані, на основі яких можна оцінити втрати, і будь-які показники моделі наприкінці кожної епохи;

– кількість епох – кількість разів повторення тренування на вхідних даних;

– виклики – дозволяє передавати функції, які викликаються під час різних етапів навчання, такі як зупинка раннього навчання, збереження моделі.

Для побудови даної моделі використано два виклики: ModelCheckpoint() та EarlyStopping().

Основна мета ModelCheckpoint() – забезпечити можливість зберігати прогрес моделі та викликати збереження ваг, якщо вона досягла певного покращення в якості під час навчання. Цей метод використано з параметром save_best_only= True, який вказує, чи слід зберігати лише найкращі моделі відносно моніторингової величини.

EarlyStopping() використовується для автоматичної зупинки тренування моделі, якщо певна метрика не покращується протягом заданої кількості епох.

Цей метод використано з параметрами: monitor='loss' (величина, за якою слід визначати, чи відбулося покращення), patience=5 (ціле значення, яке вказує, скільки епох може пройти без покращення, перед тим як тренування буде зупинено), mode='min' (режим, в якому слід визначати покращення).

На рисунку 4.8 зображено результат роботи скрипта «model_train.py».

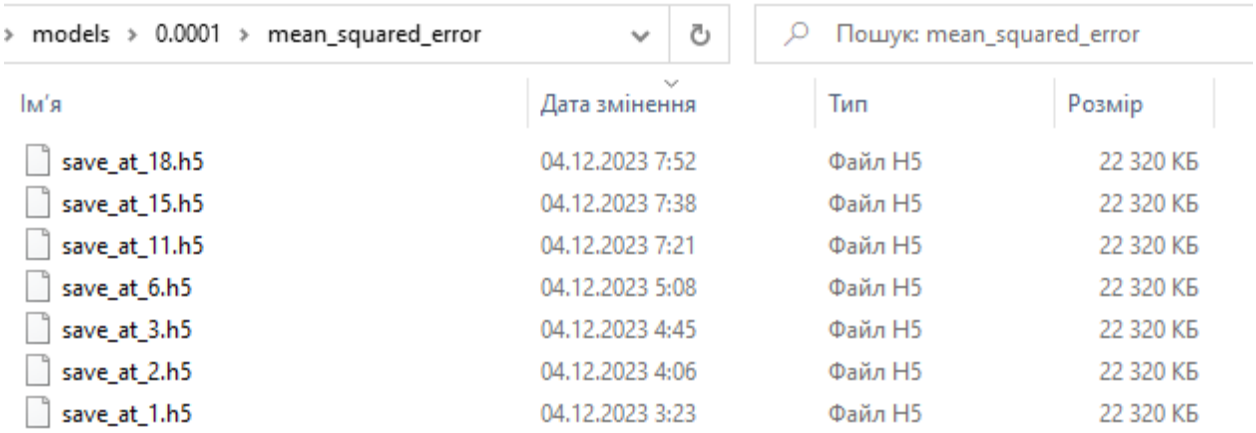


Рисунок 4.8 – Результат роботи «model_train.py»

Після завершення виконання даного скрипта, у вказаній при виконанні папці, з'являться файли типу .h5, які містять архітектуру та ваги моделі, конфігурацію навчання.

4.4 Модуль автономного керування автомобілем

Даний модуль автоматичного керування автомобілем «drive.py» реалізує з'єднання з симулятором та обмін даними. При виконанні цього скрипта необхідно вказати шлях до збереженої Keras моделі з форматом файлу . h5. Виконується ініціалізація полів та встановлення з'єднання з симулятором. Також в цьому модулі описано події «@sio.on('telemetry')» та «@sio.on('connect')».

В коді на мові програмування Python з використанням бібліотеки Socket.IO, фрагмент коду «@sio.on('connect')» вказує на те, що визначена функція або метод буде викликана при події з'єднання з клієнтом через сокет.

Коли клієнт успішно встановлює з'єднання із сервером Socket.IO, подія 'connect' спрацьовує, і викликає функцію «connect(sid, environ)», яка виконує тестову передачу даних до програми у вигляді значень прискорення та куту повороту керма рівними нулю за допомогою «send_control(0, 0)».

В коді на мові програмування Python з використанням бібліотеки Socket.IO, фрагмент коду @sio.on('telemetry') вказує на те, що визначена функція або метод буде обробляти подію з ім'ям 'telemetry', яка приходить через сокет. Коли клієнт відправляє подію 'telemetry', цей фрагмент коду визначає, що для обробки цієї події необхідно викликати функцію telemetry(sid, data). У випадку коли автомобіль керується вручну, обробка цієї події ігнорується бо не було надіслано даних. Інакше, отримані дані про кут повороту керма, прискорення, швидкості, зображення дороги завантажуються до моделі використовуючи вбудований метод Keras predict(). У бібліотеці Keras функція model.predict() використовується для отримання прогнозів моделі на нових даних. Ця функція приймає вхідні дані та повертає вихідні прогнози моделі відносно кута повороту автомобіля.

Для розрахунку прискорення використовується наступна формула з попередньо вказаними значеннями MAX_SPEED та MIN_SPEED, $speed_limit = \text{MAX_SPEED}$:

$$a = 1 - steering_angel^{\ 2} - (speed/speed_limit)^2, \qquad (4.2)$$

Де $steering_angel$ - кут повороту керма,

$speed$ – поточна швидкість автомобіля,

max_speed – максимальна допустима швидкість автомобіля.

Таким чином, авто буде рухатися із швидкістю вказаною як MAX_SPEED та зменшувати її на поворотах. При досягненні MAX_SPEED, $speed_limit = \text{MIN_SPEED}$, що змушує авто гальмувати.

Отримані значення прискорення та куту повороту керма відправляються симулятору, використовуючи функцію emit(). Це функція в бібліотеці Socket.IO для відправки повідомлень від сервера до клієнтів чи між серверами. Вона використовується для передачі даних або сповіщення клієнтів про події на сервері. Результат роботи модуля «drive.py» зображено на рисунку 4.9.

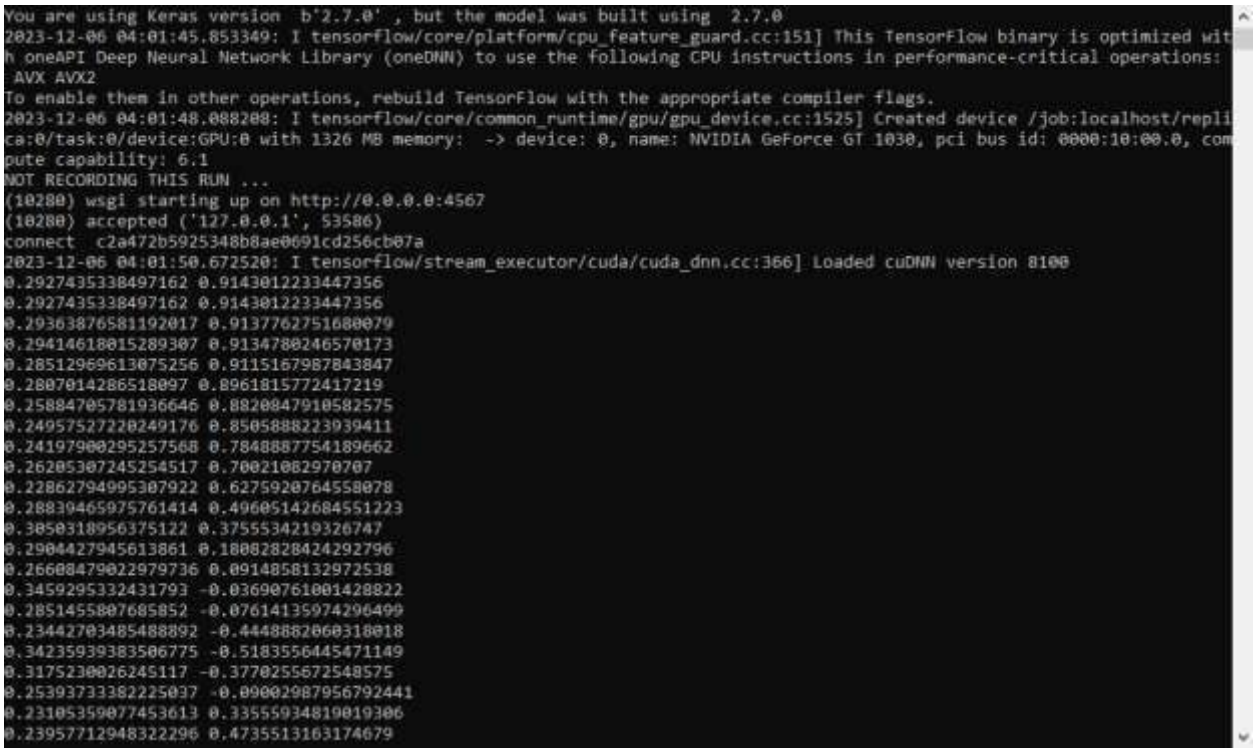


Рисунок 4.9 – Результат роботи модуля «drive.py»

Таким чином «drive.py» отримує дані про прогнозований кут повороту керма з навченої нейронної мережі, рахує прискорення авто та передає ці дані до симулятора.

4.5 Висновки за розділом

У даному розділі було розглянуто існуючі топології нейронних мереж для автономного керування, спроектовано та реалізовано власну структуру нейронної мережі. Наведено її реалізацію за допомогою TensorFlow та його надбудови Keras. Реалізовано модуль автономного керування автомобілем на основі збережених моделей глибокого навчання.

РОЗДІЛ 5
НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

5.1 Налаштування параметрів симулятора

Для проведення тестів та збору даних в симуляторі необхідно налаштувати швидкість пересування авто.

Періодичність генерації кожного тренувального елемента в симуляторі фіксована. Через це, надмірна швидкість авто призведе до генерації неякісного тренувального набору даних. Водій повинен буде вводити велику кількість даних за короткий проміжок часу, а симулятор встигне записати лише невелику частину його поведінки. В ключові моменти, такі як уникнення зіткнення чи крутий поворот, необхідно мати якнайбільше інформації про поведінку водія, яка втратиться через швидкість всіх подій.

Наприклад при швидкості 100км/год, або 28м/с та частотою запису даних 1 кадр кожні 0,7с, або 1,4кадрів/с. Дистанції 28м достатньо, щоб виконати швидкий маневр. Цей маневр буде зафіксована в 1 або 2 кадри, чого замало для розуміння логіки необхідної для його виконання.

З іншого боку, при занадто малій швидкості тренувальний набір даних також спотвориться однотипними даними. Також проблемою стане розмір цього набору та подальше тренування нейронної мережі на його основі.

Емпіричним методом, для даного типу дороги, було виявлено, що найкраща швидкість це приблизно 60км/год. При такій швидкості, для кожного повороту є достатньо часу для запису в набір даних.

5.2 Формування набору даних

Набір даних для навчання нейронної мережі є одним з найголовніших факторів успішності фінальної моделі, особливо при використанні алгоритму копіювання поведінки. Тому необхідно використовувати тільки ті дані, які відповідають бажаній поведінці.

В даному випадку було згенеровано два набори даних, кожен з яких включає в себе двадцять повних кіл на дорозі. Перший набір даних включає в себе одне коло на якому при кожному повороті було зіткнення та послідуючий виїзд на цент дороги. Другий – містив лише бажану поведінку та рідкі випадкові зіткнення. В результаті, моделі треновані на першому наборі, набагато частіше стикалися з оточенням. Ці моделі також не могли викарабкатися з тупикової ситуації, від’їхавши назад, і продовжували їхати в стіну.

5.3 Вибір гіперпараметрів нейронної мережі

В даному випадку під гіперпараметрами розуміється швидкість навчання, кількість епох, розмір пакету. Відслідковується вплив цих значень на параметри моделі: loss, val_loss, час на тренування.

5.3.1 Розмір пакету

Першим було проведено тестування вплив параметра розмір пакету на показники моделі. Значення показників швидкість навчання рівна 0,0001, кількість епох 10, розмір пакету 30 та 40.

В таких умовах параметр loss на протязі всіх епох зменшується з однаковою швидкістю та в кінці досягає приблизно рівних значень. Параметр val_loss є кращим при розміру пакету рівному 40, що є перевагою в 36,7%. Проте загальний час, витрачений на тренування моделі з розміром пакету рівним 40 в 8,35 разів більший ніж моделі з розміром пакету рівним 30. Такий результат отримано через недостачу відеопам’яті у відеокарті. В таблиці 5.1 та 5.2 вказані всі порівняні параметри. На рисунку 5.1 та 5.2 зображено графіки залежності параметрів loss, val_loss від кількості епох.

Таблиця 5.1 – Параметри моделей при learning_rate=0.0001, batch_size=40, epochs=10

LR:0.0001 BS:40			
epoch	loss	val_loss	time
1	0,1110	0,0603	1410
2	0,0840	0,0534	2142
3	0,0688	0,0440	1934
4	0,0590	0,0463	1848
5	0,0563	0,0457	1725
6	0,0533	0,0424	1684
7	0,0479	0,0426	1612
8	0,0436	0,0464	1591
9	0,0430	0,0441	1868
10	0,0378	0,0378	2904
		avg time	1872
		sum (min)	312

Таблиця 5.2 – Параметри моделей при learning_rate=0.0001, batch_size=30, epochs=10

LR:0.0001 BS:30			
epoch	loss	val_loss	time
1	0,1172	0,0723	213
2	0,0870	0,0534	212
3	0,0765	0,0574	224
4	0,0672	0,0526	228
5	0,0591	0,0529	226
6	0,0523	0,0497	227
7	0,0470	0,0511	231
8	0,0432	0,0525	228
9	0,0364	0,0528	227
10	0,0320	0,0513	227
		avg time	224
		sum (min)	37

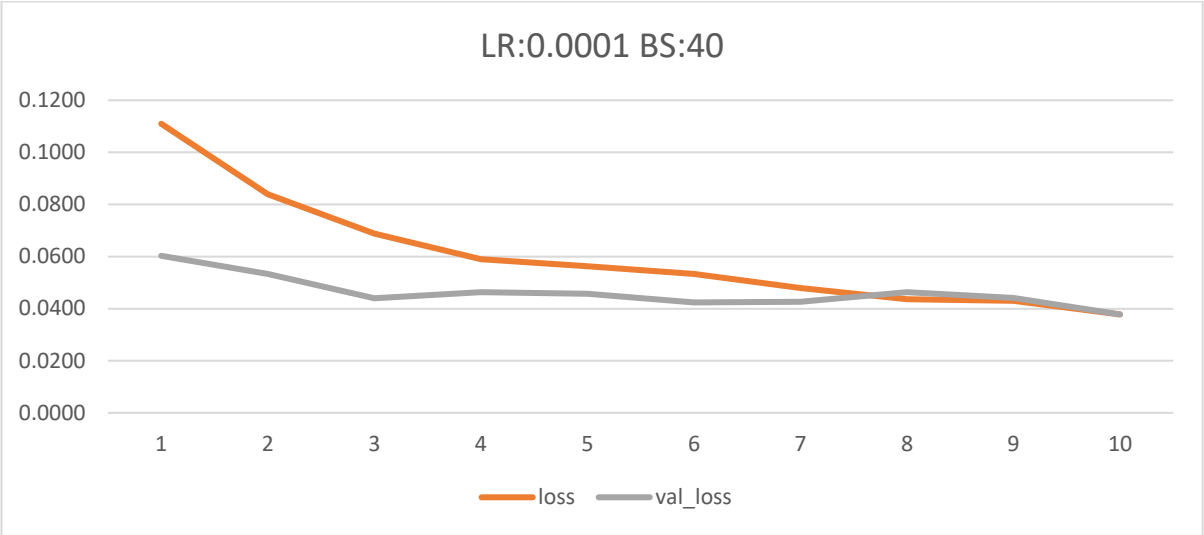


Рисунок 5.1 – Графік моделей з batch_size=40

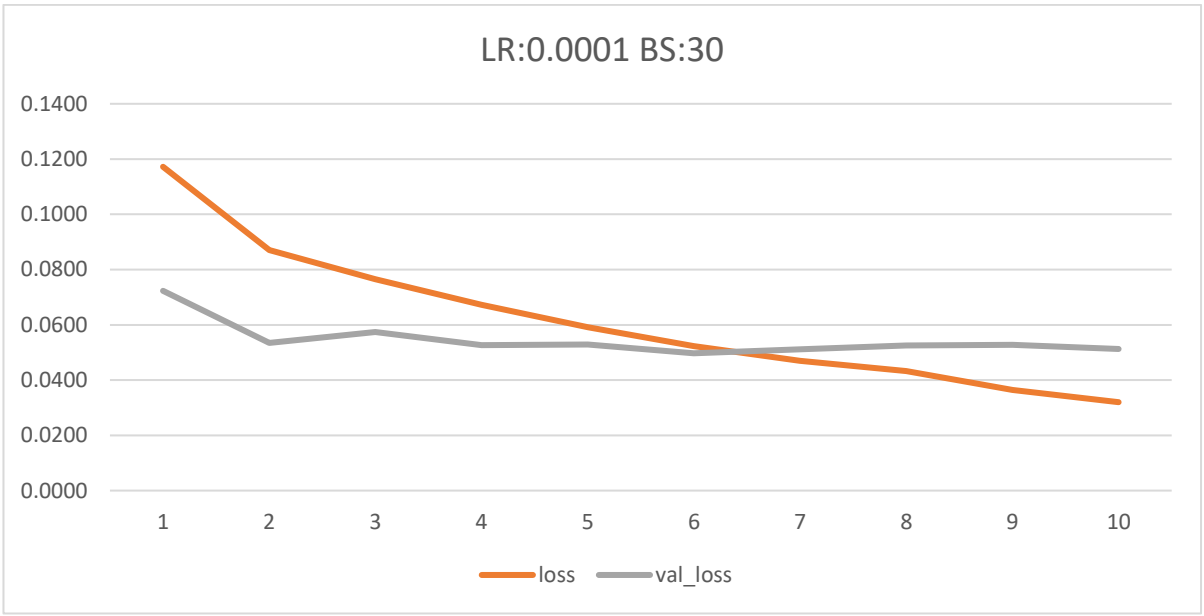


Рисунок 5.2 – Графік моделей з batch_size=30

5.3.2 Швидкість навчання

Наступним було проведено тестування вплив параметра швидкість навчання на показники моделі. Значення показників швидкість навчання рівна 0,0001 та 0,01, кількість епох 10, розмір пакету 30.

В таких умовах, модель з швидкістю навчання 0,0001, поступово зменшує параметр loss до значення 0,032, тоді як модель з швидкістю навчання 0,01 швидко зменшила параметр loss до значення 0,1387 і залишилася на

ньому. У цієї моделі спостерігається подібна тенденція з параметром val_loss, коли у першої моделі цей параметр майже не змінювався. Час тренування моделей не відрізняється. В таблиці 5.3 та 5.4 вказані всі порівняні параметри. На рисунку 5.3 та 5.4 зображено графіки залежності параметрів loss, val_loss від кількості епох.

Таблиця 5.3 – Параметри моделей при learning_rate=0.0001, batch_size=30, epochs=10

LR:0.0001 BS:30			
epoch	loss	val_loss	time
1	0,1172	0,0723	213
2	0,0870	0,0534	212
3	0,0765	0,0574	224
4	0,0672	0,0526	228
5	0,0591	0,0529	226
6	0,0523	0,0497	227
7	0,0470	0,0511	231
8	0,0432	0,0525	228
9	0,0364	0,0528	227
10	0,0320	0,0513	227
		avg time	224
		sum (min)	37

Таблиця 5.4 – Параметри моделей при learning_rate=0.01, batch_size=30, epochs=10

LR:0.01 BS:30			
epoch	loss	val_loss	time
1	2,0000	2,0000	223
2	1,2217	1,0331	223
3	0,8851	0,6351	222
4	0,4721	0,2587	223
5	0,2019	0,1173	223
6	0,1410	0,1081	223
7	0,1387	0,1081	221
8	0,1390	0,1081	224
9	0,1395	0,1079	222
10	0,1397	0,1137	221
		avg time	223
		sum (min)	37

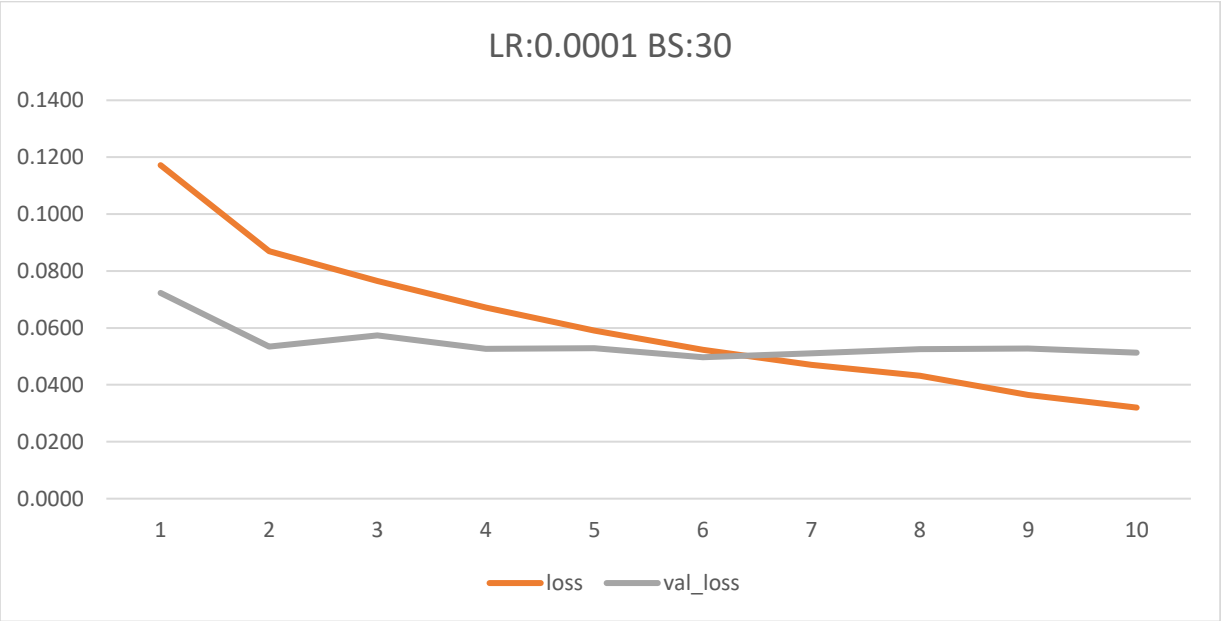
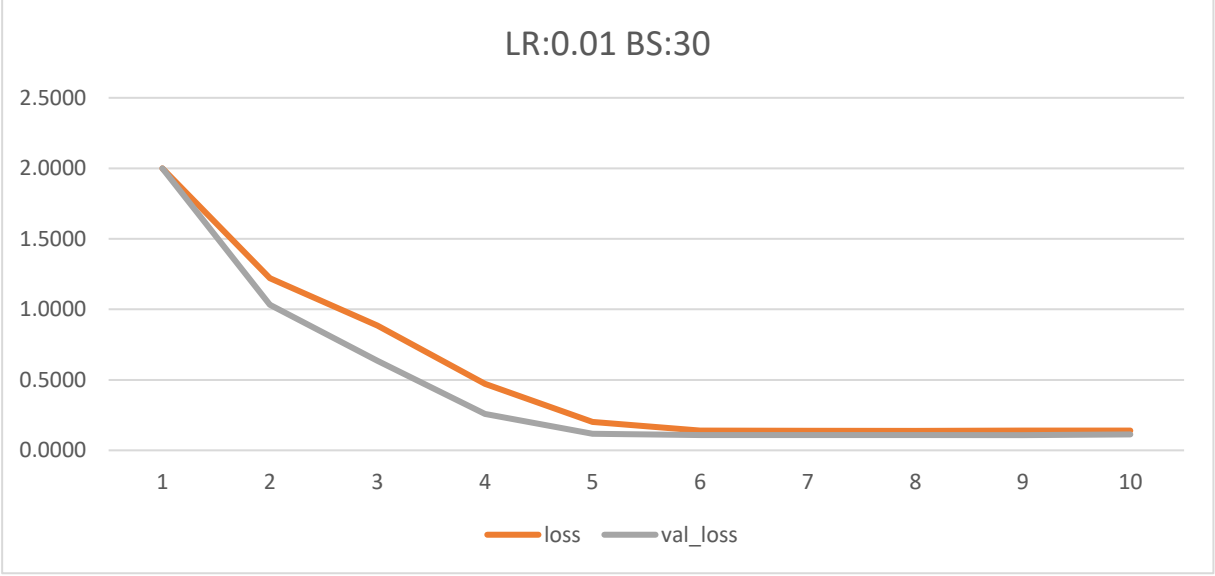


Рисунок 5.3 – Графік моделей з learning_rate=0.0001



3

Рисунок 5.4 – Графік моделей з learning_rate=0.01

Також проведено експеримент зі значеннями швидкості навчання 0,0001 та 0,00001. Показники відносно однакові. В таблиці 5.5 вказані всі порівняні параметри. На рисунку 5.5 зображено графік залежності параметрів loss, val_loss від кількості епох.

Таблиця 5.5 – Параметри моделей при learning_rate=0.00001, batch_size=30, epochs=20

LR:0.00001 BS:32			
epoch	loss	val_loss	time
1	0,1144	0,0613	271
2	0,0823	0,0548	268
3	0,0722	0,0501	268
4	0,0662	0,0488	268
5	0,0606	0,0471	268
6	0,0565	0,0483	268
7	0,0528	0,0481	268
8	0,0495	0,0488	267
9	0,0469	0,0493	268
10	0,0445	0,0452	268
11	0,0431	0,0460	268
12	0,0408	0,0466	268
13	0,0396	0,0469	267
14	0,0377	0,0470	266
15	0,0364	0,0466	267
16	0,0350	0,0463	266
17	0,0342	0,0487	267
18	0,0329	0,0467	267
19	0,0320	0,0493	266
20	0,0310	0,0463	268
		avg time	268
		sum (min)	89

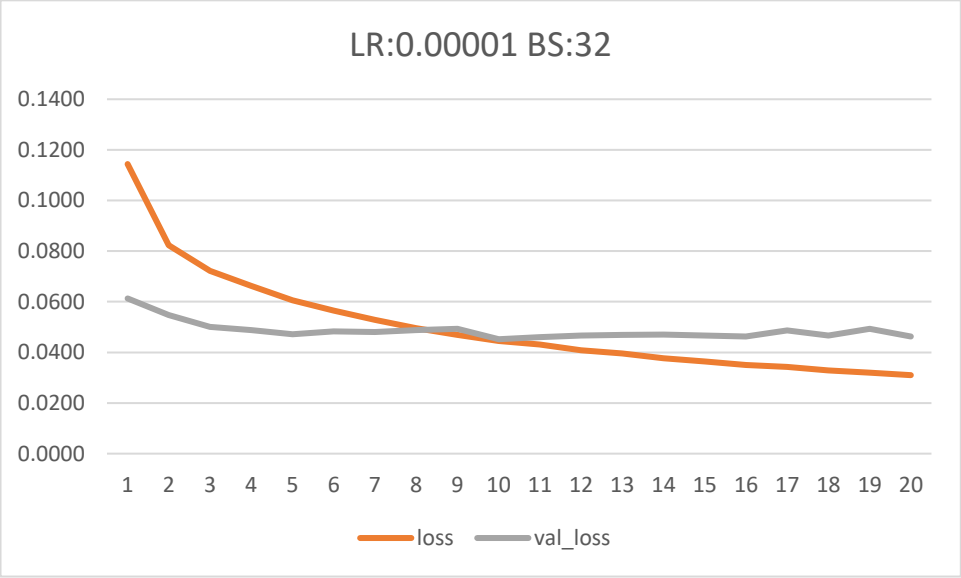


Рисунок 5.5 – Графік моделей з learning_rate=0.00001

5.3.3 Кількість епох

Наступним було проведено тестування вплив параметра кількості епох на показники моделі. Значення показників швидкість навчання рівна 0,0001, кількість епох 10, 20, 50, розмір пакету 32.

Раніше представлені графіки показують, що зі збільшенням кількості епох покращуються параметри loss та val_loss але збільшується й затрачений час на тренування. При збільшенні кількості епох до 50, значни збільшується затрачений час, а з ним і показники loss та val_loss (таблиця 5.6, рисунок 5,6).

Таблиця 5.6 – Параметри моделей при learning_rate=0.0001, batch_size=32, epochs=50

LR:0.0001 BS:32 v1			
epoch	loss	val_loss	time
1	0,1107	0,0619	266
2	0,0788	0,0734	267
3	0,0677	0,0514	268
4	0,0608	0,0451	268
5	0,0555	0,0436	268
6	0,0488	0,0436	269
7	0,0461	0,0475	270
8	0,0435	0,0454	848
9	0,0391	0,0437	278
10	0,0348	0,0420	291
40	0,0045	0,0467	266
41	0,0044	0,0461	267
42	0,0041	0,0463	266
43	0,0041	0,0454	267
44	0,0039	0,0453	265
45	0,0039	0,0451	266
46	0,0038	0,0447	266
47	0,0037	0,0464	266
48	0,0036	0,0446	266
49	0,0036	0,0457	266
50	0,0035	0,0449	267
		avg time	279
		sum (min)	233

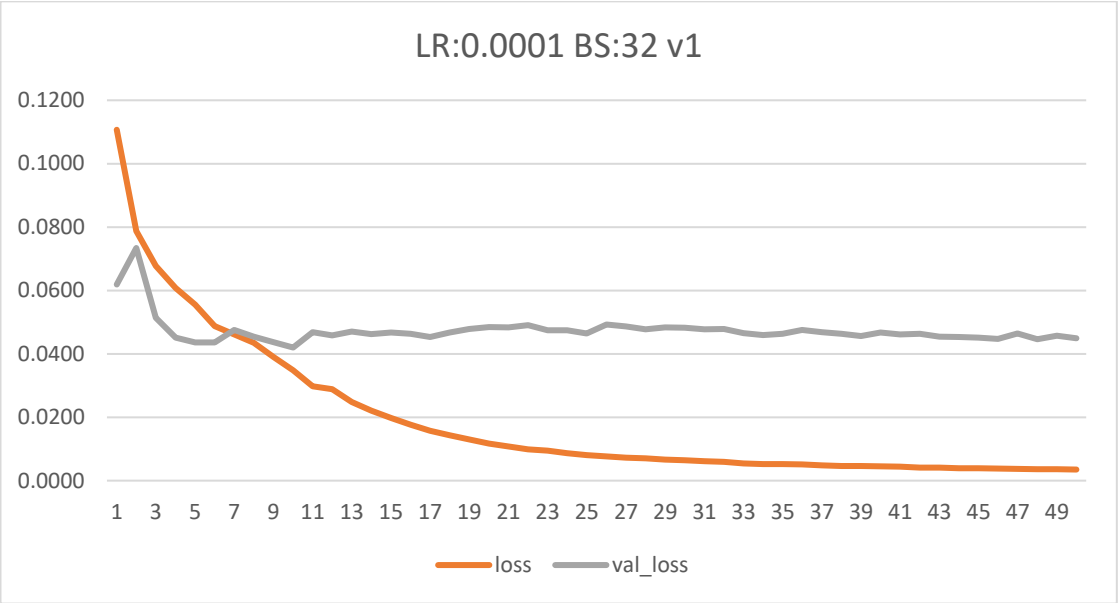


Рисунок 5.6 – Графік моделей з epochs=50

З попереднього графіка можна дізнатися, що після приблизно тридцятої епохи значення loss не змінювалося значною мірою. Тобто п’ятдесят епох вже забагато. Це припущення підтверджується повторним тестом з доповненим набором вхідних даних (таблиця 5.7, рисунок 5.7).

Таблиця 5.6 – Параметри оновлених моделей при learning_rate=0.0001, batch_size=32, epochs=50

LR:0.0001 BS:32 v2			
epoch	loss	val_loss	time
1	0,0758	0,0407	494
2	0,0226	0,0377	232
3	0,0180	0,0423	242
4	0,0162	0,0323	245
5	0,0139	0,0348	242
6	0,0132	0,0302	243
45	0,0012	0,0276	247
46	0,0011	0,0271	298
47	0,0011	0,0272	435
48	0,0012	0,0274	265
49	0,0011	0,0271	265
50	0,0010	0,0272	265
		avg time	266
		sum (min)	222

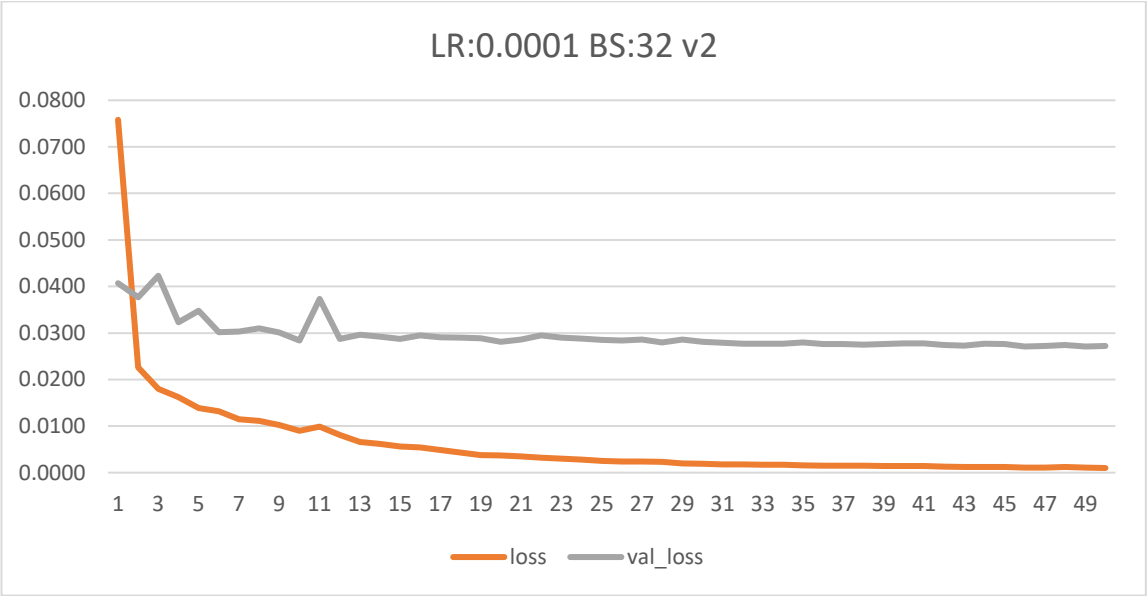


Рисунок 5.7 – Графік оновлених моделей з epochs=50

В доповнення про надмірність останніх двадцяти епох, можна ще привести приклад поведінки моделі save_at_49.h5 натренованої з даними параметрами. Ця модель виявилася однією з найкращих при тестуванні на дорозі з якої бралися дані для тренування. Але при тестуванні моделі save_at_49.h5 на незнайомій дорозі, автомобіль вперше зіткнувся через 15 секунд їзди. В порівнянні з цим, на відомій дорозі він протримався 3 хвилини. Тому ці зайві епохи не тільки непотрібні, а й шкідливі, оскільки через це сталося перенавчання. Це явище в машинному навчанні, коли модель надто точно адаптується до тренувальних даних, включаючи шум та випадкові відмінності, і тому не ефективно узагальнюється на нові, раніше не бачені дані.

Узагальнюючи, можна сказати, що найбільш ефективною є модель із значеннями: швидкість навчання 0,0001, кількість епох 30, розмір пакету 32.

5.4 Тестування найкращої моделі

На основі даних отриманих з попереднього розділу: швидкість навчання 0,0001, кількість епох 30, розмір пакету 32, було згенеровано 8 моделей для керування автомобілем. Їх перелік зображено на рисунку 5.7

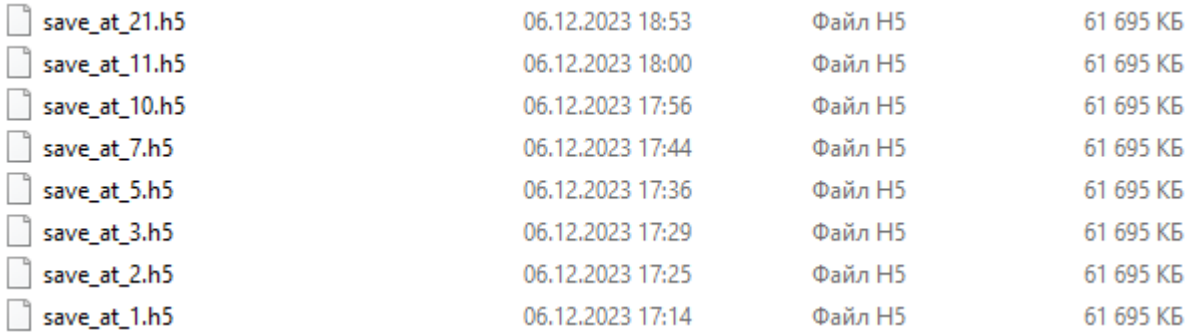


Рисунок 5.7 – Перелік тренованих моделей

При збереженні цих моделей був встановлений параметр save_best_only = True, зберігає лише тоді, коли модель вважається «найкращою», а остання найкраща модель не буде перезаписана. Використана метрика val_loss для виявлення найкращої моделі. В таблиці 5.7 вказані всі порівняні параметри.

Таблиця 5.7 – Параметри найкращих моделей при learning_rate=0.0001, batch_size=32, epochs=30

LR:0.00001 BS:32		
epoch	loss	val_loss
1	0,0429	0,0305
2	0,0158	0,0270
3	0,0136	0,0266
5	0,0121	0,0258
7	0,0116	0,0253
10	0,0097	0,0247
11	0,0096	0,0243
21	0,0070	0,0241

Зважаючи на дані таблиці можна зробити припущення про успішність деяких моделей. Різниця між значеннями val_lossне є великою в даному випадку. Найкращі результати на тренувальних даних покаже модель 21, оскільки ця модель має найкращий показник loss. Вона також може виявитися перетренованою, виходячи з великої різниці показників loss та val_loss. У моделей 2 та 3 можуть добре себе показати на валідаційних даних.

Параметром тестування є час до першого зіткнення. Тестування проводиться з встановленою максимальною швидкістю 10, 20 кілометри на годину. Також для тестування використано дві дороги: на якій проводилася генерація тренувального набору даних та нова дорога. Зібрані дані про моделі зображено в таблицях на рисунках 5.8-5.15.

save_at_ 21.h5	HILL TRACK	
	10км/год	20км/год
1	322	62
2	111	48
3	318	49
4	106	50
5	322	49
top	322	62
bot	106	48

save_at_ 21.h5	SNAKE TRACK	
	10км/год	20км/год
1	111	6
2	108	44
3	110	52
4	111	51
5	109	52
top	111	52
bot	108	6

Рисунок 5.8 – Результат тестування моделі «save_at_21.h5»

save_at_ 11.h5	HILL TRACK	
	10км/год	20км/год
1	12	39
2	14	5
3	15	4
4	13	4
5	95	4
top	95	39
bot	12	4

save_at_ 11.h5	SNAKE TRACK	
	10км/год	20км/год
1	32	16
2	62	2
3	62	2
4	62	2
5	62	2
top	62	16
bot	32	2

Рисунок 5.9 – Результат тестування моделі «save_at_11.h5»

save_at_ 10.h5	HILL TRACK	
	10км/год	20км/год
1	29	27
2	17	10
3	23	9
4	26	9
5	17	58
top	29	58
bot	17	9

save_at_ 10.h5	SNAKE TRACK	
	10км/год	20км/год
1	134	55
2	65	52
3	356	95
4	120	59
5	145	38
top	356	95
bot	65	38

Рисунок 5.10 – Результат тестування моделі «save_at_10.h5»

save_at_ 7.h5	HILL TRACK	
	10км/год	20км/год
1	31	10
2	17	31
3	16	35
4	17	11
5	17	14
top	31	35
bot	16	10

save_at_ 7.h5	SNAKE TRACK	
	10км/год	20км/год
1	75	14
2	79	53
3	91	44
4	76	50
5	198	48
top	198	53
bot	75	14

Рисунок 5.11 – Результат тестування моделі «save_at_7.h5»

save_at_ 5.h5	HILL TRACK	
	10км/год	20км/год
1	20	56
2	20	7
3	11	20
4	18	11
5	20	8
top	20	56
bot	11	7

save_at_ 5.h5	SNAKE TRACK	
	10км/год	20км/год
1	87	50
2	82	39
3	87	41
4	87	41
5	76	39
top	87	50
bot	76	39

Рисунок 5.12 – Результат тестування моделі «save_at_5.h5»

save_at_ 3.h5	HILL TRACK	
	10км/год	20км/год
1	241	92
2	217	93
3	239	91
4	245	92
5	215	92
top	245	93
bot	215	91

save_at_ 3.h5	SNAKE TRACK	
	10км/год	20км/год
1	264	113
2	220	67
3	229	104
4	241	102
5	260	112
top	264	113
bot	220	67

Рисунок 5.13 – Результат тестування моделі «save_at_3.h5

save_at_ 2.h5	HILL TRACK			save_at_ 2.h5	SNAKE TRACK	
	10км/год	20км/год			10км/год	20км/год
1	35	23		1	105	36
2	34	7		2	100	23
3	18	6		3	115	16
4	28	7		4	111	21
5	18	4		5	99	18
top	35	23		top	115	36
bot	18	4		bot	99	16

Рисунок 5.14 – Результат тестування моделі «save_at_2.h5

save_at_ 1.h5	HILL TRACK			save_at_ 1.h5	SNAKE TRACK	
	10км/год	20км/год			10км/год	20км/год
1	35	23		1	88	18
2	30	23		2	20	17
3	29	11		3	79	21
4	24	18		4	80	23
5	35	4		5	88	23
top	35	23		top	88	23
bot	24	4		bot	20	17

Рисунок 5.15 – Результат тестування моделі «save_at_1.h5

В результаті тестувань виявилося, що жодна модель не здатна завершити коло без зіткнень, та проїхати повне коло загалом. Кожна модель має місце на дорозі з яким вона обов’язково зіткнеться та не зможе виїхати назад на дорогу. Лише моделі 7, 10, 11 показали можливість від’їхати назад, але не на середину дороги з подальшим розворотом та продовженням маршруту.

Порівняння значень найбільшої та найменшої тривалості заїзду зображено на рисунках 5.16-5.19, де сині стовпці – максимальне значення, помаранчеві – мінімальне. Вісь ординат відображає час в секундах, вісь абсцис – номер моделі. HILL TRACK – назва дороги на які генерувалися тренувальні дані.

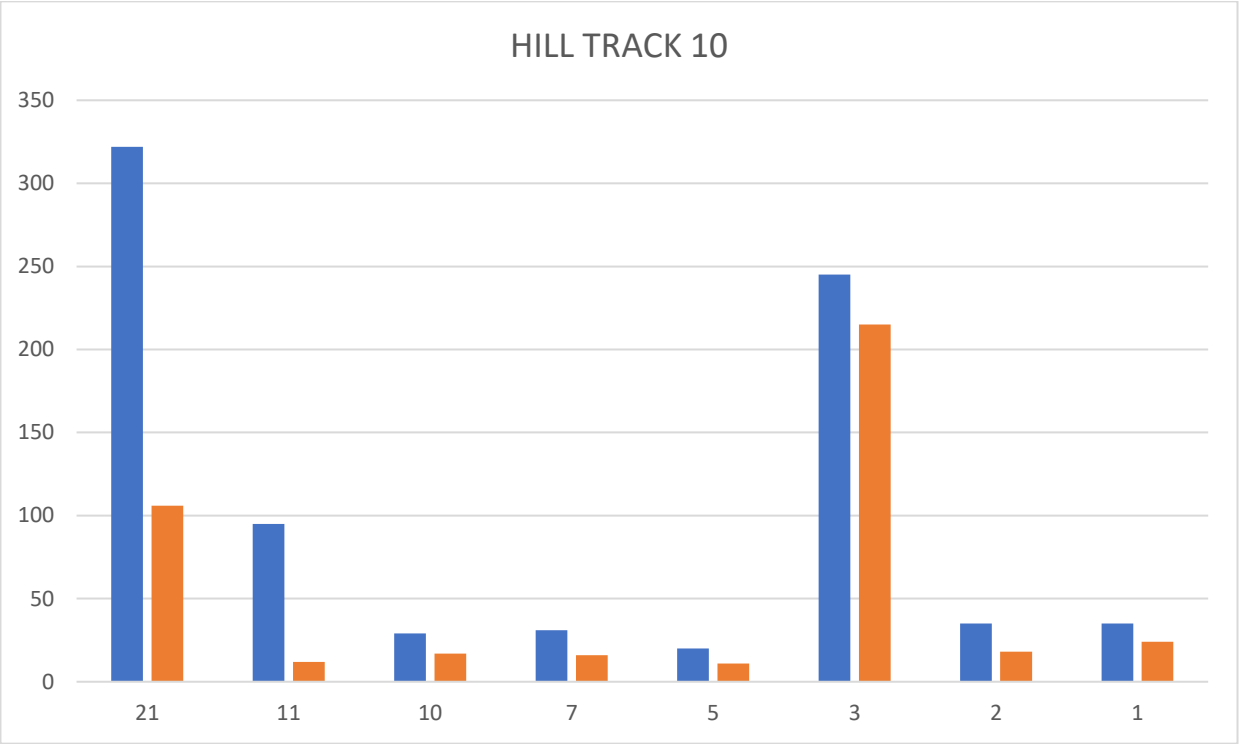


Рисунок 3.16 – Порівняння при максимальній швидкості 10км/год на дорозі «HILL TRACK»

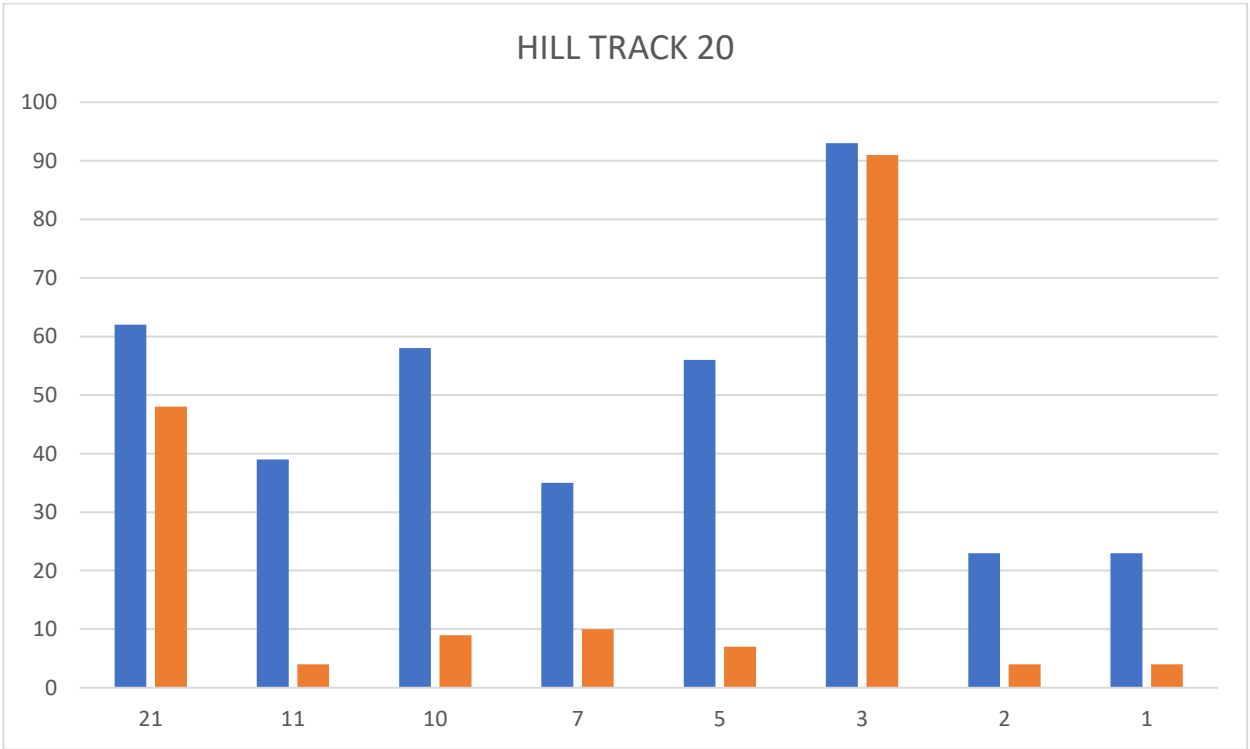


Рисунок 3.17 – Порівняння при максимальній швидкості 20км/год на дорозі «HILL TRACK»

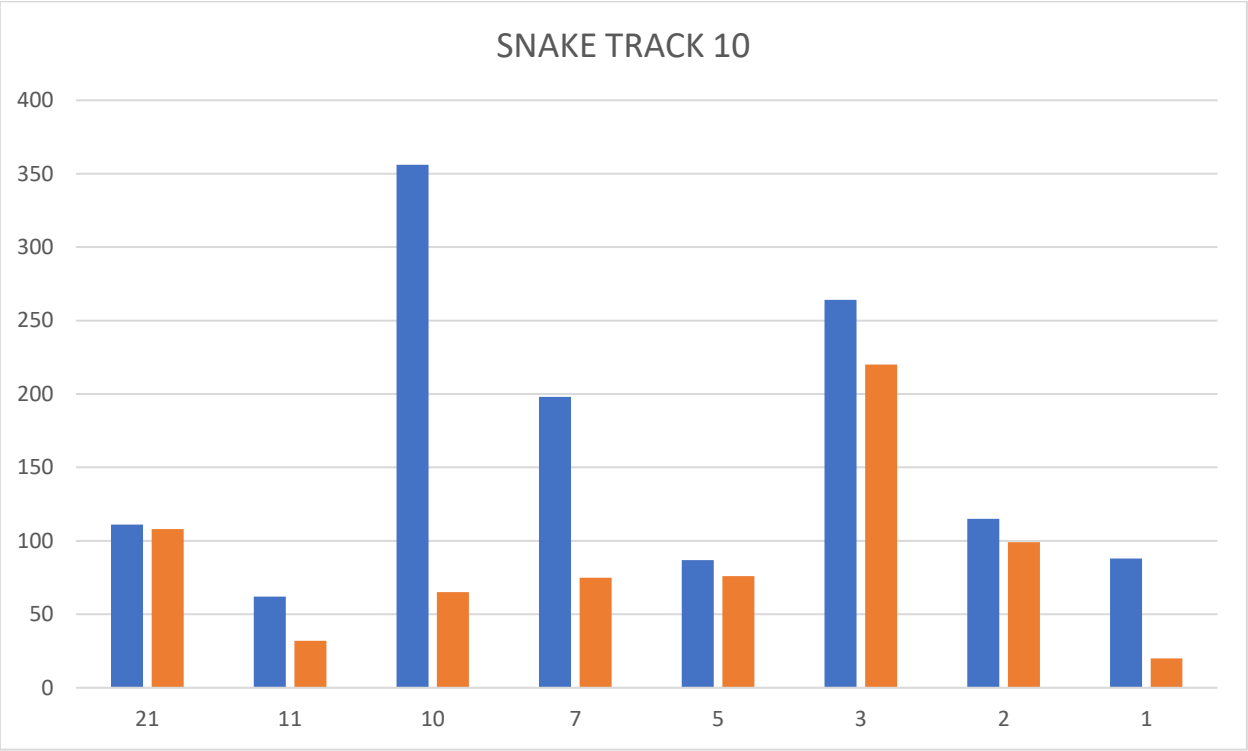


Рисунок 3.18 – Порівняння при максимальній швидкості 10км/год на дорозі «SNAKE TRACK»

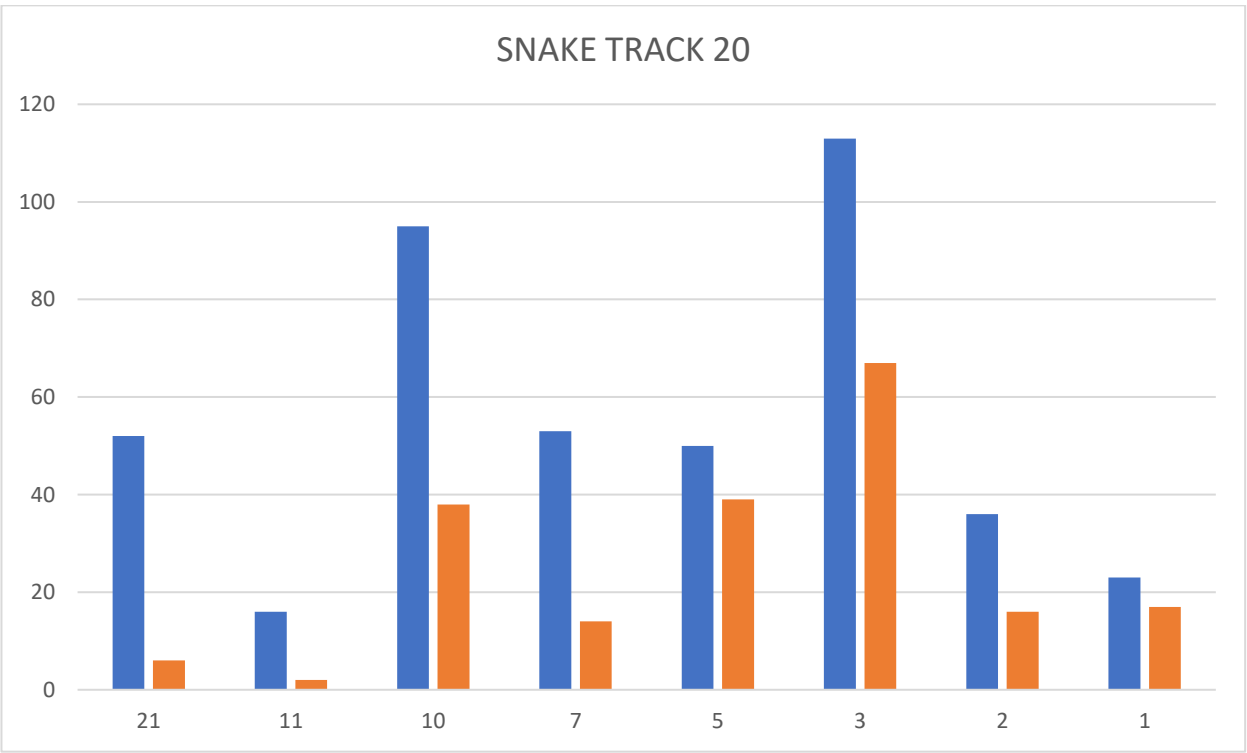


Рисунок 3.19 – Порівняння при максимальній швидкості 20км/год на дорозі «SNAKE TRACK»

Виходячи з поведінки моделей на дорозі та оперуючись на побудовані графіки можна зробити висновки про їх ефективність.

З точки зору стабільності та передбачуваності, найкращою виявилася модель під номером 3. Вона має найменшу різницю мінімального та максимального показника на протязі всіх тестів. Ця модель показує одні з найкращих результатів.

Модель під номером 21 показала гарні результати лише на тренувальній дорозі, з високим рівнем нестабільності результатів.

5.5 Висновки за розділом

Встановлено максимальну швидкість автомобіля симулятора 60КМ/год. Досліджено гіперпараметри нейронної мережі та встановлено їх значення. Проведено порівняльний аналіз моделей, отриманих з використанням досліджених гіперпараметрів. В результаті тестувань виявилося, що жодна модель не здатна завершити коло без зіткнень. Як показало дослідження, для покращення загальних результатів та параметра val_loss зокрема, необхідно збільшити набір вхідних даних.

ВИСНОВКИ

Зі своїм розвитком, нейронні мережі мають все більше галузей застосування. У галузі безпілотного керування вони застосовуються як для обробки вхідних даних з камер та різних датчиків, так і для керування транспортом. Для безпечної перевірки працездатності системи, її розміщують в симуляцію розроблену для тестування окремих її частин або системи в цілому.

В результаті роботи було побудовано симулятор системи нейромережевого керування автомобілем за допомогою ігрового рушія Unity та відкритої нейромережевої бібліотеки, написаної мовою Python – Keras, яка призначена для спрощення швидких експериментів з мережами глибокого навчання. Для досягнення поставленої мети використовувалися методи досліджень, які базуються на використанні основних положень теорії алгоритмів, теорії обчислювальних систем, теорії цифрової обробки зображень.

На початку роботи було проведено огляд проблеми, розглянуто сучасні підходи до розробки автономних транспортних засобів з використанням нейронних мереж.

Сформовано основні вимоги до симулятора:

– візуальний контраст між проїжджою частиною та узбіччям дороги;
– наявність різносторонніх поворотів;
– повна видимість дороги;
– наявність сили тертя автомобіля та дороги;
– наявність інерції автомобіля;
– данні для системи автоматичного керування збираються з камер, розташованих спереду автомобіля на капоті;

– введення даних про кут повороту та силу прискорення здійснюється за допомогою клавіатури.

Схематично зображено головне меню додатка симулятора автомобіля та користувацький інтерфейс під час керування автом. Наведено сторонні ресурси, які використовувалися при розробці симулятора. Засобами Unity та C# реалізовано взаємодію симулятора та програми нейромережевого керування автомобілем, використовуючи SocetIO. Реалізовано функцію двоетапної генерації набору даних для тренування нейронної мережі. Створено сцени для тестування навчених моделей та сцени для генерації тренувальних даних.

Розглянуто існуючі топології нейронних мереж для автономного керування, спроектовано та реалізовано власну структуру нейронної мережі. Наведено її реалізацію за допомогою TensorFlow та його надбудови Keras. Реалізовано модуль автономного керування автомобілем на основі збережених моделей глибокого навчання.

Встановлено максимальну швидкість автомобіля симулятора 60КМ/год. Досліджено гіперпараметри нейронної мережі та встановлено їх значення: швидкість навчання 0,0001, кількість епох 30, розмір пакету 32. Проведено порівняльний аналіз моделей, отриманих з використанням досліджених гіперпараметрів.

В результаті тестувань виявилося, що жодна модель не здатна завершити коло без зіткнень. Як показало дослідження, для покращення загальних результатів та параметра val_loss зокрема, необхідно збільшити набір вхідних даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Deep Understanding Tesla FSD Part 1: HydraNet [Електронний ресурс] – Режим доступу: https://saneryee-studio.medium.com/deep-understanding-tesla-fsd-part-1-hydranet-1b46106d57

2. Tesla's HydraNet - How Tesla's Autopilot Works [Електронний ресурс] – Режим доступу: https://www.thinkautonomous.ai/blog/how-tesla-autopilot-works/

3. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/329525538_ChauffeurNet_Learning_to_Drive_by_Imitating_the_Best_and_Synthesizing_the_Worst

4. DARPA, “Darpa Urban Challenge 2007,” [Електронний ресурс] – Режим доступу: https://www.darpa.mil/

5. Stanley: The Robot that Won the DARPA Grand Challenge. [Електронний ресурс] – Режим доступу: https://onlinelibrary.wiley.com/doi/epdf/10.1002/rob.20147

6. Mcculloch, Warren S., and Pitts, Walter. A Logical Calculus of the Ideas Immanent in Nervous Activity. Сполучені Штати Америки, University Press, 1943.

7. Keras API reference [Електронний ресурс] – Режим доступу: https://keras.io/api/

8. Unity Documentation [Електронний ресурс] – Режим доступу: https://docs.unity.com/

9. ImageNet Large Scale Visual Recognition Challenge [Електронний ресурс] – Режим доступу: https://www.image-net.org/challenges/LSVRC/

10. Borraz, R.; Navarro, P.J.; Fernández, C.; María Alcover, P.I. Cloud Incubator Car: A Reliable Platform for Autonomous Driving. Appl. Sci. 2018, 8, 303.

11. Rosique, F.; Navarro, P.J.; Fernández, C.; Padilla, A. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. Sensors 2019, 19, 648.

12. Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. arXiv 2016, arXiv:1604.07316.

13. Hecker, S.; Dai, D.; Van Gool, L. End-to-end learning of driving models with surround-view cameras and route planners. arXiv 2018, arXiv:1803.10158v2.

14. Mandic, D.P.; Chambers, J.A. Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures, and Stability; John Wiley: Hoboken, NJ, USA, 2001.

15. Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions. N.p., Machine Learning Mastery, 2018.

16. Goodfellow, Ian, et al. Deep Learning. Велика Британія, MIT Press, 2016.

17. Chollet, Francois. Deep Learning with Python. Сполучені Штати Америки, Manning, 2017.

18. Reed, Russell, and MarksII, Robert J. Neural smithing. Кембридж, MIT Press, 1999.

19. Christopher M. Bishop. Neural Networks for Pattern Recognition (Advanced Texts in Econometrics (Paperback)) 1st Edition. Oxford University Press, USA, 1996.

20. Socket.IO Protocol specification. SocketIO for Unity - v1.0.1. [Електронний ресурс] – Режим доступу: https://github.com/socketio/socket.io-protocol

21. Murray, Jeff W. C# Game Programming Cookbook for Unity 3D. Сполучені Штати Америки, CRC Press, 2021.

22. Puntambekar, Anuradha A. Object Oriented Programming. Індія, UNICORN Publishing Group, 2020.

23. Taher, Raihan. Hands-On Object-Oriented Programming with C#: Build Maintainable Software with Reusable Code Using C#. Велика Британія, Packt Publishing, 2019.

24. Sarcar, Vaskaran. Interactive C#: Fundamentals, Core Concepts and Patterns. Сполучені Штати Америки, Apress, 2017.

25. Mayo, Joseph. C# Unleashed. Велика Британія, Sams, 2002.

26. Kolev, Vesselin, et al. Fundamentals of Computer Programming with C#: The Bulgarian C# Book. Болгарія, Faber Publishing, 2014.

27. Griffiths, Ian. Programming C# 10. Сполучені Штати Америки, O'Reilly Media, 2022.

28. Taher, Raihan. Hands-On Object-Oriented Programming with C#: Build Maintainable Software with Reusable Code Using C#. Велика Британія, Packt Publishing, 2019.

29. Michaelis, Mark. Essential C# 8.0. Велика Британія, Pearson Education, 2020.

30. Skeet, Jonathan. C# in Depth. Сполучені Штати Америки, Manning, 2019.

31. Alls, Jason. Clean Code in C#: Refactor Your Legacy C# Code Base and Improve Application Performance by Applying Best Practices. Велика Британія, Packt Publishing, 2020.

32. Jamro, Marcin. C# Data Structures and Algorithms: Explore the Possibilities of C# for Developing a Variety of Efficient Applications. Велика Британія, Packt Publishing, 2018.

33. Lanaro, Gabriele, et al. Advanced Python Programming: Build High Performance, Concurrent, and Multi-threaded Apps with Python Using Proven Design Patterns. Індія, Packt Publishing, 2019.

34. Turner, Ryan. Python Programming: 3 Books in 1 - Ultimate Beginner's, Intermediate & Advanced Guide to Learn Python Step by Step. N.p., nelly B.L. International Consulting Limited, 2020.

35. Hetland, Magnus Lie. Beginning Python: From Novice to Professional. Україна, Apress.

36. Telles, Matt. Python for Professionals. Індія, BPB Publications, 2019.

37. Driscoll, Michael. Python Interviews: Discussions with Python Experts. Велика Британія, Packt Publishing, 2018.

38. Vaughan, Lee. Real-World Python: A Hacker's Guide to Solving Problems with Code. Сполучені Штати Америки, No Starch Press, 2020.

39. Sweigart, Al. The Big Book of Small Python Projects: 81 Easy Practice Programs. Сполучені Штати Америки, No Starch Press, 2021.

40. Raschka, Sebastian, and Mirjalili, Vahid. Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-learn, and TensorFlow 2. Велика Британія, Packt Publishing, 2019.

41. Mueller, John Paul, and Massaron, Luca. Machine Learning For Dummies. Велика Британія, Wiley, 2021.

42. Vijayvargia, Abhishek. MACHINE LEARNING WITH PYTHON: Design and Develop Machine Learning and Deep Learning Technique using real world code examples. Індія, BPB Publications, 2018.

43. Rebala, Gopinath, et al. An Introduction to Machine Learning. Німеччина, Springer International Publishing, 2019.

44. Esposito, Dino, and Esposito, Francesco. Introducing Machine Learning. Сполучені Штати Америки, Pearson Education, 2020.

45. Bonaccorso, Giuseppe. Machine Learning Algorithms. Велика Британія, Packt Publishing, 2017.

46. Machine Learning Algorithms and Applications. Сполучені Штати Америки, Wiley, 2021.

47. Eliot, Lance, and Eliot, Michael. Autonomous Vehicle Driverless Self-Driving Cars and Artificial Intelligence: Practical Advances in AI and Machine Learning. Велика Британія, LBE Press Publishing, 2017.

48. Eliot, Lance. AI Self-Driving Cars Headway: Practical Advances In Artificial Intelligence And Machine Learning. Сполучені Штати Америки, R R BOWKER LLC, 2019.

49. Ranjan, Sumit, and Senthamilarasu, Dr. S.. Applied Deep Learning and Computer Vision for Self-Driving Cars: Build Autonomous Vehicles Using Deep Neural Networks and Behavior-cloning Techniques. Велика Британія, Packt Publishing, 2020.

50. Eliot, Lance. AI Self-Driving Cars Trendsetting: Practical Advances In Artificial Intelligence And Machine Learning. N.p., R R BOWKER LLC, 2019.

51. Eliot, Lance. AI Self-Driving Cars Evolvement: Practical Advances in Artificial Intelligence and Machine Learning. Сполучені Штати Америки, LBE Press Publishing, 2019.

52. Distante, Arcangelo, and Distante, Cosimo. Handbook of Image Processing and Computer Vision: Volume 2: From Image to Pattern. Німеччина, Springer International Publishing, 2020.

53. Rever, Matthew. Computer Vision Projects with OpenCV and Python 3: Six End-to-end Projects Built Using Machine Learning with OpenCV, Python, and TensorFlow. Велика Британія, Packt Publishing, 2018.

54. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/uk-ua/dotnet/csharp.

55. Інструмент для створення діаграм [Електронний ресурс] – Режим доступу: https://app.diagrams.net/?src=about.

Додаток А
Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
С. 16, 55, 57		Невірні переліки
ПЗ (розд. 3 та 4)		Нещільне запровнення сторінок
С. 30		Заголовок підрозділу відірвано від тексту

Додаток Б

Графічна частина

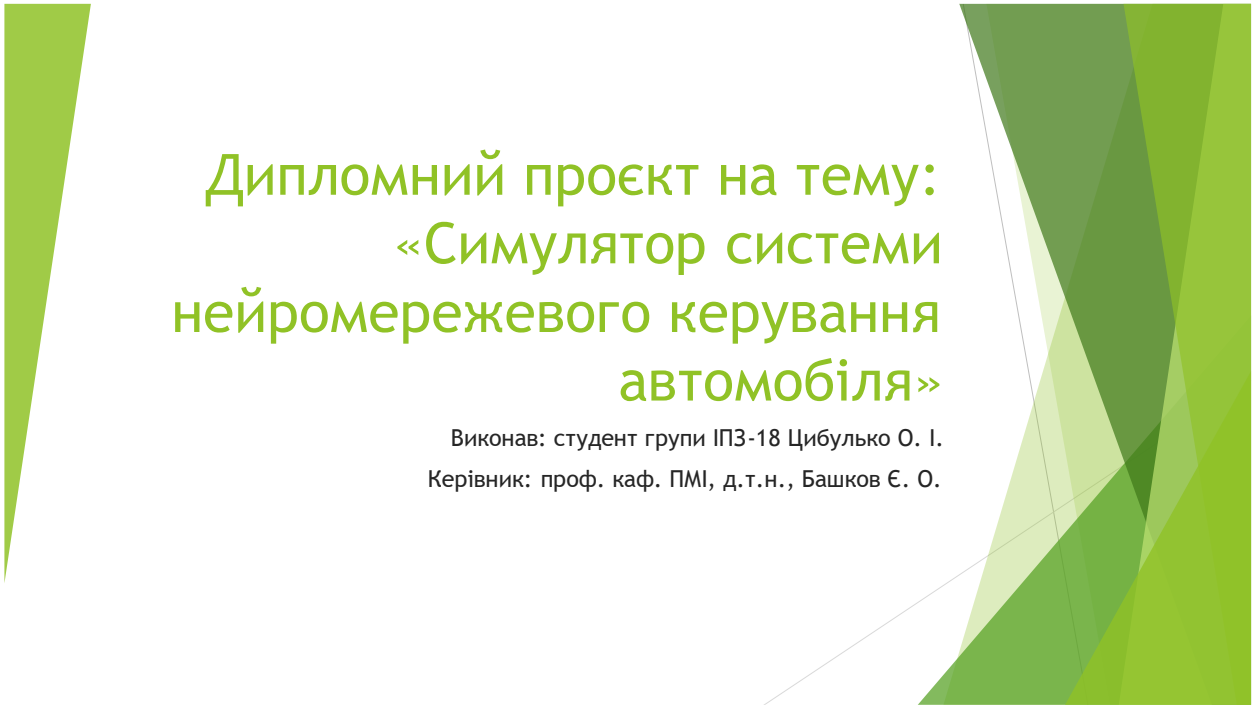


Рисунок Б.1 – Слайд 1

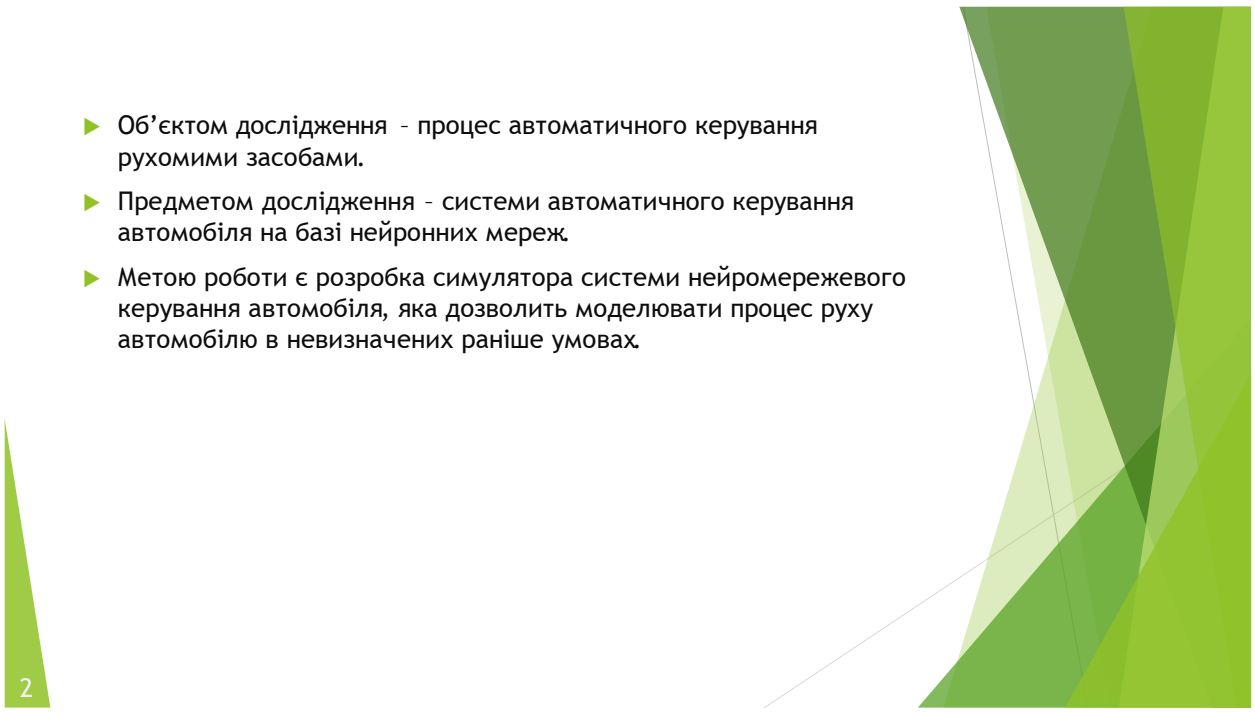


Рисунок Б.2 – Слайд 2

Основні завдання розробки

- Аналіз сучасних систем автономного керування
- Вибір інструментів для розробки дослідницького зразка і вбудовування моделей штучного інтелекту
- Визначення вимог до додатка
- Проєктування системи автономного керування автомобіля
- Розробка і опис системи генерації тренувальних даних
- Розробка і опис системи комунікації симулятора та модуля автономного керування
- Розробка симулятора
- Розробка модуля автономного керування
- Тестування і оцінка розробки

3

Рисунок Б.3 – Слайд 3

Огляд проблеми

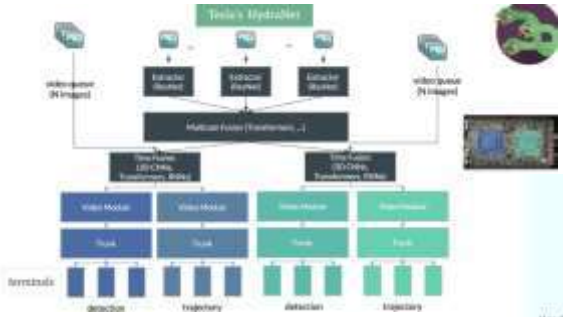
Безпілотний автомобіль Uber

Безпілотний автомобіль Waymo

4

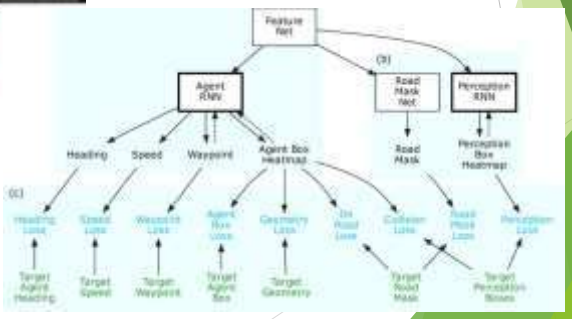
Рисунок Б.4 – Слайд 4

Існуючі системи безпілотного керування на основі нейронних мереж



HydraNet

ChauffeurNet

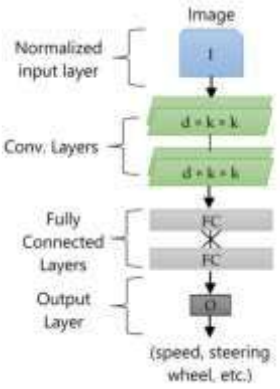


5

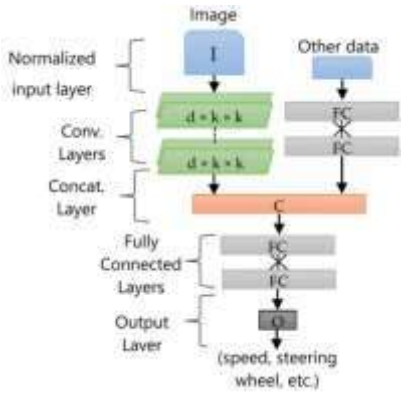
Рисунок Б.5 – Слайд 5

Топології нейронних мереж для автономного керування

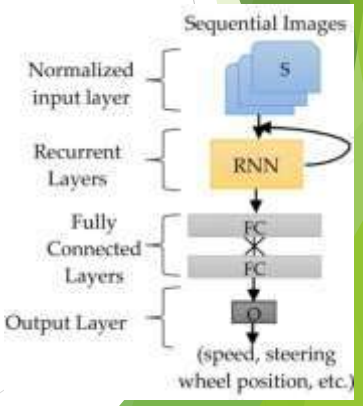
SiD-e2e



MiD-e2e



SeD-e2e

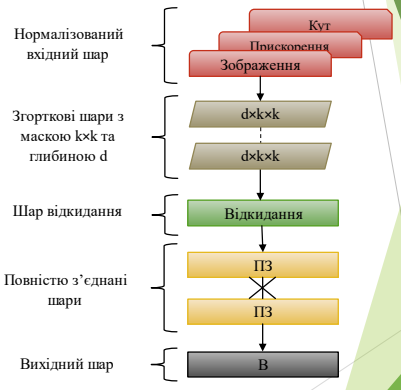


6

Рисунок Б.6 – Слайд 6

Загальні вимоги

- візуальний контраст між проїжджою частиною та узбіччям дороги;
- наявність різносторонніх поворотів;
- повна видимість дороги;
- наявність сили тертя автомобіля та дороги;
- наявність інерції автомобіля;
- данні для системи автоматичного керування збираються з єдиної камери розташованої спереду автомобіля на капоті;
- введення даних про кут повороту та силу прискорення здійснюється за допомогою клавіатури.



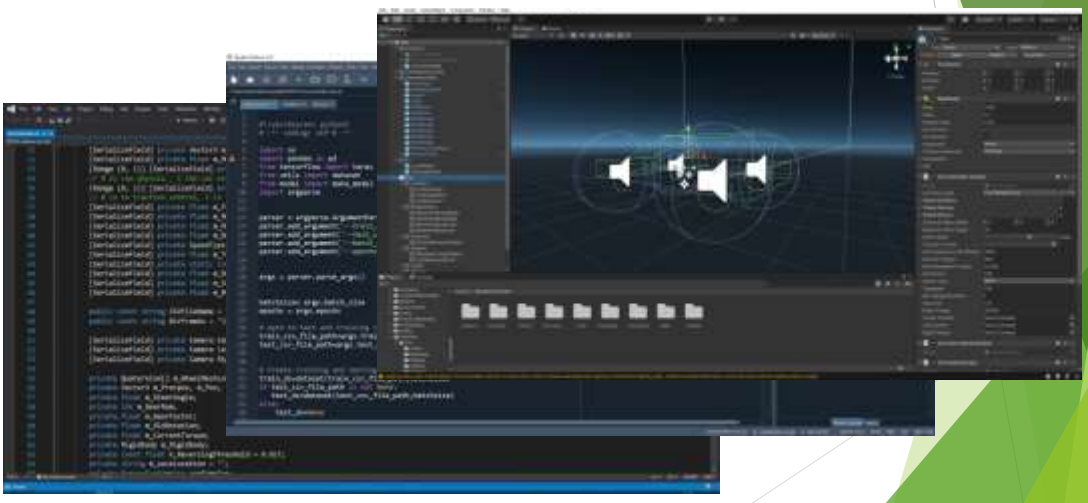
7

Рисунок Б.7 – Слайд 7

Засоби розробки

TensorFlow

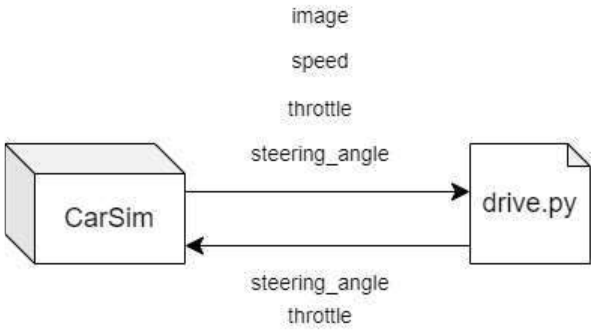
Keras



8

Рисунок Б.8 – Слайд 8

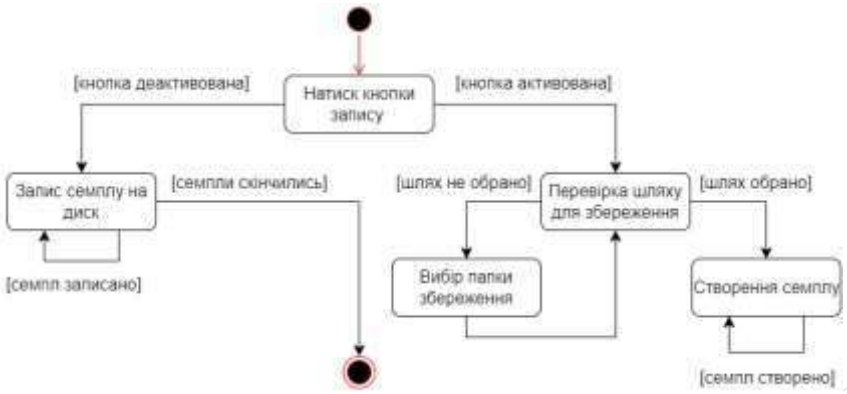
Система обміну даними



9

Рисунок Б.9 – Слайд 9

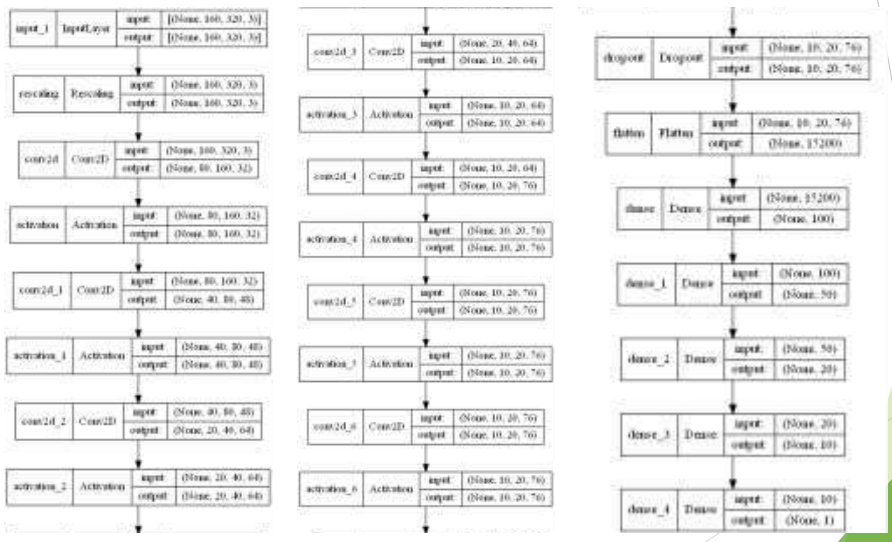
Діаграма станів алгоритму генерації тренувальни даних



10

Рисунок Б.10 – Слайд 10

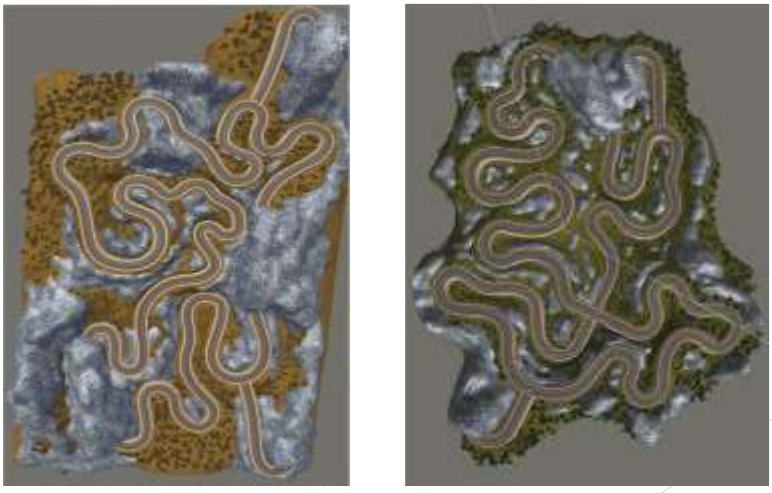
Структура отриманої моделі



11

Рисунок Б.11 – Слайд 11

3D моделі доріг



12

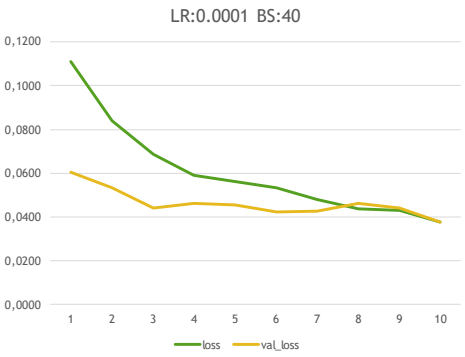
Рисунок Б.12 – Слайд 12

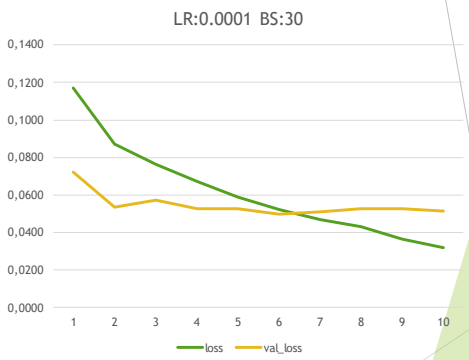
Користувацький інтерфейс

13

Рисунок Б.13 – Слайд 13

Дослідження параметру розмір пакету





14

Рисунок Б.14 – Слайд 14

Рисунок Б.15 – Слайд 15

Рисунок Б.16 – Слайд 16

Рисунок Б.17 – Слайд 17

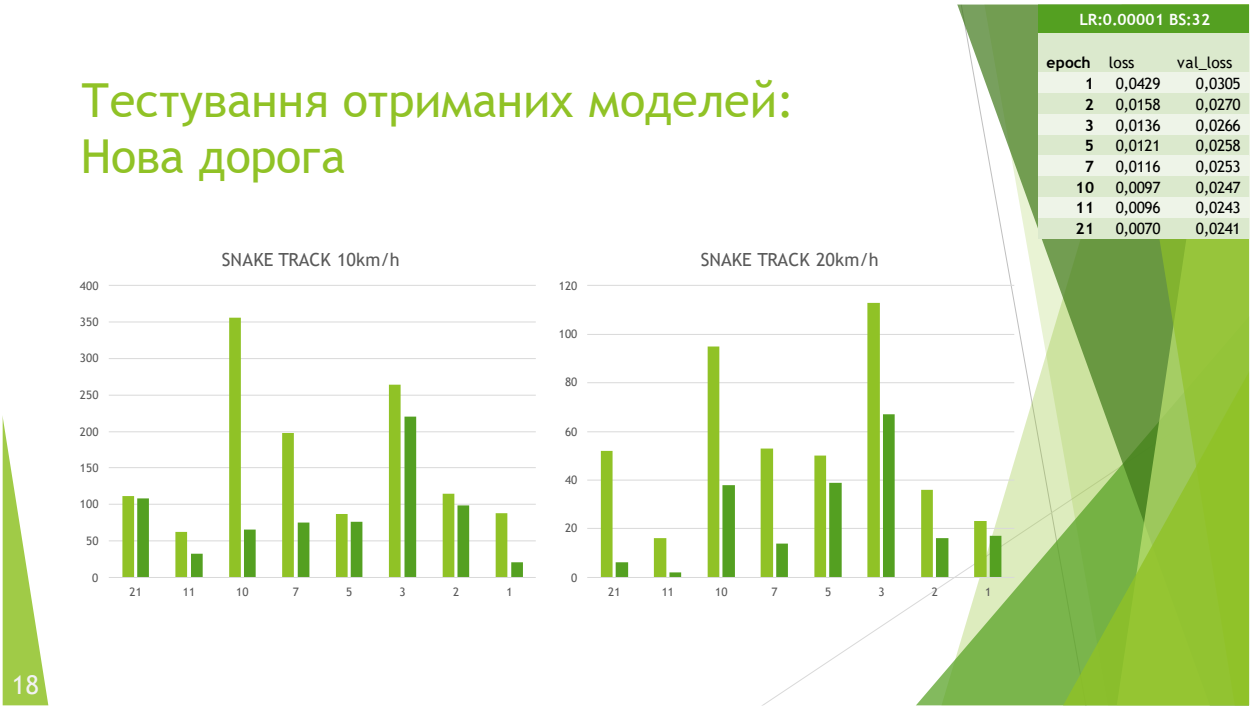


Рисунок Б.18 – Слайд 18

Можливі вдосконалення

- Збільшити обсягу тренувальних даних
- Вдосконалити архітектури нейронної мережі
- Створити нові рівні з різними дорожніми умовами
- Впровадити нові модулі для побудови маршруту, розпізнавання дорожніх знаків.

Рисунок Б.19 – Слайд 19

Висновки

В результаті роботи було побудовано симулятор системи нейромережевого керування автомобілем за допомогою ігрового рушія Unity та відкритої нейромережевої бібліотеки, написаної мовою Python– Keras.

Проведено огляд проблеми, розглянуто сучасні підходи до розробки автономних транспортних засобів з використанням нейронних мереж. Сформовано основні вимоги до симулятора.

Реалізовано функцію двоетапної генерації набору даних для тренування нейронної мережі. Створено сцени для тестування навчених моделей та сцени для генерації тренувальних даних.

Реалізовано модуль автономного керування автомобілем на основі згенерованих моделей глибокого навчання.

Проведено порівняльний аналіз моделей, отриманих з використанням досліджених гіперпараметрів. В результаті тестувань виявилося, що жодна модель не здатна завершити коло без зіткнень. Для покращення загальних результатів необхідно збільшити набір вхідних даних.

Рисунок Б.20 – Слайд 20

Додаток В.1

Програмний код програми симулятора

CarController.cs

```
using System;
using System.Collections;
using System.IO;
using UnityEngine;
using UnityEngine.EventSystems;
using System.Collections.Generic;



namespace UnityStandardAssets.Vehicles.Car
{
   internal enum CarDriveType
   {
      FrontWheelDrive,
      RearWheelDrive,
      FourWheelDrive
   }

   internal enum SpeedType
   {
      MPH,
      KPH
   }

   public class CarController : MonoBehaviour
   {
      [SerializeField] private CarDriveType m_CarDriveType = CarDriveType.FourWheelDrive;
      [SerializeField] private WheelCollider[] m_WheelColliders = new WheelCollider[4];
      [SerializeField] private GameObject[] m_WheelMeshes = new GameObject[4];
      [SerializeField] private WheelEffects[] m_WheelEffects = new WheelEffects[4];
      [SerializeField] private Vector3 m_CentreOfMassOffset;
      [SerializeField] private float m_MaximumSteerAngle;
      [Range (0, 1)] [SerializeField] private float m_SteerHelper;
      [Range (0, 1)] [SerializeField] private float m_TractionControl;
      [SerializeField] private float m_FullTorqueOverAllWheels;
      [SerializeField] private float m_ReverseTorque;
      [SerializeField] private float m_MaxHandbrakeTorque;
      [SerializeField] private float m_Downforce = 100f;
      [SerializeField] private SpeedType m_SpeedType;
      [SerializeField] private float m_Topspeed = 200;
      [SerializeField] private static int NoOfGears = 5;
      [SerializeField] private float m_RevRangeBoundary = 1f;
      [SerializeField] private float m_SlipLimit;
      [SerializeField] private float m_BrakeTorque;
```

```
public const string CSVFileName = "driving_log.csv";
public const string DirFrames = "IMG";

[SerializeField] private Camera CenterCamera;
[SerializeField] private Camera LeftCamera;
[SerializeField] private Camera RightCamera;

private Quaternion[] m_WheelMeshLocalRotations;
private Vector3 m_Prevpos, m_Pos;
private float m_SteerAngle;
private int m_GearNum;
private float m_GearFactor;
private float m_OldRotation;
private float m_CurrentTorque;
private Rigidbody m_Rigidbody;
private const float k_ReversingThreshold = 0.01f;
private string m_saveLocation = "";
private Queue<CarSample> carSamples;
        private int TotalSamples;
        private bool isSaving;
        private Vector3 saved_position;
        private Quaternion saved_rotation;

void OnCollisionEnter(Collision collision)
{
   isCollided = true;
}

public bool isCollided { get; private set; }

public bool Skidding { get; private set; }

public float BrakeInput { get; private set; }

private bool m_isRecording = false;
public bool IsRecording
{
   get
   {
      return m_isRecording;
   }

   set
   {
      m_isRecording = value;
      if(value == true)
      {
                                    Debug.Log("Starting to record");
                                    carSamples = new Queue<CarSample>();
                                    StartCoroutine(Sample());
      }
                              else
```

```
            {
                Debug.Log("Stopping record");
                StopCoroutine(Sample());
                Debug.Log("Writing to disk");
                                        saved_position = transform.position;
                                        saved_rotation = transform.rotation;
                                        TotalSamples = carSamples.Count;
                                        isSaving = true;
                                        StartCoroutine(WriteSamplesToDisk());

            };
        }

    }


                public bool checkSaveLocation()
                {
                        if (m_saveLocation != "")
                        {
                                return true;
                        }
                        else
                        {
                                SimpleFileBrowser.ShowSaveDialog (OpenFolder, null, true, null, "Select
Output Folder", "Select");
                        }
                        return false;
                }

    public float CurrentSteerAngle {
        get { return m_SteerAngle; }
        set { m_SteerAngle = value; }
    }

    public float CurrentSpeed{ get { return m_Rigidbody.velocity.magnitude * 2.23693629f; } }

    public float MaxSpeed{ get { return m_Topspeed; } }

    public float Revs { get; private set; }

    public float AccelInput { get; set; }

    private void Start ()
    {
        m_WheelMeshLocalRotations = new Quaternion[4];
        for (int i = 0; i < 4; i++) {
            m_WheelMeshLocalRotations [i] = m_WheelMeshes [i].transform.localRotation;
        }
        m_WheelColliders [0].attachedRigidbody.centerOfMass = m_CentreOfMassOffset;

        m_MaxHandbrakeTorque = float.MaxValue;
```

```
        m_Rigidbody = GetComponent<Rigidbody> ();
        m_CurrentTorque = m_FullTorqueOverAllWheels - (m_TractionControl *
m_FullTorqueOverAllWheels);
    }

    private void GearChanging ()
    {
        float f = Mathf.Abs (CurrentSpeed / MaxSpeed);
        float upgearlimit = (1 / (float)NoOfGears) * (m_GearNum + 1);
        float downgearlimit = (1 / (float)NoOfGears) * m_GearNum;

        if (m_GearNum > 0 && f < downgearlimit) {
            m_GearNum--;
        }

        if (f > upgearlimit && (m_GearNum < (NoOfGears - 1))) {
            m_GearNum++;
        }
    }

    private static float CurveFactor (float factor)
    {
        return 1 - (1 - factor) * (1 - factor);
    }

    private static float ULerp (float from, float to, float value)
    {
        return (1.0f - value) * from + value * to;
    }


    private void CalculateGearFactor ()
    {
        float f = (1 / (float)NoOfGears);
        var targetGearFactor = Mathf.InverseLerp (f * m_GearNum, f * (m_GearNum + 1), Mathf.Abs
(CurrentSpeed / MaxSpeed));
        m_GearFactor = Mathf.Lerp (m_GearFactor, targetGearFactor, Time.deltaTime * 5f);
    }


    private void CalculateRevs ()
    {
        CalculateGearFactor ();
        var gearNumFactor = m_GearNum / (float)NoOfGears;
        var revsRangeMin = ULerp (0f, m_RevRangeBoundary, CurveFactor (gearNumFactor));
        var revsRangeMax = ULerp (m_RevRangeBoundary, 1f, gearNumFactor);
        Revs = ULerp (revsRangeMin, revsRangeMax, m_GearFactor);
    }

    public void Move (float steering, float accel, float footbrake, float handbrake)
    {
        for (int i = 0; i < 4; i++) {
            Quaternion quat;
```

```
        Vector3 position;
        m_WheelColliders [i].GetWorldPose (out position, out quat);
        m_WheelMeshes [i].transform.position = position;
        m_WheelMeshes [i].transform.rotation = quat;
    }

    steering = Mathf.Clamp (steering, -1, 1);
    AccelInput = accel = Mathf.Clamp (accel, 0, 1);
    BrakeInput = footbrake = -1 * Mathf.Clamp (footbrake, -1, 0);
    handbrake = Mathf.Clamp (handbrake, 0, 1);

    m_SteerAngle = steering * m_MaximumSteerAngle;
    m_WheelColliders [0].steerAngle = m_SteerAngle;
    m_WheelColliders [1].steerAngle = m_SteerAngle;



    SteerHelper ();
    ApplyDrive (accel, footbrake);
    CapSpeed ();

    if (handbrake > 0f)
    {
        var hbTorque = handbrake * m_MaxHandbrakeTorque;
        m_WheelColliders [2].brakeTorque = hbTorque;
        m_WheelColliders [3].brakeTorque = hbTorque;
    }

    CalculateRevs ();
    GearChanging ();

    AddDownForce ();
    CheckForWheelSpin ();
    TractionControl ();
}

private void CapSpeed ()
{
    float speed = m_Rigidbody.velocity.magnitude;
    switch (m_SpeedType)
    {
        case SpeedType.MPH:
            speed *= 2.23693629f;
            if (speed > m_Topspeed)
                m_Rigidbody.velocity = (m_Topspeed / 2.23693629f) * m_Rigidbody.velocity.normalized;
            break;

        case SpeedType.KPH:
            speed *= 3.6f;
            if (speed > m_Topspeed)
                m_Rigidbody.velocity = (m_Topspeed / 3.6f) * m_Rigidbody.velocity.normalized;
            break;
    }
```

```
}


private void ApplyDrive (float accel, float footbrake)
{

   float thrustTorque;
   switch (m_CarDriveType)
   {
      case CarDriveType.FourWheelDrive:
         thrustTorque = accel * (m_CurrentTorque / 4f);
         for (int i = 0; i < 4; i++)
         {
            m_WheelColliders [i].motorTorque = thrustTorque;
         }
         break;

      case CarDriveType.FrontWheelDrive:
         thrustTorque = accel * (m_CurrentTorque / 2f);
         m_WheelColliders [0].motorTorque = m_WheelColliders [1].motorTorque = thrustTorque;
         break;

      case CarDriveType.RearWheelDrive:
         thrustTorque = accel * (m_CurrentTorque / 2f);
         m_WheelColliders [2].motorTorque = m_WheelColliders [3].motorTorque = thrustTorque;
         break;

   }

   for (int i = 0; i < 4; i++) {
      if (CurrentSpeed > 5 && Vector3.Angle (transform.forward, m_Rigidbody.velocity) < 50f)
      {
         m_WheelColliders [i].brakeTorque = m_BrakeTorque * footbrake;
      }
      else if (footbrake > 0)
      {
         m_WheelColliders [i].brakeTorque = 0f;
         m_WheelColliders [i].motorTorque = -m_ReverseTorque * footbrake;
      }
   }
}


private void SteerHelper ()
{
   for (int i = 0; i < 4; i++)
   {
      WheelHit wheelhit;
      m_WheelColliders [i].GetGroundHit (out wheelhit);
      if (wheelhit.normal == Vector3.zero)
         return;
   }
```

```
        if (Mathf.Abs (m_OldRotation - transform.eulerAngles.y) < 10f)
        {
            var turnadjust = (transform.eulerAngles.y - m_OldRotation) * m_SteerHelper;
            Quaternion velRotation = Quaternion.AngleAxis (turnadjust, Vector3.up);
            m_Rigidbody.velocity = velRotation * m_Rigidbody.velocity;
        }
        m_OldRotation = transform.eulerAngles.y;
    }


    private void AddDownForce ()
    {
        m_WheelColliders [0].attachedRigidbody.AddForce (-transform.up * m_Downforce *
        m_WheelColliders [0].attachedRigidbody.velocity.magnitude);
    }


    private void CheckForWheelSpin ()
    {
        // loop through all wheels
        for (int i = 0; i < 4; i++)
        {
            WheelHit wheelHit;
            m_WheelColliders [i].GetGroundHit (out wheelHit);

            if (Mathf.Abs (wheelHit.forwardSlip) >= m_SlipLimit || Mathf.Abs (wheelHit.sidewaysSlip) >=
m_SlipLimit)
            {
                m_WheelEffects [i].EmitTyreSmoke ();
                continue;
            }

            if (m_WheelEffects [i].PlayingAudio)
            {
                m_WheelEffects [i].StopAudio ();
            }
            m_WheelEffects [i].EndSkidTrail ();
        }
    }

    private void TractionControl ()
    {
        WheelHit wheelHit;
        switch (m_CarDriveType)
        {
            case CarDriveType.FourWheelDrive:
                for (int i = 0; i < 4; i++)
                {
                    m_WheelColliders [i].GetGroundHit (out wheelHit);

                    AdjustTorque (wheelHit.forwardSlip);
                }
                break;
```

```
            case CarDriveType.RearWheelDrive:
               m_WheelColliders [2].GetGroundHit (out wheelHit);
               AdjustTorque (wheelHit.forwardSlip);

               m_WheelColliders [3].GetGroundHit (out wheelHit);
               AdjustTorque (wheelHit.forwardSlip);
               break;

            case CarDriveType.FrontWheelDrive:
               m_WheelColliders [0].GetGroundHit (out wheelHit);
               AdjustTorque (wheelHit.forwardSlip);

               m_WheelColliders [1].GetGroundHit (out wheelHit);
               AdjustTorque (wheelHit.forwardSlip);
               break;
         }
      }


      private void AdjustTorque (float forwardSlip)
      {
         if (forwardSlip >= m_SlipLimit && m_CurrentTorque >= 0)
         {
            m_CurrentTorque -= 10 * m_TractionControl;
         }
         else
         {
            m_CurrentTorque += 10 * m_TractionControl;
            if (m_CurrentTorque > m_FullTorqueOverAllWheels)
            {
               m_CurrentTorque = m_FullTorqueOverAllWheels;
            }
         }
      }


                  public IEnumerator WriteSamplesToDisk()
                  {
                        yield return new WaitForSeconds(0.000f);
                        if (carSamples.Count > 0)
      {
                              CarSample sample = carSamples.Dequeue();

                              transform.position = sample.position;
                              transform.rotation = sample.rotation;

                              string centerPath = WriteImage (CenterCamera, "center",
sample.timeStamp);
                              string leftPath = WriteImage (LeftCamera, "left", sample.timeStamp);
                              string rightPath = WriteImage (RightCamera, "right",
sample.timeStamp);
```

```
                                string row = string.Format ("{0},{1},{2},{3},{4},{5},{6}\n", centerPath,
leftPath, rightPath, sample.steeringAngle, sample.throttle, sample.brake, sample.speed);
                                File.AppendAllText (Path.Combine (m_saveLocation, CSVFileName),
row);
                        }
                        if (carSamples.Count > 0)
        {
                                StartCoroutine(WriteSamplesToDisk());
                        }
                        else
                        {
                                StopCoroutine(WriteSamplesToDisk());
                                isSaving = false;

                                transform.position = saved_position;
                                transform.rotation = saved_rotation;
                                m_Rigidbody.velocity = new Vector3(0f,-10f,0f);
                                Move(0f, 0f, 0f, 0f);

                        }
                }

                public float getSavePercent()
                {
                        return (float)(TotalSamples-carSamples.Count)/TotalSamples;
                }

                public bool getSaveStatus()
                {
                        return isSaving;
                }


    public IEnumerator Sample()
    {
      yield return new WaitForSeconds(0.0666666666666667f);

      if (m_saveLocation != "")
      {
        CarSample sample = new CarSample();

        sample.timeStamp = System.DateTime.Now.ToString("yyyy_MM_dd_HH_mm_ss_fff");
        sample.steeringAngle = m_SteerAngle / m_MaximumSteerAngle;
        sample.throttle = AccelInput;
        sample.brake = BrakeInput;
        sample.speed = CurrentSpeed;
        sample.position = transform.position;
        sample.rotation = transform.rotation;

        carSamples.Enqueue(sample);

        sample = null;
      }
```

```
        if (IsRecording)
        {
            StartCoroutine(Sample());
        }

    }

    private void OpenFolder(string location)
    {
        m_saveLocation = location;
        Directory.CreateDirectory (Path.Combine(m_saveLocation, DirFrames));
    }

    private string WriteImage (Camera camera, string prepend, string timestamp)
    {
        camera.Render();
        RenderTexture targetTexture = camera.targetTexture;
        RenderTexture.active = targetTexture;
        Texture2D texture2D = new Texture2D (targetTexture.width, targetTexture.height,
TextureFormat.RGB24, false);
        texture2D.ReadPixels (new Rect (0, 0, targetTexture.width, targetTexture.height), 0, 0);
        texture2D.Apply ();
        byte[] image = texture2D.EncodeToJPG ();
        UnityEngine.Object.DestroyImmediate (texture2D);
        string directory = Path.Combine(m_saveLocation, DirFrames);
        string path = Path.Combine(directory, prepend + "_" + timestamp + ".jpg");
        File.WriteAllBytes (path, image);
        image = null;
        return path;
    }
  }

  internal class CarSample
  {
    public Quaternion rotation;
    public Vector3 position;
    public float steeringAngle;
    public float throttle;
    public float brake;
    public float speed;
    public string timeStamp;
  }

}
```

CarRemoteControl.cs

```
using System;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

namespace UnityStandardAssets.Vehicles.Car
{
   [RequireComponent(typeof(CarController))]
   public class CarRemoteControl : MonoBehaviour
   {
      private CarController m_Car;

      public float SteeringAngle { get; set; }
      public float Acceleration { get; set; }
      private Steering s;

      private void Awake()
      {
         m_Car = GetComponent<CarController>();
         s = new Steering();
         s.Start();
      }

      private void FixedUpdate()
      {
         if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.S))
         {
            s.UpdateValues();
            m_Car.Move(s.H, s.V, s.V, 0f);
         }
         else
         {
                                    m_Car.Move(SteeringAngle, Acceleration, Acceleration, 0f);
         }
      }
   }
}
```

CarUserContral.cs

```
using UnityEngine;
using System.Collections;


namespace UnityStandardAssets.Vehicles.Car
{
   [RequireComponent(typeof(CarController))]
   public class CarUserControl : MonoBehaviour
   {
      private CarController m_Car;
      private Steering s;
```

```
    private void Awake()
    {
      m_Car = GetComponent<CarController>();
      s = new Steering();
      s.Start();
    }

    private void FixedUpdate()
    {
      s.UpdateValues();
      m_Car.Move(s.H, s.V, s.V, 0f);

    }
  }
}
```

CommandServer.cs

```
using UnityEngine;
using System.Collections.Generic;
using System.Collections;
using SocketIO;
using UnityStandardAssets.Vehicles.Car;
using System;
using System.Security.AccessControl;

public class CommandServer : MonoBehaviour
{
        public CarRemoteControl CarRemoteControl;
        public Camera FrontFacingCamera;
        private SocketIOComponent _socket;
        private CarController _carController;

        void Start()
        {
                _socket = GameObject.Find("SocketIO").GetComponent<SocketIOComponent>();
                _socket.On("open", OnOpen);
                _socket.On("steer", OnSteer);
                _socket.On("manual", onManual);
                _carController = CarRemoteControl.GetComponent<CarController>();
        }

        void OnOpen(SocketIOEvent obj)
        {
                Debug.Log("Connection Open");
                EmitTelemetry(obj);
        }

        void onManual(SocketIOEvent obj)
        {
                EmitTelemetry (obj);
```

```
        }

        void OnSteer(SocketIOEvent obj)
        {
                JSONObject jsonObject = obj.data;
                CarRemoteControl.SteeringAngle =
float.Parse(jsonObject.GetField("steering_angle").str);
                CarRemoteControl.Acceleration = float.Parse(jsonObject.GetField("throttle").str);
                EmitTelemetry(obj);
        }

        void EmitTelemetry(SocketIOEvent obj)
        {
                UnityMainThreadDispatcher.Instance().Enqueue(() =>
                {
                        print("Attempting to Send...");
                        if ((Input.GetKey(KeyCode.W)) || (Input.GetKey(KeyCode.S))) {
                                _socket.Emit("telemetry", new JSONObject());
                        }
                        else {
                                Dictionary<string, string> data = new Dictionary<string, string>();
                                data["steering_angle"] =
_carController.CurrentSteerAngle.ToString("N4");
                                data["throttle"] = _carController.AccelInput.ToString("N4");
                                data["speed"] = _carController.CurrentSpeed.ToString("N4");
                                data["image"] =
Convert.ToBase64String(CameraHelper.CaptureFrame(FrontFacingCamera));
                                _socket.Emit("telemetry", new JSONObject(data));
                        }
                });
        }
}
```

MenuOptions.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityEngine.SceneManagement;

public class MenuOptions : MonoBehaviour
{
   private int track = 0;
   private Outline[] outlines;

   public void Start ()
   {
      outlines = GetComponentsInChildren<Outline>();
   }

        public void MainMenu()
        {
```

```
                SceneManager.LoadScene("MenuScene");
        }

    public void StartDrivingMode()
    {
        switch (track)
        {
            case 0:
                SceneManager.LoadScene("HillTrackTraining");
                break;

            case 1:
                SceneManager.LoadScene("SnakeTrackTraining");
                break;

        }
    }

    public void StartAutonomousMode()
    {
        switch (track)
        {
            case 0:
                SceneManager.LoadScene("HillTrackAuto");
                break;

            case 1:
                SceneManager.LoadScene("SnakeTrackAuto");
                break;
        }
    }

    public void SetHillTrack()
    {
        outlines[0].enabled = true;
        outlines[1].enabled = false;
        track = 0;
    }

    public void SetSnakeTrack()
    {
        outlines[0].enabled = false;
        outlines[1].enabled = true;
        track = 1;
    }
}
```

MonoSingelton.cs

```
using UnityEngine;

public abstract class MonoSingleton<T> : MonoBehaviour where T : MonoSingleton<T>
{
```

```
    private static T m_Instance = null;
    public static T instance
    {
        get
        {
            if (m_Instance == null)
            {
                m_Instance = GameObject.FindObjectOfType(typeof(T)) as T;

                if (m_Instance == null)
                {
                    Debug.LogWarning("No instance of " + typeof(T).ToString() + ", a temporary one is created.");
                    m_Instance = new GameObject("Temp Instance of " + typeof(T).ToString(),
typeof(T)).GetComponent<T>();

                    if (m_Instance == null)
                    {
                        Debug.LogError("Problem during the creation of " + typeof(T).ToString());
                    }
                }
                m_Instance.Init();
            }
            return m_Instance;
        }
    }
    private void Awake()
    {
        if (m_Instance == null)
        {
            m_Instance = this as T;
            m_Instance.Init();
        }
    }

    public virtual void Init() { }

    private void OnApplicationQuit()
    {
        m_Instance = null;
    }
}
```

SplashTimer.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;

public class SplashTimer : MonoBehaviour {

        void Start () {
                StartCoroutine (LoadMenuScene());
        }
```

```
        IEnumerator LoadMenuScene(){
                yield return new WaitForSeconds (1.5f);
                SceneManager.LoadScene ("MenuScene");
        }
}
```

UISystem.cs

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using UnityStandardAssets.Vehicles.Car;
using UnityEngine.SceneManagement;

public class UISystem : MonoSingleton<UISystem> {

   public CarController carController;
   public string GoodCarStatusMessage;
   public string BadSCartatusMessage;
   public Text MPH_Text;
   public Text Angle_Text;
   public Text RecordStatus_Text;
        public Text DriveStatus_Text;
        public Text SaveStatus_Text;
   public Text Stopwatch_Text;
   public GameObject RecordingPause;
        public GameObject RecordDisabled;
        public bool isTraining = false;

   private bool recording;
   private float topSpeed;
        private bool saveRecording;
   private float timeElapsed = 0f;


   void Start()
   {
      topSpeed = carController.MaxSpeed;
      recording = false;
      RecordingPause.SetActive(false);
                RecordStatus_Text.text = "RECORD";
                DriveStatus_Text.text = "";
                SaveStatus_Text.text = "";

                SetAngleValue(0);
      SetMPHValue(0);

                if (!isTraining)
      {
                        DriveStatus_Text.text = "Auto controll";
                        RecordDisabled.SetActive (true);
                        RecordStatus_Text.text = "";
```

```
		}
}

public void SetAngleValue(float value)
{
   Angle_Text.text = value.ToString("N2") + "°";
}

public void SetMPHValue(float value)
{
   MPH_Text.text = value.ToString("N2");
}

public void SetStopwatchValue(float value)
{
   float minutes = Mathf.FloorToInt(value / 60);
   float seconds = Mathf.FloorToInt(value % 60);
   Stopwatch_Text.text = string.Format("{0:00}:{1:00}", minutes, seconds);

}

public void ToggleRecording()
{
		if (!isTraining)
   {
			return;
		}

   if (!recording)
   {
			if (carController.checkSaveLocation())
			{
				recording = true;
				RecordingPause.SetActive(true);
				RecordStatus_Text.text = "RECORDING";
				carController.IsRecording = true;
			}
   }
   else
   {
			saveRecording = true;
			carController.IsRecording = false;
   }
}

void UpdateCarValues()
{
   SetMPHValue(carController.CurrentSpeed);
   SetAngleValue(carController.CurrentSteerAngle);
   SetStopwatchValue(timeElapsed);
}

	void Update ()
```

```
  {
    if (carController.getSaveStatus())
    {
                        SaveStatus_Text.text = "Data collecting: " + (int)(100 *
carController.getSavePercent ()) + "%";
                }
                else if(saveRecording)
                {
                        SaveStatus_Text.text = "";
                        recording = false;
                        RecordingPause.SetActive(false);
                        RecordStatus_Text.text = "RECORD";
                        saveRecording = false;
                }

    if (Input.GetKeyDown(KeyCode.R))
    {
      ToggleRecording();
    }

                if (!isTraining)
                {
                        if ((Input.GetKey(KeyCode.W)) || (Input.GetKey(KeyCode.S)))
                        {
                                DriveStatus_Text.color = Color.red;
                                DriveStatus_Text.text = "Manual controll";
                        }
                        else
                        {
                                DriveStatus_Text.color = Color.white;
                                DriveStatus_Text.text = "Auto controll";
                        }
                }

            if(Input.GetKeyDown(KeyCode.Escape))
    {
      SceneManager.LoadScene("MenuScene");
    }

    if (!carController.isCollided)
    {
      timeElapsed += Time.deltaTime;
    }

    UpdateCarValues();
  }
}
```

Додаток В.2

Програмний код тренування нейронної мережі

model_train.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import os
import pandas as pd
from tensorflow import keras
from utils import dataset
from model import make_model
import argparse

parser = argparse.ArgumentParser(description = 'Train the model for self driving car')
parser.add_argument('--train_csv_file',metavar='string',type=str,required = True,help='Path to train
driving log csv file')
parser.add_argument('--test_csv_file',metavar='string',type=str,required = True,help='Path to test
driving log csv file')
parser.add_argument('--batch_size',metavar='int',type=int,default=32,help='batchsize for training,
default: 32')
parser.add_argument('--epochs',metavar='int',type=int,default=10,help='epochs for training, default:
50')

args = parser.parse_args()

batchsize= args.batch_size
epochs = args.epochs

train_csv_file_path=args.train_csv_file
test_csv_file_path=args.test_csv_file

train_ds=dataset(train_csv_file_path,batchsize)
if test_csv_file_path is not None:
  test_ds=dataset(test_csv_file_path,batchsize)
else:
   test_ds=None

img_shape= train_ds.get_img_shape()

model=make_model(inshape=img_shape)
model.summary()

loss="mean_squared_error"
#lrs=[0.001,0.0001,0.00001,0.01]            ----------------------------
lrs=[0.0001]
#lrs=[0.01, 0.001, 0.0001]

for lr in lrs:
```

```
    print("learning rate: "+str(lr))
    print("loss : "+loss)
    callbacks = [

keras.callbacks.ModelCheckpoint(os.path.join('models',str(lr),str(loss),'save_at_{epoch}.h5'),save_best_
only= True),
        keras.callbacks.EarlyStopping(monitor='loss',patience=5,mode='min'),
    ]
    model.compile(
        optimizer=keras.optimizers.Adam(lr),
        loss=loss,
        metrics=loss,
    )

    result=model.fit(
        train_ds, validation_data=test_ds, epochs=epochs, callbacks=callbacks,
    )
```

model.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from tensorflow import keras
from tensorflow.keras import layers

def make_model(inshape):
  inputs = keras.Input(shape=inshape+(3,))
  x = layers.Rescaling(1.0 / 255.0)(inputs)
  x = layers.Conv2D(32, 5, strides=2, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(48, 5, strides=2, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(64, 5, strides=2, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(64, 5, strides=2, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(76, 3, strides=1, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(76, 3, strides=1, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Conv2D(76, 3, strides=1, padding="same")(x)
  x = layers.Activation("relu")(x)
  x = layers.Dropout(0.5)(x)
  x = layers.Flatten()(x)
  x= layers.Dense(100,activation="relu")(x)
  x= layers.Dense(50,activation="relu")(x)
  x= layers.Dense(20,activation="relu")(x)
  x= layers.Dense(10,activation="relu")(x)
  outputs= layers.Dense(1)(x)

  return keras.Model(inputs,outputs)
```

utils.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import pandas as pd
from tensorflow import keras
import numpy as np
from tensorflow.keras.preprocessing.image import load_img


class dataset(keras.utils.Sequence) :

   def __init__(self,csvfile,batchsize=32,count=None,imgsize=None):

     self.batchsize=batchsize
     self.csvfile=csvfile
     self.data=pd.read_csv(self.csvfile)

     if count==None:
        self.count=self.data.shape[0]
     else:
        self.count=count

     if imgsize==None:
       size=load_img(self.data.iloc[0,0]).size
       self.img_size= (size[1],size[0])
     else:
        self.img_size=imgsize


   def __len__(self,):
     return  (self.count // self.batchsize)


   def get_img_shape(self):
      return self.img_size


   def __getitem__(self,idx):
     i=idx*self.batchsize
     batch_data=self.data.iloc[i:i+self.batchsize]
     x=np.zeros((self.batchsize,)+(self.img_size)+(3,),dtype='uint8')
     y=np.zeros((self.batchsize,)+(1,))
     for i in range(self.batchsize):
       imgc=load_img(batch_data.iloc[i,0],target_size=self.img_size)
       imgl=load_img(batch_data.iloc[i,1],target_size=self.img_size)
       imgr=load_img(batch_data.iloc[i,2],target_size=self.img_size)
       x[i]=np.hstack((imgl,imgc,imgr))
       x[i]=imgc
       y[i]=(batch_data.iloc[i,3])
     return x,y
```

Додаток В.3

Програмний код модуля автономного керування

drive.py

```
import argparse
import base64
from datetime import datetime
import os
import shutil

import numpy as np
import socketio
import eventlet
import eventlet.wsgi
from PIL import Image
from flask import Flask
from io import BytesIO

from keras.models import load_model
import h5py
from keras import __version__ as keras_version

sio = socketio.Server()
app = Flask(__name__)
model = None
prev_image_array = None

MAX_SPEED = 15
MIN_SPEED = 12
speed_limit = MAX_SPEED

class SimplePIController:
    def __init__(self, Kp, Ki):
        self.Kp = Kp
        self.Ki = Ki
        self.set_point = 0.
        self.error = 0.
        self.integral = 0.

    def set_desired(self, desired):
        self.set_point = desired

    def update(self, measurement):
        # proportional error
        self.error = self.set_point - measurement
        self.integral += self.error
```

```
        return self.Kp * self.error + self.Ki * self.integral


controller = SimplePIController(0.1, 0.002)
set_speed = 15
controller.set_desired(set_speed)


@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        steering_angle = data["steering_angle"]
        throttle = data["throttle"]
        speed = data["speed"]
        imgString = data["image"]

        image = Image.open(BytesIO(base64.b64decode(imgString)))
        image_array = np.asarray(image)
        steering_angle = float(model.predict(image_array[None, :, :, :], batch_size=1))

        #global speed_limit
        #if float(speed) > speed_limit:
        #    speed_limit = MIN_SPEED  # slow down
        #else:
        #    speed_limit = MAX_SPEED
        #throttle = 1.0 - steering_angle**2 - (float(speed)/speed_limit)**2
        throttle = controller.update(float(speed))

        print(steering_angle, throttle)
        send_control(steering_angle, throttle)

        # save frame
        if args.image_folder != '':
            timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:-3]
            image_filename = os.path.join(args.image_folder, timestamp)
            image.save('{}.jpg'.format(image_filename))
    else:
        sio.emit('manual', data={}, skip_sid=True)


@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)


def send_control(steering_angle, throttle):
    sio.emit(
        "steer",
        data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        },
```

```
        skip_sid=True)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
    parser.add_argument(
        'model',
        type=str,
        help='Path to model h5 file. Model should be on the same path.'
    )
    parser.add_argument(
        'image_folder',
        type=str,
        nargs='?',
        default='',
        help='Path to image folder. This is where the images from the run will be saved.'
    )
    args = parser.parse_args()

    f = h5py.File(args.model, mode='r')
    model_version = f.attrs.get('keras_version')
    keras_version = str(keras_version).encode('utf8')

    if model_version != keras_version:
        print('You are using Keras version ', keras_version,
              ', but the model was built using ', model_version)

    model = load_model(args.model)

    if args.image_folder != '':
        print("Creating image folder at {}".format(args.image_folder))
        if not os.path.exists(args.image_folder):
            os.makedirs(args.image_folder)
        else:
            shutil.rmtree(args.image_folder)
            os.makedirs(args.image_folder)
        print("RECORDING THIS RUN ...")
    else:
        print("NOT RECORDING THIS RUN ...")

    app = socketio.Middleware(sio, app)

    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```