

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

«До захисту допущено»

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Програмне забезпечення
інтелектуальних та робототехнічних систем»**

спеціальність 121 «Інженерія програмного забезпечення»

**на тему: «Модель системи побудови маршруту для безпілотних
автомобілів»**

Виконав:

студент IV курсу, групи IT-62

Сенишин Богдан Васильович _____

Керівник:

асистент

Базака Ю. А. _____

Консультант з норм. контролю:

доцент, к.т.н.

Пасько В. П. _____

Рецензент:

професор кафедри ММСА, д.т.н.

Мухін В. Є. _____

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інтелектуальних та робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«__» _____ 2020 р.

ЗАВДАННЯ

на дипломний проєкт студента

Сенишина Богдана Васильовича

1. Тема проєкту «Модель системи побудови маршруту для безпілотних автомобілів», керівник проєкту Базака Юрій Анатолійович, асистент, затверджені наказом по університету від «07» травня 2020 р. № 1081
2. Термін подання студентом проєкту: 27.05.2020 р.
3. Вихідні дані до проєкту: модель планування маршруту для безпілотних автомобілів у симуляційному середовищі.
4. Зміст пояснювальної записки:
 - Вступ
 - 1. Опис предметної області
 - 2. Аналіз проблем при реалізації системи
 - 3. Розробка архітектури системи планування
- Висновки
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): Деталізація ієрархічної моделі планувальника: Етап №1 (А3); Деталізація ієрархічної моделі планувальника: Етап №2 (А3); Деталізація ієрархічної моделі планувальника: Етап №3 (А3).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Норм. контроль	доцент Пасько В. П.		
Перевірка на співпадіння	доцент Лісовиченко О. І.		

7. Дата видачі завдання « 01 » жовтня 2019р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Ознайомлення з завданням	04.10.19 – 12.11.19	
2	Аналіз предметної області	16.11.19 – 01.12.19	
3	Аналіз існуючих рішень	02.12.19 – 24.12.19	
4	Проектування моделі	19.01.20 – 25.02.20	
5	Розробка моделі	28.02.20 – 13.04.20	
6	Тестування моделі	15.04.20 – 18.04.20	
7	Оформлення документації	21.04.20 – 26.05.20	
8	Норм. контроль.	27.05.20 – 03.06.20	
9	Перевірка на співпадіння.	03.06.20 – 08.06.20	
10	Попередній захист.	27.05.20 – 09.06.20	
11	Захист		

Студент

Богдан СЕНИШИН

Керівник

Юрій БАЗАКА

АНОТАЦІЯ

У роботі розглянуто сучасну проблему – розробку моделі побудови та планування маршруту безпілотного автомобіля. В ході виконання проекту показано основні особливості існуючих рішень проблеми планування руху, визначено основні методи та етапи для розробки, показано, які вони мають переваги та недоліки.

Розроблена модель працює у симуляційному середовищі, до якої надана можливість інтеграції відповідних програмних частин для кожного із етапів проблеми, побудованої за принципами трьохрівневої архітектури.

Результат дипломного проекту має експериментальне застосування для осіб, що мають на меті точково розібратись в проблемі планування руху безпілотного автомобіля.

Ключові слова: безпілотний автомобіль, проблема планування, розробка у симуляційному середовищі, перенесення математичних методів на програмний код.

Розмір пояснювальної записки – 72 аркуші, містить 23 ілюстрації, 2 таблиці, 5 додатків.

ABSTRACT

The project examines a sufficient issue of developing model of system for building and planning route lanes for self-driving cars. During the project implementation, the main features of existing systems for motion planning problem were shown, key development methods and parts were defined and their advantages and disadvantages were revealed.

Developed model works in a simulation, which has the ability to integrate corresponding programming parts for every phase of the problem that are built on the principles of n-tier architecture.

The result of the project provides experimental usage for the people that are willing to understand motion planning problem for self-driving cars in more detail.

Keywords: self-driving car, planning problem, development in simulation environment, implementing mathematical methods with programming code.

Explanatory note size – 72 pages, contains 23 illustrations, 2 tables, 5 applications.

**Пояснювальна записка
до дипломного проєкту
на тему: «Модель системи побудови маршруту для
безпілотних автомобілів»**

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ВАЖЛИВИХ СКОРОЧЕНЬ І ТЕРМІНІВ	9
ВСТУП	10
РОЗДІЛ 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Загальний огляд об'єкта дослідження	12
1.1.1 Проблема автономності	12
1.1.2 Основні концепції програмної частини	13
1.2 Огляд існуючих рішень	16
1.3 Постановка задачі	18
1.4 Загальні відомості про прототип	21
ВИСНОВКИ ДО РОЗДІЛУ	23
РОЗДІЛ 2 АНАЛІЗ ПРОБЛЕМ ПРИ РЕАЛІЗАЦІЇ СИСТЕМИ	24
2.1 Аналіз проблем прототипу та їх рішень	24
2.1.1 Планування місії	28
2.1.2 Вираховування часу до зіткнення	30
2.1.3 Планування поведінки	31
2.1.4 Планування локалізації	34
2.2 Обґрунтування оптимальності технічних та теоретичних методів	36
2.2.1 Алгоритм A* для планування місії	37
2.2.2 Методи вираховування часу до зіткнення	40
2.2.3 Скінченний автомат для планування поведінки	45
2.2.4 Оптимізація проблеми планування локалізації	47
2.3 Конкретизація постановки задачі	50
ВИСНОВКИ ДО РОЗДІЛУ	51
РОЗДІЛ 3 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ПЛАНУВАННЯ	52
3.1 Тип додатку	52
3.2 Архітектура проекту	53

					ІТ-25.18.1081.01 ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дат.	Модель системи побудови маршруту для безпілотних автомобілів Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Сенишин Б.В.						
Перевір.		Базака Ю.А.					7	72
						КПІ ім. Ігоря Сікорського Каф. ТК Гр. ІТ-62		
Н. Контр.		Пасько В.П.						
Затверд.		Пархомей І.Р.						

3.3 Використані технології	55
3.3.1 Python	55
3.3.2 NumPy та SciPy	58
3.3.3 MySQL	60
3.4 Розробка та реалізація моделі планування руху	62
3.4.1 Опис апаратної бази та середовища розробки	62
3.4.2 Опис структури програмного продукту	65
3.4.3 Опис інструкції користувача	69
ВИСНОВКИ ДО РОЗДІЛУ	69
ВИСНОВКИ	71
ПЕРЕЛІК ПОСИЛАНЬ	72
ДОДАТКИ	73

ПЕРЕЛІК ВАЖЛИВИХ СКОРОЧЕНЬ І ТЕРМІНІВ

SAE – Society of Automotive Engineers

HD – High Definition

API – Application Programming Interface

СУБД – Система управління базами даних

SQL – Structured Query Language

DAO – Data Access Object

IDE – Integrated Development Environment

GUI – Graphical User Interface

FPS – Frames per second

CLI – Command Line Interface

					ІТ-62.25.1081.01 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дат.		

ВСТУП

В теперішній час все більшого значення набуває питання автономності середовища, у якому знаходиться кожен із нас та як зменшити людський вплив на повсякденні речі та проблеми. Це охоплює широкий спектр задач: медицина, військова індустрія, розумні дома, пошукові та рятувальні служби.

Значуще місце серед досліджень та виробництв займають безпілотні автомобілі і зараз їм приділяють вагому увагу як ніколи раніше. Сегмент таких засобів пересування розвивається дуже швидко, хоч сама ідея зародилась ще у XX столітті. Так, у 1939 році на виставці General Motors Норман Бель Геддес створив перший автономний автомобіль, який представляв собою електромобіль, керований радіокерованими електромагнітними полями, генерованими намагніченими металевими шипами, які були вбудовані в проїжджу частину.

Зараз багато транспортних засобів на дорозі вважаються напівавтономними завдяки таким характеристикам безпеки, як допоміжна система паркування та гальмування, а саме безпілотні автомобілі мають можливість самостійно керувати, гальмувати та припарковуватися. При тому багато з них оснащені не тільки GPS-системою, а також вдосконалені системою зондування, які можуть визначати межі смуги, знаки, сигнали та несподівані перешкоди.

Науковці та розробники сподіваються на те, що автономні транспортні засоби принесуть із собою кілька різних переваг, але найважливішим із них є, мабуть, підвищення безпеки на дорогах. Згідно звіту патрульної поліції України, у 2019 році зафіксовано понад 160 000 аварій, у яких загинуло близько 3500 людей і ще понад 32 000 людей були травмовані. Кількість ДТП, спричинених неправильним керуванням, ймовірно, значно зменшиться завдяки безпілотникам, оскільки автомобілі не мають людського фактору. Виймаючи

					ІТ-62.25.1081.01 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат.		10

людей з рівняння, очікується, що такі транспортні засоби зроблять дороги набагато безпечнішими для всіх.

Метою написання даної роботи є створення моделі будування маршрутних ліній і планування руху безпілотних автомобілів, яка допомагає полегшити складність проблем з якими можуть стикнутись дані види транспортних засобів. Дана робота не описує такі проблеми, як сприйняття автомобілем середовища (наприклад, розпізнавання об'єктів на дорозі), а саме те як машина планує свій рух в різних ситуаціях і як цей автомобіль контролюється.

Головним моментом є те, що дана методологія дозволить показати проблему планування маршруту безпілотного автомобіля в більших деталях, при тому буде надана можливість додавати нові методи та оброблювати різні ситуації в залежності від того, з чим може стикнутись транспортний засіб на дорозі.

					ІТ-62.25.1081.01 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дат.		

РОЗДІЛ 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд об'єкта дослідження

1.1.1 Проблема автономності

Автономність безпілотних автомобілів визначається 6-ма рівнями встановленими класифікаційною системою розробленою SAE International у 2014 році.

Кожен рівень класифікується за тим, наскільки часто водій має втручатись в керування машиною і наскільки уважним він повинен бути за кермом автономного транспортного засобу.

Нижче описано кожен із рівнів:

- 0 рівень – автомобілі взагалі не мають автономного або самостійного керування, рух проходить з усіма аспектами втручання водія, включаючи реагування на небезпеки;
- 1 рівень – автономія першого рівня є найпоширенішою на сьогоднішній день. Це стосується автомобілів, які мають системи, що дозволяють автомобілю та водію ділити управління автомобілем. Адаптивний круїз-контроль, який контролює швидкість і відстань до іншого транспортного засобу попереду, є хорошим прикладом, оскільки водієві все ж таки потрібно подбати про управління. Функція допомоги в паркінгу - також приклад 1 рівня, оскільки водій повинен контролювати швидкість транспортного засобу, а автомобіль піклується про рульове управління. У більшості виробників є цілий ряд засобів безпеки, які пропонують певну автономію рівня 1, тому на даний момент цей рівень є найбільш широко імплементований;
- 2 рівень – автомобілі другого рівня мають внутрішні системи, які піклуються про більшість аспектів водіння: рульове управління, прискорення та гальмування. Однак водій повинен мати можливість

					ІТ-62.25.1081.01 ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дат.		

втрутитися, якщо будь-яка частина системи виходить з ладу. Рівень 2 також називають «безручним». Однак водій зобов'язаний постійно тримати руки на кермі;

- 3 рівень – транспортні засоби третього рівня - це ті, які справді можна вважати автономними. Ті автомобілі, що потрапляють на рівень 3, можуть дозволити водієві сісти і робити ті речі, які зазвичай за кермом не займаються, оскільки автомобіль може подбати про рух на дорозі. Водіям дозволяється безпечно користуватися своїм телефоном або дивитися фільми, хоча вони все ще зобов'язані бути готовими, щоб втрутитися, якщо це необхідно;
- 4 рівень – саме автомобілі четвертого рівня називають "безпілотними", оскільки вони направлені на те, щоб якомога зменшити вплив людини під час руху на дорозі. Однак існують деякі обмеження, оскільки повноцінний режим самостійного керування може бути активований лише в певних районах, які вже відомі для транспортного засобу. Якщо автомобіль не перебуває у визначеній зоні або знаходиться в пробці, то він повинен бути в змозі забезпечити себе і водія безпекою, якщо другий не може взяти на себе управління в надзвичайних ситуаціях. Саме для такого рівня автономії буде розроблена модель планування руху у даній роботі;
- 5 рівень – останній рівень, який класифікується відсутністю будь-якої взаємодії водія у русі.

1.1.2 Основні концепції програмної частини

Програмна частина безпілотних автомобілів 4 рівня автономності складається з 3 основних ідей: сприйняття середовища і локалізація, планування та контроль. На зображенні (рис. 1.1) показано, де вони знаходяться в стандартній безпілотній системі:

					ІТ-62.25.1081.01 ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дат.		



Рис. 1.1. Типова автономна безпілотна система

Процес сприйняття в безпілотних автомобілях, використовує комбінацію високотехнологічних датчиків і камер у поєднанні з програмним забезпеченням для обробки та локалізації навколишнього середовища навколо автомобіля в режимі реального часу.

Сприйняття має вагомe значення для безпечного та надійного руху, оскільки дані, отримані від цих датчиків, збагачують основне прийняття рішень, щоб автомобіль прямував в правильному напрямку (до місця призначення) і не загрожує життю людей в машині і навколо неї.

Датчики, такі як радари (пристрій, який використовує підказки радіохвиль для картографування світу) та LIDAR (пристрій, який використовує лазери для відображення світу), у поєднанні з серією камер - це очі автомобіля.

Програмні системи, як і конволюційні нейронні мережі, діють як “мозок” транспортного засобу. Разом автомобіль може зібрати детальне розуміння того, що відбувається у світі.

Контроль руху – це процес перетворення намірів у дії; його головне призначення – виконати ті кроки, які були створені на етапі планування, шляхом надання необхідних входів на апаратний рівень, який генеруватиме бажані рухи. Розрахунки в контрольній системі відображають карту взаємодії в реальному світі з точки зору сил та енергії, в той час як когнітивна навігація та алгоритми планування в безпілотній системі зазвичай стосуються швидкості та положення транспортного засобу по відношенню до свого оточення.

Виміри всередині системи управління можуть також бути використані щоб визначити, наскільки добре система в принципі працює, і тому контролер може реагувати на відхилення порушень і змінювати динаміку системи на бажаний стан.

Планування стосується процесу прийняття цілеспрямованих рішень для досягнення цілей, як правило, щоб перевезти транспортний засіб зі стартового місця до цільового місця, уникаючи при цьому перешкоди та оптимізовувати можливий шлях, по якому буде рухатись машина.

На базовому рівні місія планування – це вийти з точки А до в В. Можна розглянути це з перспективи навігації, де нам потрібно перейти від однієї точки на карті до іншої точки, що відповідає до нашому кінцевому пункту призначення.

Найголовніше, це те, що такі речі, як зустрічні машини, перешкоди та інші агенти на дорозі, мають вирішальне значення для проблеми планування автономного руху.

					ІТ-62.25.1081.01 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дат.		

1.2 Огляд існуючих рішень

Проблема планування безпілотного руху широко вирішена в існуючих прототипів та моделей автомобілів, які випускаються у вільному виробництві. До таких можна віднести Nissan Leaf, Audi A8L, Tesla Model. Вартує виділити Google Waymo Project (рис. 1.2), що зараз на даний момент вважається найперспективнішим проектом і вже зараз має 4 рівень автономності.



Рис. 1.2. Google Waymo Project

Проте, це вже існуючі проекти, які вже широко продаються і якими користуються і те як вони вирішують питання планування – залишається в секреті.

Хорошим прикладом того як ця задача вирішується у симуляції – це результати курсу від Udacity Self-Driving Car Engineer Nanodegree Program. Проблема вирішується за допомогою декількох кроків:

- передбачення – включало в собі прогнозування поведінки інших транспортних засобів та оцінку їх розташування за часом у майбутньому;

					ІТ-62.25.1081.01 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дат.		

- генерація траєкторії – під час цього кроку заздалегідь встановлювались максимальні значення прискорення та максимальної швидкості. Обчислювались взаємодії з іншими автомобілями, такі як найближчий підхід та найменший час стикання. Крім того, додавались інші штрафні санкції, наприклад, за недотримання цільової швидкості;
- отримання даних з сенсорів – різні датчики (LIDAR, радари, камери тощо) безперервно надавали інформацію про навколишнє середовище, щоб відповісти на ключові питання: які об'єкти знаходяться поблизу, наскільки далеко вони, вони статичні чи динамічні, як швидко і в якому напрямку вони подорожують та інші;
- jerk minimization або ж мінімізація ривків – використовуючи координати Френе, цей метод міг “злагодити” траєкторію руху автомобіля описуючи її функцією 5-степені.

Результат цього рішення можна поглянути на зображенні (рис. 1.3):



Рис. 1.3. Результати роботи в курсі Udacity Self-Driving Car Engineer Nanodegree Program

1.3 Постановка задачі

Рішення проблеми планування, які описані вище мають один великий недолік: кожен із них вирішує задачу комплексно, зачіпаючи інші етапи руху безпілотного автомобіля. Наприклад, без відповідних знань середовища і локалізації автомобіля неможливо реалізувати метод *jerk minimization*.

Наша задача все-таки полягає в тому, щоб детально поглибитись саме в задачу планування, припускаючи, що середовище для нас вже відоме. Для цього потрібно провести аналіз різних ситуацій, які можуть виникнути на дорозі та розробити модель, яка на основі вже отриманих даних на етапі сприйняття середовища та локалізації автомобілі, зможе знайти оптимальний шлях, найбезпечніший маневр та визначити найбільш можливу траєкторію руху безпілотного автомобіля.

Щоб у повному вигляді описати те з чим може тикнутись безпілотний автомобіль під час свого руху, потрібно описати ряд сценаріїв на дорожній структурі. Під дорожньою структурою варто розуміти вартує розуміти межі дороги та регуляторні елементи, які стосуються водія.

Найпростіший із сценаріїв – це коли автомобіль просто їде по дорозі і дотримуються меж, які встановлені. У номінальному випадку це коли автомобіль слідує за центральною лінією поточної смуги. У цьому випадку наша мета полягає в тому, щоб мінімізувати наше відхилення від центральної лінії шляху, а також досягти нашої опорної швидкості, яка часто є обмеженням швидкості, для забезпечення ефективної подорожі до місця призначення.

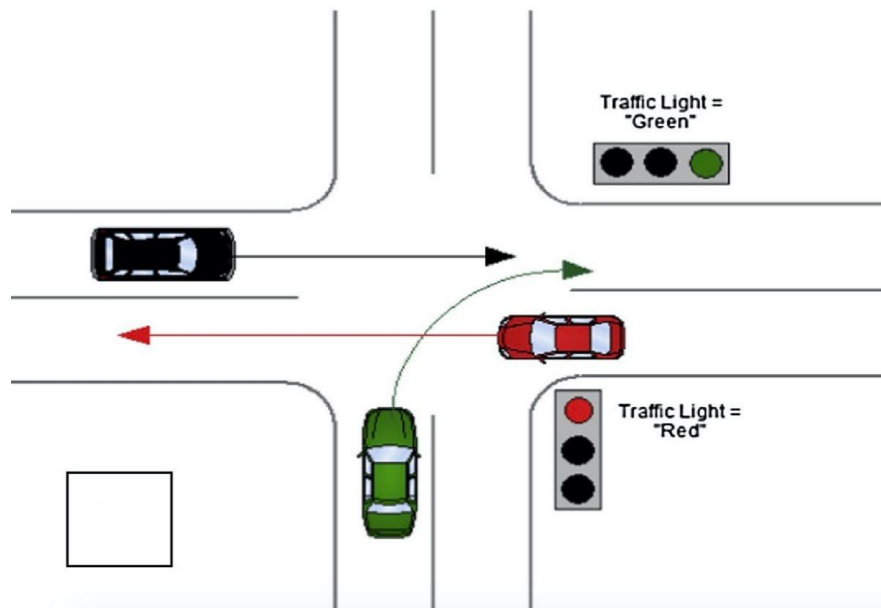


Рис. 1.4. Сценарій виконання повороту на перехресті

Третій сценарій – це коли автомобіль повинен виконувати поворот ліворуч або праворуч. Це часто виникає, коли автономна машина має працювати з перехрестями. Як і при зміні смуги руху, форма та агресивність повороту змінюватимуться залежно від ситуації. Крім того, на можливість здійснення дій впливає стан навколишнього середовища.

Наприклад, автономний транспортний засіб не може виконувати поворот ліворуч на червоне світло, навіть якщо перехрестя пусте. Такий варіант показаний на зображенні (рис. 1.4). Як остаточний приклад, ми маємо U-поворот, який важливий для навігації в певних сценаріях, коли автомобіль повинен ефективно змінювати напрямок. U-поворот у великій мірі також сильно залежить від стану навколишнього середовища, оскільки не завжди законно здійснювати розворот на перехресті залежно від законів певної країни.

Доріжна структура – не єдиний фактор, який може вплинути на сценаріях. Статичні та динамічні перешкоди різко змінюють як структуру сценарію, так і труднощі у визначенні необхідної поведінки для сценарію.

Динамічні перешкоди визначаються як рухомі агенти в робочому просторі планування безпілотного транспортного засобу. Статичні перешкоди – це перешкоди, які не рухаються, наприклад, припарковані автомобілі, вуличні повороти.

Статичні перешкоди обмежують, у яких місцях безпілотник може займати простір, коли він планує шлях до місця призначення. Якщо автомобіль перетинається з перешкодою, наш план, очевидно, призведе до зіткнення. З іншого боку, динамічні перешкоди мають більший вплив на наш профіль швидкості та поведінку водіння. Поширений приклад – коли перед нашим автомобілем їде ведучий транспортний засіб. Така ситуація показана на зображенні (рис. 1.5).

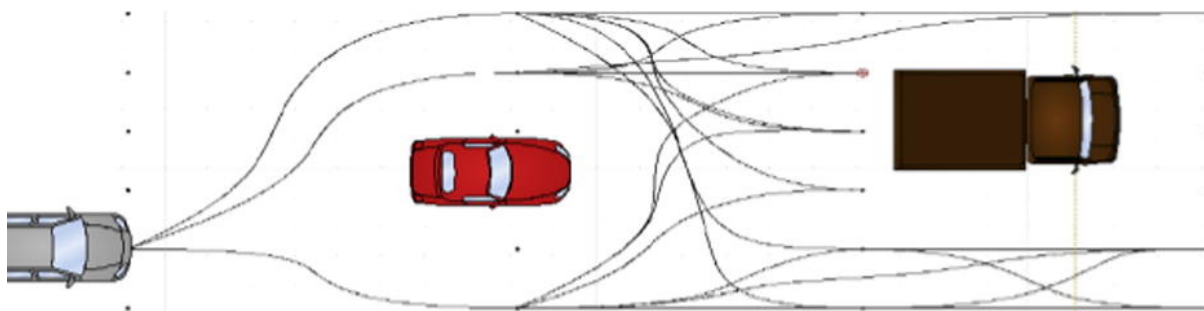


Рис. 1.5. Ситуація, коли перед автомобілем їде ведучий транспортний засіб

Очевидно, що цей транспортний засіб матиме вплив на наше прийняття рішень.

Динамічні перешкоди також вплинуть на наш сценарій зміни повороту та смуги руху. Залежно від їх місцезорозташування та швидкості, автомобіль має певний час, щоб виконати маневр. Цей час буде оціночний, оскільки він базується на прогнозах усіх інших агентів у навколишньому середовищі.

При тому, не всі динамічні перешкоди автомобілі, але є інші суб'єкти, такі як велосипедисти, вантажівки та пішоходи. Кожен з них також матиме власну поведінку та свої правила, яких слід дотримуватись у дорозі.

Тому, наша задача полягає в тому, щоб вибрати один із наведених вище сценаріїв, поєднати його з статичними та динамічними об'єктами та на основі нашої отриманої інформації розробити модель, яка зможе спланувати оптимальний шлях, найбезпечніший маневр та визначити найбільш можливу траєкторію руху безпілотного автомобіля.

1.4 Загальні відомості про прототип

Задача планування на перший погляд здається не настільки комплексною, як наприклад сприйняття або контроль.

Насправді, це не так. Дана задача має досить високий рівень складності і охоплює досить багато аспектів, з якими може стикнутись автономна машина. Для того, щоб її розглянути в повному обсязі, потрібно описати більшість моментів та ситуацій, які можуть виникнути на дорозі і для кожного із цих випадків визначити, які кроки будуть виконуватись на цьому етапі, наскільки комплексно це відобразиться на моделюванні і в принципі руху на дорозі.

Головним недоліком опису та моделювання даної задачі – це обмежений вибір середовища. Існуючі прототипи (тут мається на увазі автомобілі, які вже використовуються) охоплюють планування разом з іншими проблемами.

Що стосується комплексності задачі, потрібно описати кожен рівень ієрархії (рис. 1.6) який визначає власну проблему оптимізації, при цьому кожен етап має специфічну ступінь абстракції щодо відповідної цільової функції та обмежень

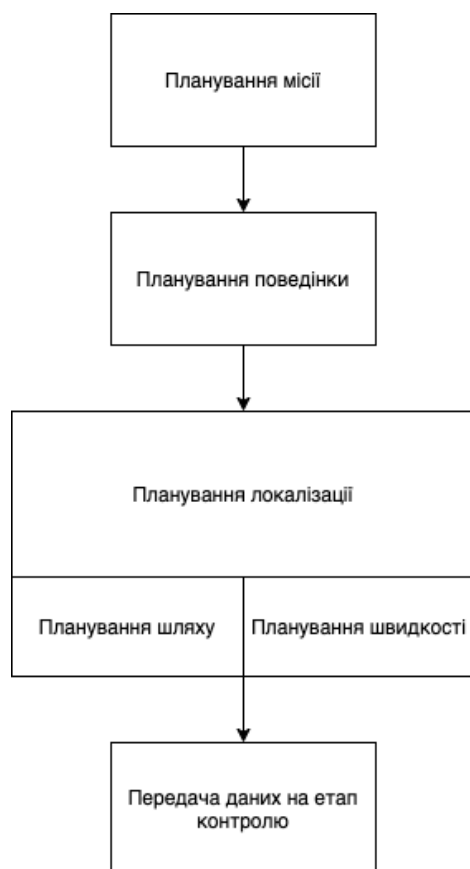


Рис. 1.6. Ієрархія проблеми планування

Планування місії – це проблема оптимізації найвищого рівня, де ми зосереджуємось на навігації на рівні карти від поточного положення транспортних засобів до заданого пункту призначення.

З іншого боку, планування поведінки буде зосереджено на поточному сценарії водіння, і вирішить, який маневр виконувати на основі інших агентів у робочій області та правил дорожнього руху.

В кінці, у нас є планування локалізації, яка зосередиться на генеруванні кінематично можливих шляхів без зіткнень, а також контроль швидкості та надання різних профілів для різних ситуацій.

Потім вся ця інформація надається нашому контролеру в безпілотному автомобілі. Детальніша інформація про ієрархію винесена у додатки Б, В, Г.

ВИСНОВКИ ДО РОЗДІЛУ

У першому розділі висвітлені головні аспекти автономності та програмної частини безпілотних автомобілів. Проведено аналіз предметної області та методів вирішення даної задачі. Розглянуті основні способи вирішення проблеми планування руху та їх застосування в існуючих рішеннях.

В результаті проведеного аналізу (табл. 1.1) сформульована мета задачі подальшої розробки та описані основні характеристики системи.

Таблиця 1.1. Порівняльна характеристика існуючих рішень та обраного прототипу

Порівняльна характеристика існуючих рішень та обраного прототипу		
	Існуючі рішення	Обраний прототип
Середовище розробки	Готова продукція	Симуляційне
Залежність від інших етапів процесу	Висока	Висока
Спосіб вирішення проблеми	Враховування всіх аспектів процесу руху безпілотного автомобіля	Декомпозиція проблеми на підзадачі
Обсяг деталізації проблеми	Середній, багато інформації отримується на інших етапах	Високий, задача розглядається точково, зачіпаючи всі аспекти планування
Оптимальність руху	Повна	Неповна, проте це покривається тим, що рух планується в реальному часі

РОЗДІЛ 2 АНАЛІЗ ПРОБЛЕМ ПРИ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Аналіз проблем прототипу та їх рішень

Головним елементом в нашій поставленій задачі постає симуляційне середовище, в якому буде змодельована проблема планування безпілотного автомобіля. Фактично кожне середовище можна представити у вигляді карти, де визначені основні елементи такі як динамічні та статичні об'єкти, межі дороги та ліній, місця перехресть та інші.

Щоб описати таке середовище у зрозумілому вигляді для автономної машини введене таке поняття як High-Definition Road Map, або ж дорожня мапа високої чіткості.

Така мапа схожа на традиційні паперові карти або онлайн-карти. Однак інформація, що міститься на карті високої чіткості або на HD карті, є значно більш детальною. У той час як традиційна карта зберігає приблизне розташування доріг, даний вид мап зберігає точні місця доріг, включаючи всі смуги руху до точності на кілька сантиметрів. Поряд із цим, дорожні карти високої чіткості зберігають усі місця дорожніх знаків та сигналів, які можуть впливати на автономний транспортний засіб.

HD-мапа складається з двох базових структур: смуговий та перехресний елементи. Перший зберігає всю інформацію, пов'язану з невеликим поздовжнім відрізком смуги на дорозі, яку він представляє. Другий зберігає всі елементи смуги, які є частиною єдиного перехрестя для простого пошуку під час завдань планування руху. Обидва елементи грають велику роль в задачі планування, оскільки саме вони є представленням того, яку інформацію отримує автомобіль для обробки, щоб виконати загальні обчислення та прописати кроки, які вона має виконати в тій чи іншій ситуації. Це показано на зображенні (рис. 2.1).

					ІТ-62.25.1081.01 ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дат.		

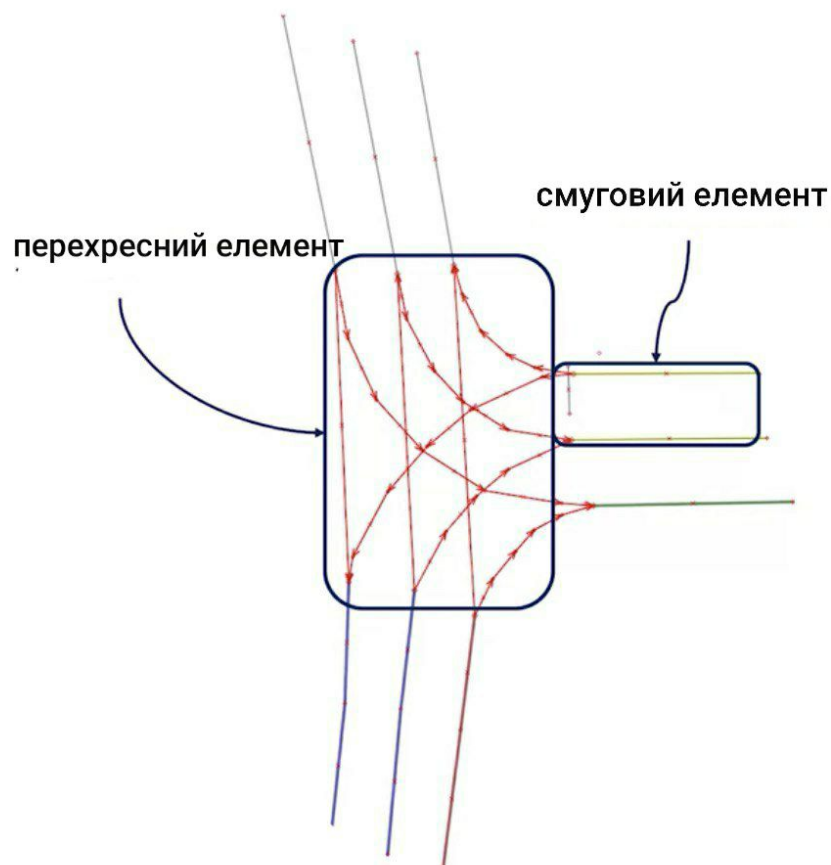


Рис. 2.1. Основні елементи HD-мапи

Смугові елементи мають кілька наборів інформації, яку вони зберігають, про той чи інший відрізок дороги:

- зберігаються ліва і права межі даної смуги;
- зберігається будь-які регулюючі елементи, які можуть бути присутніми, такі як лінія знаку зупинки або статична лінія знаків;
- зберігаються всі регуляторні атрибути, які можуть впливати на цей відрізок дороги, наприклад обмеження швидкості;
- зберігається зв'язок між собою з іншими елементами смуги.

Це дозволяє легко пройти та обчислити граф, створений набором смугових елементів на HD-мапі.

Кожна смуга закінчується регуляторним елементом або відбувається зміна регуляторного атрибута. Це означає, що смуговий елемент може бути коротким

лише на кілька метрів, або може бути довжиною сотні метрів, наприклад на шосе.

Межі даного елемента представлені у вигляді країв даного елемента і зберігаються у вигляді сукупності точок, що створюють суцільну багатокутну лінію. Відстань між точками може бути такою ж маленькою, як кілька сантиметрів, або, звичайно, декілька метрів залежно від гладкості смуги, про яку йдеться.

Використовуючи цю структуру смуг, для збору даних для планування руху можна виконати багато корисних операцій. Впорядкованість точок визначає напрямок руху та прямування смуги руху. Центральна лінія між двома межами може бути використана як бажаний шлях руху автономного транспортного засобу на цій смузі.

Існує два типи регламенту щодо смуги: елементи регулювання, які знаходяться в кінці, і атрибути регулювання, які впливають на повноту елемента.

Регулюючі елементи представлені у вигляді ліній, які визначаються набором колінійних точок. Елементи регулювання зазвичай вимагають прийняття дії або рішення, наприклад, у зупинитись на червоне світло світлофора, або рішення про продовження руху повинно базуватися на поточному стані цього світлофора.

Регулюючі атрибути зберігаються як ті ознаки, які зараз мають бути присутні на смузі, наприклад обмеження швидкості.

Кожен смуговий елемент має чотири можливих з'єднання: ліве, право, передуюче та заднє.

Вся структура смугових елементів з'єднана у спрямований граф, який є базовою структурою HD-мапи. Щоб краще зрозуміти дану концепцію, можна подивитись на простий приклад тристорннього перехрестя на зображенні (рис. 2.2):

					ІТ-62.25.1081.01 ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дат.		

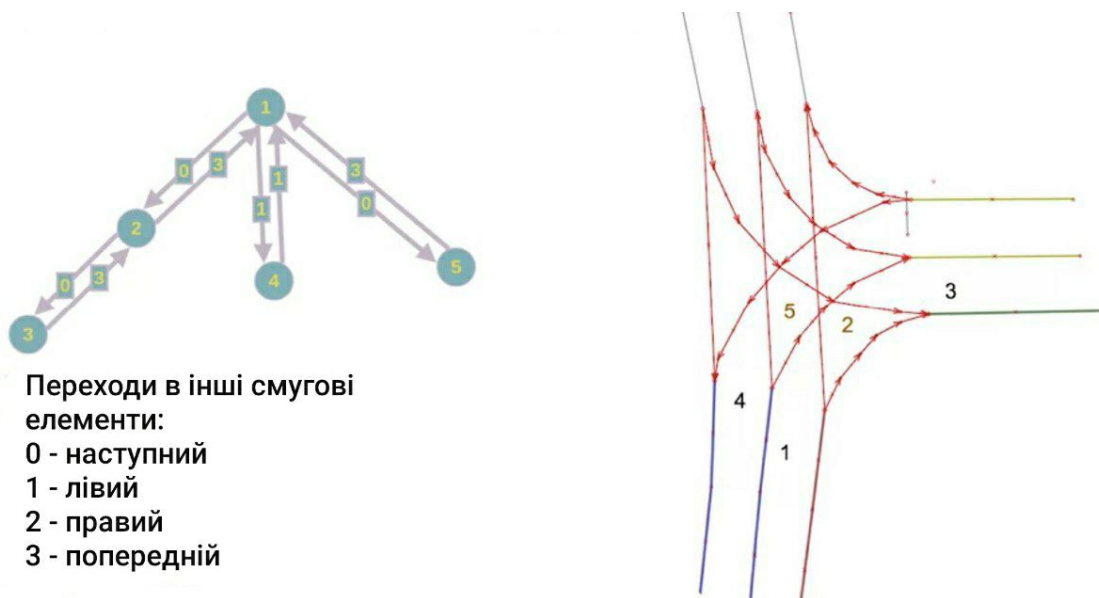


Рис. 2.2. Переходи між смуговими елементами у вигляді графа

Як можна побачити, деякі з смугових елементів пронумеровані та представлені у вигляді вузлів на графі, які, представляють лише частину повного графа з на цьому перехресті. Вузли позначаються від 1 до 5 та утворюють вершини графа смугового елемента. Усі краї цього графа спрямовані і мають мітку для позначення зв'язку між двома вершинами. Перехід, позначений як нулем, являє собою перехід до наступного елемента, одиниця вказує на з'єднання з лівим, двійка - правим, трійка - попереднім.

Наприклад, смуговий елемент, який зображений номером один на цьому малюнку, насправді має два можливих наступні елементи – п'ятий та другий, які мають переходи номер 3 та 0 відповідно. Аналогічно, четвертий елемент з'єднаний з першим переходом номер один, так як він зліва від нього.

Щодо елемента перетину, він містить вказівку на всі регуляторні елементи, які складають перетин. Усі смугові елементи, які є частиною перетину, також вказують на цей елемент перетину. Ця структура подібна до контейнера, яка вміщує в собі смугові елементи, та спрощує огляд по всій структурі дороги при призначенні поведінки безпілотного автомобіля.

Великою перевагою структури, яка складається з даних двох елементів є те, що вона робить частини процесу планування руху більш простими та обчислювально ефективними. Процес планування шляху через складні багатосмугові дорожні мережі, які потребують декількох змін смуг та інших кроків, стає можливим саме завдяки цій структурі, оскільки кожна смуга трактується як окрема вершина або ж одиниця. При тому, локалізація динамічних та статичних об'єктів на відомій карті покращує етап планування шляху, як ми побачимо далі. Цей тип можливості локалізації також покращує взаємодію з цими об'єктами, надаючи простий метод планування поведінки автономного автомобіля в напружених ситуаціях.

Отримавши необхідну інформацію про середовище, безпілотний автомобіль може переходити безпосередньо до планування. Для цього потрібно зрозуміти, що собою являє кожен із ієрархічних рівнів, які ми описали раніше та які методи застосовуються для оптимального руху транспорту.

2.1.1 Планування місії

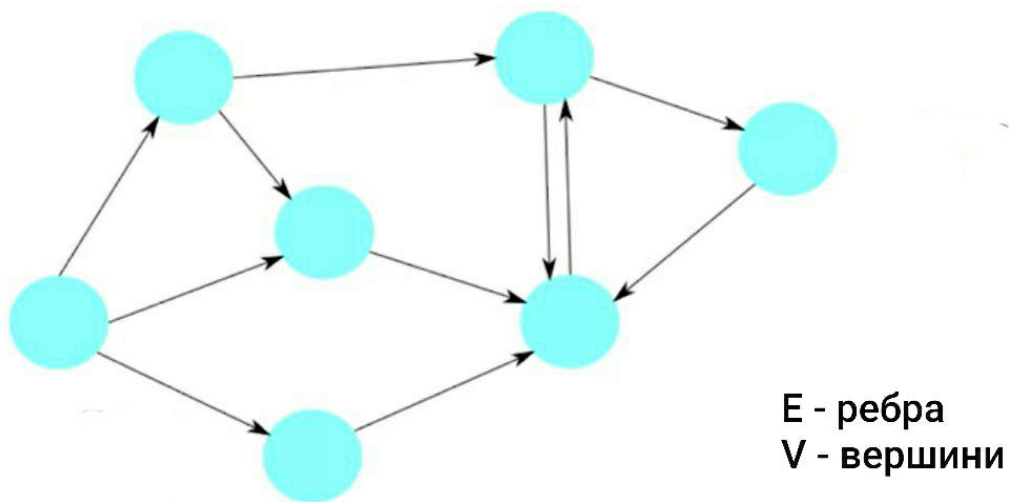
Завдання планування місії водіння полягає в пошуку оптимального шляху для транспортного засобу з його поточного положення до заданого пункту призначення шляхом навігації по дорожній мережі, а також вилучення деталей нижнього рівня, таких як правила дорожнього руху та інших агентів, присутніх в сценарії водіння.

Для автономного водіння планування місії вважається проблемою планування найвищого рівня. Це відбувається тому, що масштаб просторового планування обчислюється на рівні кілометрів і планувальник місії не фокусується на обмеженнях планування на низькому рівні, таких як перешкоди чи динаміка. Натомість планувальник місії при виконанні задачі зосередить увагу на аспектах дорожньої мережі, таких як обмеження швидкості та довжина дороги, швидкість руху. Виходячи з цих обмежень, поставлених перед нами на

карті, планувальник місій повинен знайти оптимальний шлях до потрібного місця призначення. Одне, що слід зазначити про дорожню мережу, – це те, що вона є високоструктурованою, що можна використати в процесі планування для спрощення проблеми. Використовуючи дану структуру, ми можемо ефективно знайти оптимальний шлях до нашого пункту призначення, ґрунтуючись на наданій нам карті.

Для цього нам потрібно буде використовувати вже відомий для нас граф в якості математичної структури, який ми накладаємо на нашу дорожню мережу. Базовий приклад графа можна побачити на зображенні (рис. 2.3).

Граф: $G = (V, E)$



**E - ребра
V - вершини**

Рис. 2.3. Загальний вигляд графа

У цьому сенсі послідовність суміжних ребер на графі відповідає шляху через дорожню мережу від однієї точки до іншої. В прикладі вважається, що кожен відрізок дороги має однакову довжину, тому краї цього графіка всі не зважені. Однак далі це обмеження буде забрано.

Щоб згенерувати такий граф, дорожню мережу потрібно дискретно поділити на вибірки, що дасть нам вершини нашого графа. Потім ребра

визначатимуться відрізками дороги, які з'єднують кожну точку вибірки відповідно до правил дорожнього руху. Потрібно зауважити що, якщо точка А з'єднана до точки В з використанням відрізка дороги, то це не означає, що точка А може бути досягнута з точки В на цьому відрізку дороги. Це тому, що в багатьох випадках існує лише один напрямок, через який можна легко пройти ділянку дороги.

2.1.2 Вираховування часу до зіткнення

Перед тим як перейти безпосередньо до планування поведінки безпілотного автомобіля, потрібно звернути увагу на те як транспорт взаємодіє з динамічними об'єктами і уникає зіткнення з ними.

Концепція часу до зіткнення забезпечує цінний показник безпеки поведінки у безпілотному автомобілі і вона широко використовується для оцінки потенційних маневрів, виходячи з поточних умов водіння. Знаючи, чи зіткнення є неминучими, і коли вони можуть статися, і з яким об'єктом, транспорт може краще спланувати безпечні маневри у середовищі, або підготуватися до екстрених уникнень, якщо потрібно. Щоб оцінити час зіткнення між динамічними об'єктами використовуються їх передбачувані шляхи для виявлення можливих точок зіткнення. Якщо існує точка зіткнення, час зіткнення - це міра часу до моменту зіткнення.

Отже, обчислення часу до зіткнення може бути виконано у два етапи:

- ідентифікація та обчислення розташування точки зіткнення вздовж прогнозованих шляхів між динамічними об'єктами;
- якщо така точка зіткнення існує, обчислюється кількість часу, яка знадобиться, щоб динамічні об'єкти дійшли до цієї точки зіткнення.

Завдання обчислення часу до зіткнення полягає в тому, що воно дуже сильно покладається на точне розуміння простору, який будуть займати об'єкти в майбутньому, для досягнення точних оцінок. Очевидно, що точні прогнози

траєкторій об'єктів мають вирішальне значення для виявлення точок зіткнення за часом їх виникнення. Точне прогнозування об'єктних траєкторій – це складна проблема. Крім того, нам також потрібні точні геометричні параметри для всіх динамічних об'єктів у навколишньому середовищі. Це потрібно, оскільки, коли обчислюється час зіткнення, для обчислення точної точки зіткнення враховуються розміри обох сутностей. Через помилки в обох вимогах час до зіткнення завжди слід сприймати як наближений час, який регулярно оновлюється та обробляється певним захисним буфером при прийнятті рішень безпілотним автомобілем.

2.1.3 Планування поведінки

Система планування поведінки – це концепція, яка включає в собі планування набору водійських дій або маневрів для безпечного досягнення місії безпілотного автомобіля в різних ситуаціях водіння. Набір запланованих маневрів повинен враховувати правила дорожнього руху та взаємодію з усіма статичними та динамічними об'єктами в навколишньому середовищі.

Набір рішень, що приймаються планувальником, повинен забезпечувати безпеку транспортного засобу та ефективний рух у навколишньому середовищі. Ці поняття були обговорені в попередніх пунктах, і саме планувальник поведінки повинен приймати правильні рішення, щоб безпечно рухатися до нашої мети.

У якості приклада, припустимо, що автономний транспортний засіб прибуває на зайнятому перехресті. Планувальник поведінки повинен врахувати:

- де зупинитись;
- коли зупинитися;
- коли пройти через перехрестя.

Планувальник поведінки повинен виконувати цей тип прийняття рішень в обчислювально-ефективній формі, щоб він міг швидко реагувати на зміни навколишнього середовища і бути розгорнутим на обладнанні безпілотного

автомобіля. Також мати він мусить мати справу з вхідними даними, які можуть бути неточними, зіпсованими шумом вимірювань, так і неправильними.

Список базових поведінок, з якими може стикнутись автономна машина:

- дотримання потрібної швидкості;
- слідування транспорту, що їде попереду – швидкість транспортного засобу перед безпілотним автомобілем повинна бути в нормі, і слід також дотримуватися безпечної відстані;
- сповільнитись перед зупинкою – транспортний засіб повинен уповільнитись до місця зупинки. Кожен регуляторний елемент, який вимагає повної зупинки, викликає цю поведінку;
- зупинка – транспортний засіб повинен залишатися нерухомим протягом певного часу. Наприклад, коли автомобіль зупиняється на знаку зупинки, він повинен зупинятися принаймні три секунди;
- перехід на сусідню смугу або продовження руху в теперішній смузі.

Цей основний перелік маневрів добре послужить у розробці принципів планування поведінки. Однак на практиці слід розглянути набагато більше випадків поведінки, і в результаті загальна складність планувальника зростатиме.

Основний результат планувальника поведінки – це маневр, який слід виконувати в поточному середовищі. Поряд з маневром руху, планувальник поведінки також створює набір обмежень, які потрібні для планування локалізації. Сюди входить:

- шлях від поточного розташування транспортного засобу до цільового пункту призначення (для багатьох видів поведінки – це центральна лінія поточної смуги транспортного засобу);
- обмеження швидкості по шляху;
- межі поточної смуги;

- будь-які місця зупинки в майбутньому, до яких потрібно приїхати транспортним засобом;
- набір динамічних об'єктів, що викликають високий інтерес і які мають бути враховані в плануванні локалізації. Ці динамічні об'єкти можуть бути важливими для майбутнього шляху.

Для того, щоб планувальник поведінки міг отримати необхідний результат, йому потрібна велика кількість інформації з багатьох інших програмних систем у стеку безпілотного автомобіля:

- планувальник поведінки покладається на повне знання дорожньої мережі біля транспортного засобу з HD-мапи;
- планувальник поведінки повинен знати, якими дорогами потрібно користуватись, щоб дістатися до цільового місця. Ці дані отримуються на етапі планування місії;
- важливим є розташування транспортного засобу, щоб мати можливість правильно позначити елементи HD-мапи у середовищі навколо транспортного засобу;
- нарешті, для планування поведінки потрібна вся відповідна інформація про сприйняття, щоб повністю зрозуміти дії, які потрібно вжити, щоб безпечно активувати рух. Ця інформація включає в собі всі спостережувані динамічні об'єкти в навколишньому середовищі, такі як автомобілі, пішоходи або велосипеди. Для кожного динамічного об'єкта необхідні його поточний стан, передбачувані шляхи, точки зіткнення та час зіткнення. Сюди також входять усі спостережувані статичні об'єкти, наприклад, припарковані транспортні засоби, будівельні конуси та світлофори з їхнім поточним станом.

Маючи всю наявну в ньому необхідну інформацію, планувальник поведінки повинен вибрати відповідний крок та визначити необхідні супутні

					ІТ-62.25.1081.01 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дат.		

обмеження, щоб забезпечити безпечний та ефективний рух транспортного засобу.

2.1.4 Планування локалізації

Даний етап – це частина ієрархічного планувальника, яка виконує маневр, який вимагає планувальник поведінки, без зіткнень, у вигляді ефективного та комфортного руху. Це призводить до або траєкторії, яка є послідовністю точок у просторі за даний час, або профілю шляху та швидкості, яка є послідовністю точок у просторі з необхідними швидкостями у кожній точці. Потім цей план може бути поданий як контрольний вхід для контролерів, які саме і виконують рух безпілотного автомобіля

Перший крок у розумінні проблеми планування локалізації полягає в тому, щоб спочатку зрозуміти його найбільш фундаментальні вимоги. Для даної задачі це задається початковою позицією, знаходження шляху до кінцевої позиції та кривизною, який задовольняє наші кінематичні обмеження. У контексті оптимізації вихідні та кінцеві значення можуть бути сформульовані як граничні умови задачі.

Граничні умови – це умови, які повинні дотримуватися у будь-якій точці шляху, щоб дане рішення оптимізації вважалось можливим. Якщо ці граничні умови будуть порушені, який би не був шлях – ми не досягли своєї основної мети.

Для спрощення подання задачі ми будемо визначати шлях як параметричну криву.

Параметрична крива – це крива, яку можна описати як набір рівнянь із певними параметрами. Ці параметри часто позначають проходження шляху, будь це довжина дуги або просто зміна параметра від нуля до одиниці із таким загальним описом (2.1):

$$r(u) = \langle x(u), y(u) \rangle, \quad u \in [0,1] \quad (2.1)$$

Для автономного руху часто, але не завжди, вимагається, щоб шлях був параметричною кривою. Така вимога виникає тому, що розробники часто зосереджуються на методах планування, які оптимізують заданий шлях відповідно до граничних умов, показаних на зображенні (рис. 2.4). як β_0 та β_f , кінематичні обмеження, показані як α , та об'єктивний функціонал, показаний тут як f . Параметричне подання шляху спрощує налаштування проблеми оптимізації, оскільки у нас є функція, яку ми можемо безпосередньо надати нашому об'єктивному функціоналу f . Зауважимо, що термін “функціональний” відноситься до відображень, які приймають функцію як аргумент і повертають реальне значення.

$$\min f(r(u)) \text{ s.t. } \begin{cases} c(r(u)) \leq \alpha, & \forall u \in [0,1] \\ r(0) = \beta_0 \\ r(u_f) = \beta_f \end{cases}$$

Рис. 2.4. Математичне представлення вирішення задачі оптимізації шляху

Для представлення нашого шляху ми будемо використовувати поліноміальні спіралі. Ці криві описуються рівнянням закритої форми для кривизни кривої вздовж кожної точки її довжини дуги. Для автономної їзди прийнято вибирати кубічний многочлен (2.2) як функцію кривизни довжини дуги:

$$k(s) = a_3 s^3 + a_2 s^2 + a_1 s + a_0 \quad (2.2)$$

Однак функції вищого порядку також прийнятні. Основна перевага використання поліноміальних спіралей полягає в тому, що їх структура дуже

сприятлива для задоволення приблизних обмежень кривизни, які часто потребує проблема локалізації. Оскільки спіраль є поліноміальною функцією кривизни, значення кривизни не буде змінюватися стрімко, як це може бути у випадку сплайнів. Це означає, що ми можемо обмежити викривлення лише декількох точок спіралі, і спіраль, швидше за все, задовольнить обмеження кривизни по всій кривій. Це дуже корисно при виконанні оптимізації шляху, оскільки кількість обмежень значно збільшує обчислювальні зусилля кожного кроку оптимізації.

Недоліком використання поліноміальних спіралей є те, що в закритому вигляді рішення знаходження кожної точки спіралі не існує, на відміну від випадку в сплайні. Тому ми повинні провести ітеративну оптимізацію, щоб згенерувати спіраль, яка задовольняє наші граничні умови. Рівняння знаходження x та y кожної з позиції призводять до інтегралів Френеля, які не мають рішення у своїй узагальненій формі (2.3 та 2.4):

$$x(s) = x_0 + \int_0^s \cos(\theta(s')) ds' \quad (2.3)$$

$$y(s) = y_0 + \int_0^s \sin(\theta(s')) ds' \quad (2.4)$$

2.2 Обґрунтування оптимальності технічних та теоретичних методів

Описавши основні вимоги того, що має бути присутнім в нашому планувальнику руху безпілотного автомобіля потрібно провести дослідження та обрати методи, які вирішують кожен із етапів даного процесу та показати їх оптимальність у порівняльному вигляді.

					ІТ-62.25.1081.01 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат.		36

2.2.1 Алгоритм A* для планування місії

Для того, щоб зрозуміти алгоритм A* потрібно познайомитись з евристичною функцією.

У нашому контексті евристика пошуку – це оцінка залишкової вартості для досягнення вершини призначення з будь-якої заданої вершини в графі. Звичайно, будь-яка евристика, яка використовується, не буде точною, хоч це і є вимогою планування місії. Натомість, ми покладаємось на структуру проблеми, щоб розробити розумну оцінку, яку можна швидко обчислити. У нашому випадку вершини на графі відповідають точкам у просторі, ребра відповідають за сегменти на дорогах, які мають вагу, що відповідає довжині цих відрізків доріг. Тому корисною оцінкою вартості або довжини між будь-якими двома вершинами є пряма або евклідова відстань між ними (2.5):

$$h(v) = ||t - v|| \quad (2.5)$$

Тому для будь-якої вершини, з якою ми стикаємось в процесі пошуку, наша оцінка залишкової вартості до цілі вершини буде просто евклідовою відстанню між цією вершиною та ціллю. Потрібно зауважити, що ця евклідова функція завжди є недооцінкою справжньої відстані до досягнення мети, оскільки найкоротший шлях між двома точками – це пряма лінія.

Маючи визначення нашої евристичної функції, можна описати алгоритм A* за допомогою псевдокоду як показано зображенні (рис. 2.5).

Псевдкод містить основні структури даних для обробки інформації про кожну вершину під час пошуку, зберігаючи їх та використовуючи у відповідних моментах алгоритму. Алгоритм є легкий для реалізації на будь-якій мові програмування, що є також великою перевагою.

Основна ідея – це пройти кожну із вершин по максимуму до моменту, коли знайдеться потрібна точка використовуючи нашу евклідову функцію.

Для цього потрібно створити 3 структури даних, де будуть зберігатись вершини, які ще не оброблялись, вершини, які вже обробились та результати кожної обробки вершини у вигляді словника з ключем та обробленим значенням відповідно.

```

1.  open ← MinHeap()
2.  closed ← Set()
3.  predecessors ← Dict()
4.  open.push(s, 0)
5.  while !open.isEmpty() do
6.    u, uCost ← open.pop()
7.    if isGoal(u) then
8.      return extractPath(u, predecessors)
9.    for all v ∈ u.successors()
10.     if v ∈ closed then
11.       continue
12.     uvCost ← edgeCost(G, u, v)
13.     if v ∈ open then
14.       if uCost + uvCost + h(v) < open[v] then
15.         open[v] ← uCost + uvCost + h(v)
16.         costs[v] ← uCost + uvCost
17.         predecessors[v] ← u
18.     else
19.       open.push(v, uCost + uvCost)
20.       costs[v] ← uCost + uvCost
21.       predecessors[v] ← u
22.     closed.add(u)

```

Рис. 2.5. Псевдокоду алгоритму A*

Найважливіша частина алгоритму позначена голубим кольором. Дану частину можна легко порівняти з класичним алгоритмом Дейкстри для пошуку мінімального шляху до вибраної вершини у графі. Ми записуємо наші необроблені вершини в структуру *open* разом з їхньою накопиченою вартістю від початку. Накопичена вартість – це акумульоване значення самої вершини та ваг ребер через, які проходився пошук до цієї вершини. Потім структура *open* сортує свої вершини за пов'язаною з ними накопиченою вартістю. Мінімальна різниця між алгоритмом Дейкстри і A* полягає в тому, що замість використання накопиченої вартості ми використовуємо акумульовану вартість плюс $h(v)$ – евристична передбачувана вартість до цільової вершини. Тоді структура *open* по суті сортує необроблені вершини за загальною оціночною вартістю переходу до

потрібної вершини. У цьому сенсі A^* зміщує пошук до вершин, які, ймовірно, є частиною оптимального шляху згідно нашого евристичного пошуку.

Оскільки ми зберігаємо загальну вартість на основі евристики та вершини, які знаходяться в *open* структурі, нам також слід відслідковувати справжню вартість кожної вершини, яку ми також зберігаємо. Цікаво зазначити, що якщо ми вважаємо, що наше евристичне значення дорівнює нулю для всіх вершин, що все ще є допустимою евристикою, то ми фактично використовуємо алгоритм Дейкстри. Тобто, ми додаємо стартову вершину до *open* структури, потім забираємо кожну вершину з купи і додаємо всі суміжні вершини до купи, поки ми не обробимо цільову вершину. Тому, A^* ніколи в принципі не буде гіршим, ніж алгоритм Дейкстри, а на практиці A^* призведе до набагато швидшого процесу планування місії.

Ми спростили проблему планування місії таким чином, щоб ваги ребер – це відстань уздовж ділянок доріг на карті. Однак, якби ми включили до нашої проблеми інші фактори, такі як рух транспорту, обмеження швидкості чи погода, відстань на шляху буде занадто спрощеною, щоб охопити масштаби проблеми. Щоб виправити це, ми можемо замість цього взяти передбачуваний час для перетину ділянки дороги як наш край ваги, і це враховує всі згадані фактори. Однак це робить нашу евклідову метрику відстані марною, оскільки вона більше не фіксує справжню вартість до кінцевої вершини з точки зору часу. Щоб виправити це, ми можемо використовувати нижню граничну оцінку часу до цілі в якості евклідової відстані, розділеної на максимальну швидкість, яка дозволена для всіх ділянок дороги. Тож навіть в ідеальних дорожніх та погодних умовах, а також при прямолінійному шляху до цілі, це абсолютно найменша кількість часу за яку автомобіль може подорожувати до своєї мети.

Тому, навіть, якщо спектр того, що може вплинути на вибір оптимального руху до цілі є великий, то його можливо спростити за допомогою даного алгоритма.

					ІТ-62.25.1081.01 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дат.		

2.2.2 Методи вираховування часу до зіткнення

Існує два основні підходи для обчислення часу до зіткнення двох об'єктів – симуляційний підхід та підхід на основі оцінки.

Підхід, заснований на симуляції, імітує рух кожного транспортного засобу в даному середовищі у майбутньому. На кожному етапі часу моделювання для кожного динамічного об'єкта обчислюються нові прогнози розміщення позиції. Це робиться шляхом поширення передбаченої траєкторії автомобіля. Після того, як положення всіх динамічних об'єктів буде відомим протягом певного часу в змодельованому майбутньому, проводиться перевірка, щоб визначити, чи зіткнулися якісь два чи більше динамічних об'єктів один з одним. Якщо вони зіткнулися, то відзначається місце та час зіткнення. Оскільки ці перевірки зіткнення проводяться між об'єктами, що рухаються, нам потрібно лише перевіряти зіткнення між геометричними показниками кожного об'єкта, оскільки вони будуть займати нові місця в просторі на наступному етапі часу.

Підхід, що базується на оцінці, функціонує шляхом обчислення еволюції геометричних показників кожного транспортного засобу під час руху по передбачуваному шляху. Результати – це смуги для кожного транспортного засобу, які можна порівняти на можливі зіткнення. Після обчислення їх перетину можна дослідити потенційні точки зіткнення за допомогою обчислень, якщо якісь два або більше динамічних об'єктів будуть займати один і той же простір одночасно. Якщо це вірно, потенційні точки зіткнення позначаються міткою, і для оцінки часу зіткнення використовується оцінка часу, коли кожен транспортний засіб прибуде до кожної точки зіткнення. Цей метод традиційно робить багато спрощених припущень для прискорення розрахунків. Ці припущення включають: ідентифікацію точок зіткнення на основі точок перетину об'єкта, оцінку зайнятості об'єкта на основі простих геометричних обчислень та оцінку часу досягнення точки зіткнення на основі профілю постійної швидкості.

					ІТ-62.25.1081.01 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дат.		

Кожен підхід має сильні та слабкі сторони (табл. 2.1) щодо своїх обчислювальних витрат, точності обчислення часу та типу додатків, в яких можна реалізувати той чи інший метод. Водночас потрібно спиратись також на те, що середовище в якому розроблюється модель планування також грає велику роль, оскільки наскільки швидко буде отримана інформація про динамічні та статичні об'єкти – настільки швидко цей метод обрахує час до зіткнення.

Таблиця 2.1. Порівняльна характеристика підходів обчислення часу до зіткнення

Порівняльна характеристика методів обчислення часу до зіткнення		
	Симуляційний	Оціночний
Обчислювальні витрати	Високі	Невисокі
Точність	Висока, якщо симуляційне середовище надає можливість точно визначити місцезнаходження об'єктів	Невисока через велику кількість обчислень та оцінок
Тип додатків	Симуляції	Додатки, які працюють у режимі реального часу

Підхід, що базується на оцінці, покладається на спрощення обчислень, які роблять час до зіткнення менш обчислювально дорогим з точки зору потреб в пам'яті та зусиль. Симуляційний підхід, як правило, є більш обчислювально дорогим, оскільки він включає в собі поетапні оцінки та обчислення геометричних характеристик об'єктів. Однак, завдяки спрощеним припущенням, перший підхід, зазвичай створює менш точну оцінку точки зіткнення та часу зіткнення. З іншого боку, для другого підходу не потрібно покладатися на ці наближення, і при малих розмірах симуляційних кроків можна виконувати високу оцінку вірності часу до зіткнення. Ці відносні відмінності роблять кожен

алгоритм підходящим для різних застосувань. Якщо середовище, в якому розробляється, є в автономному режимі, де важлива точність, наприклад створення набору даних та тестування на основі моделювання, перевагу має симуляційний підхід.

Оскільки моделювання буде описане в симуляційному середовищі, відповідно обраний підхід для даної проблеми буде також симуляційним чий псевдокод можна поглянути на зображенні (рис. 2.6):

```

1.   $t \leftarrow 0$ 
2.   $x_0 = x_{obj}$ 
3.  while  $t < T$  do
4.       $t = t + dt$ 
5.      for  $i \in \{1, \dots |D|\}$  do
6.           $d_i.x_t \leftarrow \text{PositionEstimation}(d_i, t)$ 
7.          for  $j \in \{i, \dots |D|\}$  do
8.               $d_j.x_t \leftarrow \text{PositionEstimation}(d_j, t)$ 
9.               $P_{coll,ij} \leftarrow \text{CollisionEstimation}(d_i.x_t, d_j.x_t, N_c)$ 
10.             if  $P_{coll,ij}$  then
11.                  $TTC_{ij} \leftarrow t$ 
12.             end
13.         end
14.     end
15. end while
16. return  $P_c, TTC$ 

```

Рис. 2.6. Псевдокод симуляційного підходу

На вхід псевдоду ми отримуємо:

- D – список всіх існуючих динамічних об’єктів;
- dt – час між кроками у симуляції;
- N_c – кількість кіл (так буде описуватись геометричні показники об’єкта) для вирахування зіткнення.

В результаті отримуємо:

- P_{coll} – список точок зіткнення;
- TTC – список всіх проміжків часу до точок зіткнення.

Алгоритм приймає як вхід список динамічних об'єктів D з прогнозованими шляхами, включаючи запланований шлях для безпілотного автомобіля, час між кроками у симуляції dt , та будь-якими параметрами, необхідними для визначення підходу перевірки зіткнення, який слід використовувати, будь це перетин багатокутників або кіл. Вихід, який дасть цей алгоритм – це точка зіткнення та час зіткнення для кожної пари об'єктів.

Алгоритм проходить ітерацію через кожний крок симуляції від поточного часу до кінця. На кожному кроці часу ми створюємо траєкторію всіх динамічних об'єктів по їх шляху на один крок, використовуючи прогнози руху динамічного об'єкта. Стани об'єкта, положення та швидкість усіх об'єктів встановлюються за допомогою цього оновлення покроково. Потім для кожної пари об'єктів ми проводимо парну перевірку зіткнень. Якщо виявлена точка зіткнення, то до цієї точки зіткнення додається поточний модельований час для оцінки часу зіткнення (рис. 2.7).

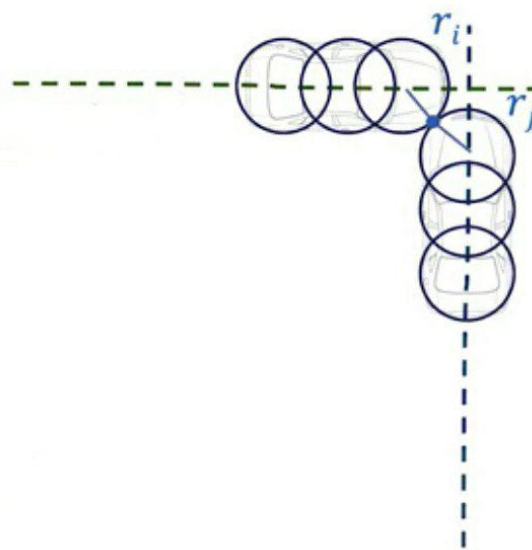


Рис. 2.7. Приблизне відображення точки зіткнення за допомогою кіл

Тут порівнюється кожна пара кіл, що призводить до дев'яти перевірок відстані для трьох кіл та 16 перевірок відстані для чотирьох кіл. Зауважте, що в

міру збільшення кількості динамічних об'єктів на сцені кількість перевірки зіткнень зростає квадратично, якщо вичерпно оцінювати всі можливі зіткнення.

Після виявлення зіткнення можна проводити розрахунок для встановлення точки зіткнення. Якщо припустити невеликі кроки симуляції, ця перша точка зіткнення є єдиною точкою, яка задовольняє обидва рівняння кіл навколо кожного об'єкта (2.6 та 2.7):

$$d_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \quad (2.6)$$

$$r_i + r_j \geq d_{i,j} \quad (2.7)$$

Перестановка призводить до наступної пари рівнянь для координат x і y точки зіткнення (2.8 та 2.9):

$$C_x = \frac{(x_i \cdot r_j) + (x_j \cdot r_i)}{(r_i + r_j)} \quad (2.8)$$

$$C_y = \frac{(y_i \cdot r_j) + (y_j \cdot r_i)}{(r_i + r_j)} \quad (2.9)$$

X -координата точки зіткнення дорівнює x -координаті центру кола I , помноженому на радіус кола J , плюс x -координата кола J , помноженому на радіус кола I , що ділиться на суму двох радіусів. Це ж схоже рівняння використовується для y -координати.

Як це часто буває, існує компроміс між точністю оцінки часу до зіткнення та загальним часом обчислення. У міру того, як ми збільшуємо вірність перевірки зіткнення, ми також збільшуємо кількість необхідних обчислень. Так само, як видно в цьому прикладі, якщо зменшити кількість розглянутих кіл до

одного, кількість обчислень зменшується, але точність точних точок зіткнення значно страждає.

2.2.3 Скінченний автомат для планування поведінки

У русі безпілотного автомобіля є ряд різноманітних сценаріїв, з якими доведеться так чи інакше стикнутися. Наприклад, є тристоронні перехрестя зупинки, перехрестя, керовані світлофором, та сценарії прямої дороги. Незважаючи на те, що багато з цих сценаріїв поділяють деякі подібності багато в чому, кожна із ситуацій може вважатися такою, що вимагає принципово різної поведінки водіння. Тому для того, щоб виконати правильний перехід з одного стану автомобіля в інший, введене поняття скінченного автомату як представлення дорожніх ситуацій та можливих переходів між ними.

Ми можемо представити кожен високорівневий сценарій як єдиний стан. Це дозволяє створити набагато простіші переходи між сценаріями високого рівня. На зображенні (рис. 2.8). можна побачити, що стани представляють собою сценарій прямої дороги та сценарій перехрестя з чотирма сторонами:

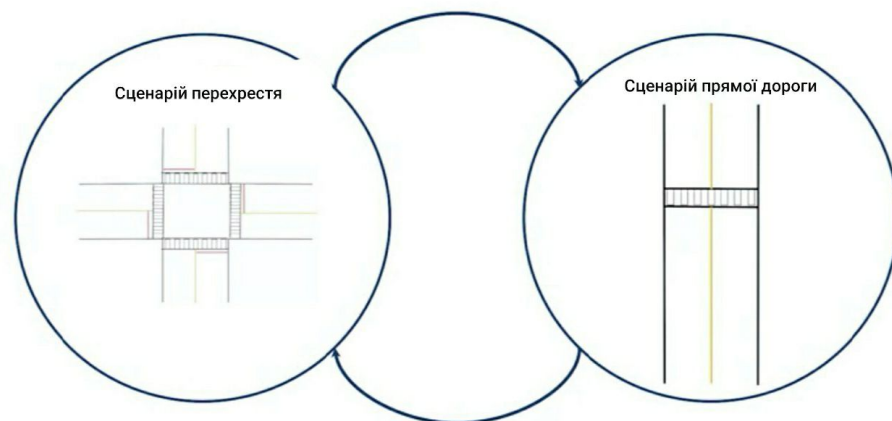


Рис. 2.8. Перехід між сценаріями перехрестя та прямої дороги

З кожним із цих високорівневих сценаріїв зберігатиметься автомат низького рівня, з яким ми можемо самостійно впоратися із заданим сценарієм.

Приклад можна поглянути на зображенні (рис. 2.9), де стани:

- 1 – зменшити швидкість до зупинки;
- 2 – зупинитись;
- 3 – слідувати машині, що їде попереду;
- 4 – слідувати за швидкістю автомобіля.

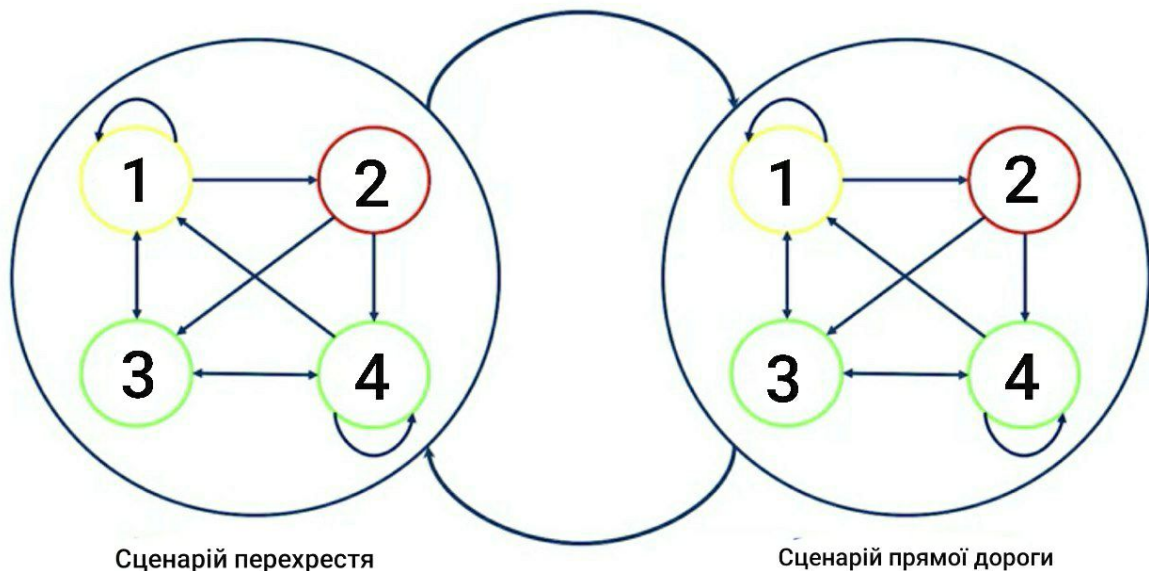


Рис. 2.9. Використання низькорівневих автоматів для сценаріїв

Цей спосіб представлення відомий як ієрархічний скінченний автомат, з високорівневими станами, що представляють кожен сценарій, і низькорівневими станами, що представляють маневри, які слід обробляти в кожному сценарії. Переходи між першими – це правило, яке визначає, яке визначає, коли буде введено новий сценарій на основі HD-мапи та динамічної інформації про транспортний засіб.

Цей тип скінченних автоматів має деякі властиві сильні та слабкі сторони. Сильні сторони цього підходу зумовлені його розбиттям високорівневих станів на низькорівневі стани. Створюючи окремі автомати, цей метод здатний обмежити час обчислення, необхідний на будь-якому етапі часу, шляхом реструктуризації нашого підпростору. Цей підрозділений характер також значно полегшує тестування та програмування даного типу автомата.

Однак ієрархічна державна машина не позбавлена своїх проблем: в той час як обмежується кількість необхідних правил для переходів, цей підхід все ще не в змозі повністю впоратися зі збільшенням цих правил. Однією з причин великої кількості правил є те, що всі сценарії мають повністю відокремлені низькорівневі автомати, і тому багато правил є дублікатами один одного для того, щоб обробляти подібну поведінку в різних сценаріях. Зрештою, ієрархічний підхід дозволяє розробнику керувати більшим обсягом складності, ніж прості кінцеві автомати. Насправді, перспектива, застосована автоматами з кінцевими станами, що одна поведінка активна в будь-який момент часу і що всі переходи до іншої поведінки можуть бути чітко визначені, має обмеження щодо великого набору сценаріїв, з якими вона може працювати. Тим не менш, цей підхід пропонує корисний інструмент для типового планування поведінки в безпілотному автомобілі.

2.2.4 Оптимізація проблеми планування локалізації

Як зазначалось раніше кубічна спіраль не має фінального рішення для положення в кінці спіралі. Для того, щоб використати обмеження, які описувались раніше, нам потрібно використати один із чисельних методів розрахунку подібних інтегралів. Таких підходів існує чимало, але найкраще з них підходить метод Сімпсона, який є один із видів апроксимативних підходів для чисельного інтегрування.

Метод Сімпсона – це одна із найпопулярніших технік чисельного інтегрування, яка, як правило, більш точна, ніж інші простіші чисельні методи. Це тому, що вона вирішує інтеграл даної функції у квадратичній формі, а не інтеграл лінійної функції, як наприклад у методі трапецій чи середньої точки. Загальна формула виглядає так (2.10):

$$\int_0^s f(s') ds' \approx \frac{s}{3n} (f(0) + 4f\left(\frac{s}{n}\right) + 2f\left(\frac{2s}{n}\right) + \dots + f(s)) \quad (2.10)$$

Даний метод працює на основі визначення декількох однаково розподілених поділів області інтегрування n , а потім підсумовування у кожному поділі та граничній точці. Для прикладу: якщо ми обираємо n , рівним 4, то ми нашу область на чотири однакові за розміром відрізки, тому ми отримуємо п'ять точок, які потрібно врахувати при агрегації. Кожен доданок у сумі - це функція, яка вираховується в точці ділення, помножена на відповідний коефіцієнт. Якщо детально подивитись на загальний вигляд метода Сімпсона, то можна помітити, що існує чергування коефіцієнтів чотири та два для кожного члена суми, за винятком доданка кінцевої точки, який має множник рівний одиниці. Коли ми ділимо на більшу кількість відрізків n , то відповідно точність обчислювального інтеграла зростає.

Хоч метод Сімпсона має можливість врахувати всі обмеження, які були описані раніше, планування локалізації зосереджує свою увагу саме на кривизні спіралі. Будь-який автомобіль має свій мінімальний радіус повороту, тому на даний момент припустимо, що транспортний засіб може досягти мінімального радіусу повороту в два метри. Це відповідає максимальній кривизні в 0.5 радіан. Проте для кожної точки спіралі у такому випадку складно описати обмеження по кривизні.

Для простоти обмежимо кривизну в одній третині та двох третіх точках кривої як це показано на зображенні (рис. 2.10), таким чином буде легше обрати точки на які варто звернути увагу при обчисленні та також обмежити їхню кількість для простоти:

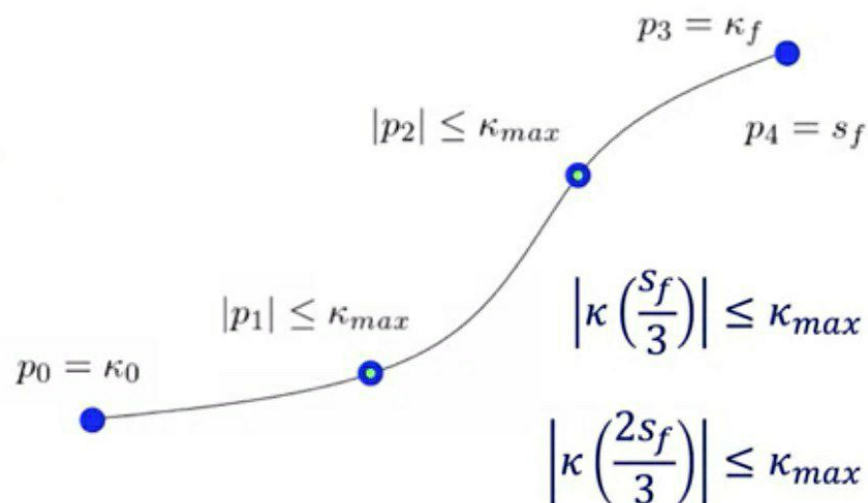


Рис. 2.10. Визначення точок на спіралі, де буде проводитись обчислення щодо обмеження кривизни

Викривлення початкової та кінцевої точки вже були обмежені в початкових граничних умовах. Тому, в результаті, можна отримати обмеження кривизни у вигляді функції параметрів спіралі, і ми маємо всі необхідні обмеження для вирішення проблеми локалізації.

Отримавши обмеження кривизни у визначених точках потрібно отримати мінімізовану функцію, яка задається на самому початку. Один із способів зробити це - розподілити абсолютну кривизну рівномірно по шляху. Це можна зробити, зводячи до мінімуму енергію вигину нашої запланованої параметричної кривої.

Енергія кривизни кривої – це інтеграл її кривизни в квадраті по всій довжині дуги шляху. Оскільки у нас є поліноміальна функція кривизни, що описує нашу кубічну спіраль, інтеграл енергії вигину є визначеним з точки зору параметрів спіралі.

Використовуючи обмеження кривизни та розподіл максимальної кривизни по всі спіралі ми отримуємо кінцеве рішення оптимізації проблеми локалізації на зображенні (рис. 2.11):

$$\min f_{be}(a_0, a_1, a_2, a_3, s_f) + \alpha(x_s(p_4) - x_f) + \beta(y_s(p_4) - y_f) + \gamma(\theta_s(p_4) - \theta_f)$$

$$\text{s. t. } \begin{cases} |p_1| \leq \kappa_{max} \\ |p_2| \leq \kappa_{max} \end{cases}$$

Рис. 2.11. Узагальнене рішення оптимізації проблеми локалізації

Хоч під капотом даного рішення є багато математичних обчислень, проте в більшості мовах програмування (в яких це підтримується) дана проблема вирішується за допомогою використанням однієї імплементованої функції з відповідної бібліотеки, де можна задати кількість точок, де буде розраховуватись кривизна та абсолютна кривизна на всій спіралі.

2.3 Конкретизація постановки задачі

Розглянувши основні методи та підходи для кожної із ієрархічної моделі проблеми планування руху безпілотного автомобіля тепер можна коротко конкретизувати нашу поставлену задачу:

- обрати симуляційне середовище, яке буде мати всі властивості HD-мапи та надавати можливість отримувати будь-яку інформацію про місцезонашування статичних та динамічних об'єктів, межі доріг та перехресть у зрозумілому вигляді;
- налаштувати середовище у симуляції, визначивши, де та які статичні об'єкти знаходяться на карті, які геометричні характеристики та який рух будуть проходити динамічні об'єкти, а також визначити початкову та кінцеву точку руху безпілотного автомобіля;
- на основі початкової та фінальної точки руху автомобіля та меж, де знаходяться повороти доріг, перехрестя, злиття двох смуг та інше, побудувати граф з відповідними вершинам і знайти найкоротший шлях до цілі за допомогою алгоритма A*;
- врахувавши, які перехресні та смугові елементи є на HD-мапі, а також, які види статичних та динамічних об'єктів беруть участь в русі,

- побудувати ієрархічний скінченний автомат, де будуть визначені основні стани та їх переходи;
- маючи всі точки на мапі, через які рухається машина, побудувати нашу кубічну спіраль, вивести її у вигляді графіка, перед цим застосувавши метод Сімпсона для оптимізації параметричного графіка та винести цю інформацію на безпілотний автомобіль, щоб він провів гладкий та безаварійний маневр чи рух на дорозі;
 - пов'язати всі етапи описані вище у загальний планувальник руху безпілотного автомобіля та надати інформацію контролеру, який вбудований в симуляцію та показати як поводить ся транспортний засіб у визначеній дорожній ситуації.

ВИСНОВКИ ДО РОЗДІЛУ

У другому розділі були висвітлені основні проблеми прототипу та були деталізовані етапи, які потрібні для побудови планувальника руху безпілотного автомобіля. Також було проведене порівняння методів та підходів для вирішення проблем кожного етапу та були обрані ті, які можуть найкраще підійти для кожного із рівнів та які можливо реалізувати за допомогою програмного коду. Кожен із методів пов'язується між собою та отримує результати один від одного, що дозволяє зібрати їх в одне ціле та спроектувати модель планування маршруту. В результаті була конкретизована постановка задачі у поетапному вигляді.

					ІТ-62.25.1081.01 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат.		51

РОЗДІЛ 3 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ПЛАНУВАННЯ

У даному розділі розглянуто процес створення моделі побудови маршруту та рішення задачі планування для безпілотного автомобіля у вигляді запуску додаткових Python-скриптів для кожного етапу проблеми на основі симуляційного середовища, яке запускається як настільний додаток з багаторівневою архітектурою

3.1 Тип додатку

Розробка настільних додатків – це розробка автономних додатків для ноутбуків та комп'ютерів. Дана розробка передбачає собою написання програмних засобів, доступних як окремі програми на комп'ютерах або ноутбуках, як, наприклад, на відміну від веб-програми, яка вимагає запуску веб-браузера та мобільних додатків, які працюють у смартфонах та планшетах.

Настільні програми корисні для бізнесу, починаючи з малого та середнього до великого бізнесу. Завдяки хорошему графічному інтерфейсу (GUI), настільний додаток може використовувати всі обчислювальні сили машини, на якій він працює, і виконувати всі функції, які він має виконувати.

Хоч популярність WEB-додатків набирає все більше і більше оборотів, існує багато вагомих причин для розробки настільних додатків. Деякі з них:

- програму не потрібно підключати до Інтернету;
- настільні програми мають кращу продуктивність, ніж веб-програми;
- запуск серйозних алгоритмів в даному середовищі набагато кращий, ніж у веб-додатку;
- використання потоків обробки інформації набагато простіше та ефективніше у настільному додатку;
- великий спектр виконання різноманітних задач, в тому числі з напрямом у дослідженнях наукової діяльності.

3.2 Архітектура проекту

Обрана багат шарова архітектура програмного забезпечення – це одна з найпопулярніших архітектурних моделей на сьогоднішній день, яка полегшує розробку, щоб вона проходила в більш спритній формі. Іноді називається багаторівневою архітектурою або n-ярусною архітектурою, багат шарова архітектура програмного забезпечення складається з різних шарів, кожен з яких відповідає різній службі або інтеграції. Оскільки кожен шар окремий, внесення змін до кожного шару простіше, ніж необхідність вирішувати проблеми всієї архітектури зразу.

Основні шари, які використовуються в більшості сучасних додатків та будуть використовуватись у розробці нашої моделі:

- презентаційний шар – це перший і найвищий шар, який присутній у багаторівневій програмі. Цей рівень забезпечує презентаційні послуги, тобто представлення вмісту додатку для кінцевого користувача через GUI. Доступ до цього рівня можна отримати через будь-який тип клієнтських пристроїв, таких як робочий стіл, ноутбук, планшет, мобільний, тонкий клієнт тощо. Щоб представити вміст, для цього рівня важливо взаємодіяти з іншими рівнями, які є перед ним;
- прикладний шар – це середній рівень даної архітектури. Це той рівень, в якому працює бізнес-логіка програми. Бізнес-логіка – це набір правил, необхідних для запуску програми відповідно до рекомендацій. Компоненти цього ярусу зазвичай працюють на одному або декількох серверах додатків;
- рівень даних – це найнижчий рівень цієї архітектури і стосується в основному зберігання та пошуку даних програми. Дані програми, як правило, зберігаються на сервері баз даних, файловому сервері або будь-якому іншому пристрої чи носії, який підтримує логіку доступу до

даних і забезпечує необхідні кроки для забезпечення викриття лише тих даних, які можна отримати без надання доступу до механізмів зберігання та пошуку даних. Це робиться за допомогою рівня даних, надаючи API для рівня додатків. Забезпечення цього API надає повну прозорість операцій з даними, які обробляються на цьому рівні, не впливаючи на рівень програми. Наприклад, оновлення або оновлення систем цього рівня не впливають на рівень програми цієї архітектури.

На зображенні (рис. 3.1) показано діаграму взаємодії кожного із рівнів між собою:

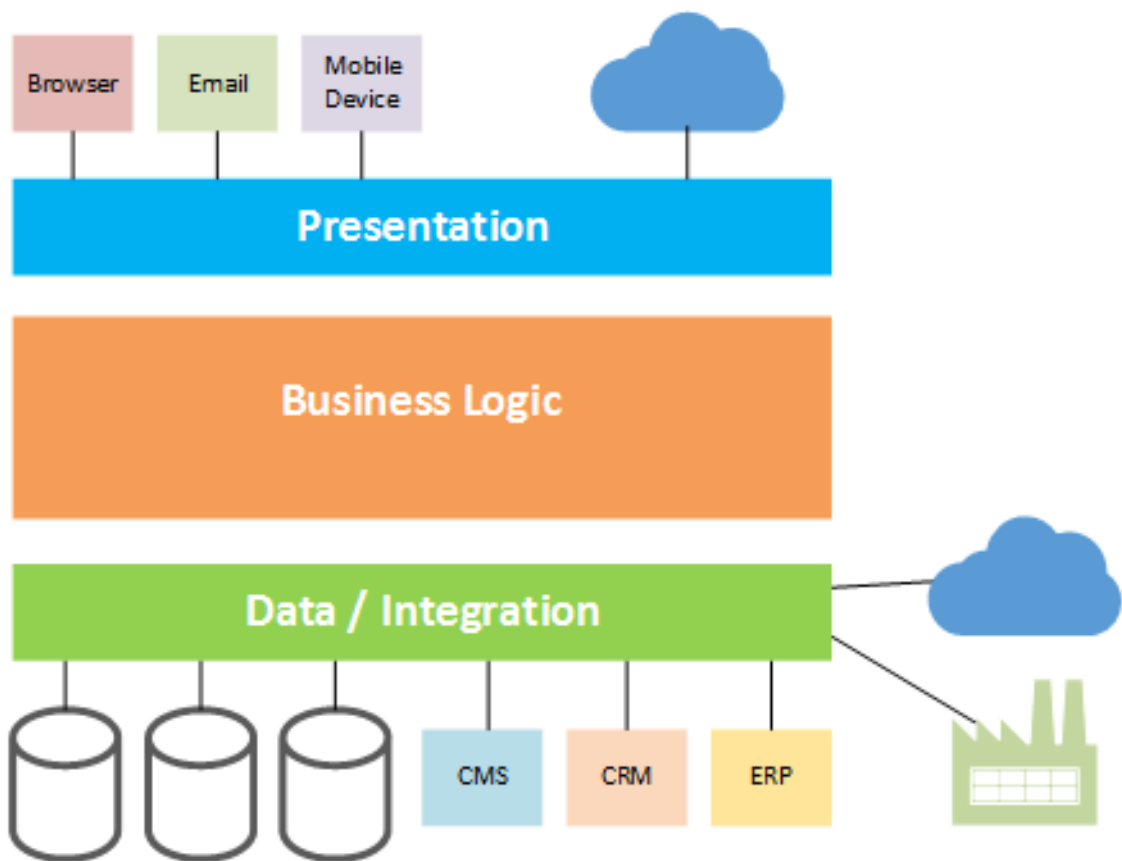


Рис. 3.1. Діаграма взаємодії рівнів тришарової архітектури додатків

Багатошарова архітектура програмного забезпечення має ряд переваг – саме тому вона стала такою популярною архітектурною схемою в останні роки. Найголовніше, що багаторівнева сегрегація дозволяє відповідно керувати та

підтримувати кожен шар. Теоретично це повинно значно спростити спосіб управління інфраструктурою програмного забезпечення.

Багатошаровий підхід особливо хороший для створення WEB-масштабних, виробничих і хмарних додатків дуже швидко та відносно без ризиків. Проте для настільних додатків цей тип архітектури також підходить, у випадку великого числа взаємодій різних сутностей та кількості можливих обчислень. Це також полегшує оновлення будь-яких застарілих систем – коли ваша архітектура розбита на кілька шарів, зміни, які потрібно внести, повинні бути простішими і менш обширними, ніж вони могли б бути в іншому випадку.

3.3 Використані технології

Архітектура з декількома шарами взаємодії надає можливості розширювати бізнес-логіку для виконання та опису всіх функцій, які потрібні для конкретної задачі. У нашому випадку потрібно реалізувати всі методи, які описувались для кожного етапу вирішення задачі планування. Для цього буде використана мова програмування Python та її залежні бібліотеки NumPy та SciPy для написання скріптів, які тим чи іншим способом виконують обчислення на кожному рівні проблеми та СУБД MySQL для керування реляційними даними, які надаються симуляційним середовищем.

3.3.1 Python

Python – це об'єктно-орієнтована мова програмування, розроблена на основі С. За своєю природою це мова програмування високого рівня, яка дозволяє створювати як прості, так і складні додатки, які виконують різні функції. Поряд із цим Python оснащений широким набором модулів, а також бібліотеками, що дозволяє йому підтримувати безліч різних мов програмування, таких як Java, С, С ++ та JSON та великий спектр задач, в тому числі математичних.

					ІТ-62.25.1081.01 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дат.		

Коли ви пишете програму на C / C++, вам слід її скомпілювати. Компіляція передбачає переклад зрозумілого для людини коду на машинно- зрозумілий код або машинний код. Машинний код – це форма інструкцій базового рівня, яка може бути безпосередньо виконана процесором. Після успішної компіляції ваш код створює виконуваний файл. Виконання цього файлу виконує операції з вашим кодом поетапно.

Python в свою чергу – це інтерпретована мова, а не компільована, хоча компіляція присутня і являється одним із кроків. Код Python, записаний у *.py*-файлі, спочатку компілюється до того, що називається байт-кодом, який зберігається у форматі *.pyc* або *.pyo*.

Замість перекладу вихідного коду до машинного коду, як C++, Python код перекладається у байт-код. Цей байт-код – це набір низьких рівнів інструкцій, який може виконувати перекладач. На більшості ПК інтерпретатор Python встановлений за адресою `/usr/local/bin/pythonX.X`. Замість того, щоб виконувати інструкції на процесорі, інструкції по байт-коду виконуються на віртуальній машині.

Однією з вагомих переваг інтерпретованих мов є те, що вони залежать від платформи. Поки байт-код Python і віртуальна машина мають однакову версію, байт-код Python може виконуватися на будь-якій платформі (Windows, MacOS тощо).

Динамічна типізація – ще одна перевага. У мовах статичного типу, таких як C++, ви повинні оголосити тип змінної, і будь-яка розбіжність, наприклад додавання рядка та цілого числа, перевіряється під час компіляції. У строго типізованих мовах, таких як Python, це завдання інтерпретатора перевірити достовірність типів змінних та виконуваних операцій.

Основні особливості даної мови програмування включають в собі:

- Python – дуже зручна для розробників мова, що означає, що кожен може навчитися користуватись нею. Порівняно з іншими об'єктно-

орієнтованими мовами програмування, такими як Java, C, C++ та C#, Python є однією з найпростіших у вивченні;

- ця мова програмування з відкритим кодом, що означає, що кожен може створювати та сприяти його розвитку. У Python є інтернет-форум, де щодня збираються тисячі програмістів, щоб вдосконалити цю мову. Поряд з цим Python вільний для завантаження та використання в будь-якій операційній системі, будь це Windows, Mac або Linux;
- GUI або графічний інтерфейс користувача є одним з ключових аспектів будь-якої мови програмування, оскільки він має можливість робити результати більш наочними. Python має підтримку широкого набору графічних інтерфейсів, які можна легко імпортувати до інтерпретатора, завдяки чому це є однією з найулюбленіших мов для розробників;
- одним із ключових аспектів Python є його об'єктно-орієнтований підхід. Це в основному означає, що Python реалізовує поняття інкапсуляції класів і об'єктів, що дозволяє програмам бути ефективними в довгостроковій перспективі;
- припустимо, ви запускаєте програму в Windows і вам потрібно перенести те саме на MacOS або Linux, тоді ви можете легко досягти того ж у Python, не турбуючись про зміну коду;
- з іншого боку, Python оснащений великою кількістю бібліотек, які можна імпортувати в будь-якому випадку та використовувати у певній програмі. Присутність бібліотек також гарантує, що вам не потрібно писати весь код самостійно і можете імпортувати той самий з тих, що вже є в бібліотеках.

Щодо імпортованих модулів Python, які потрібно використати для математичних обчислень, найкращим вибором на даний момент є NumPy.

					ІТ-62.25.1081.01 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дат.		

3.3.2 NumPy та SciPy

NumPy – найбільш фундаментальний пакет наукових обчислень в Python і є базою для багатьох інших пакетів. Оскільки Python спочатку не був розроблений для чисельних обчислень, така потреба виникла в кінці 90-х, коли Python почав ставати популярним серед інженерів та програмістів, яким потрібні швидші векторні операції. Багато популярних пакетів машинного навчання та обчислювальної техніки використовують деякі функції NumPy, і найголовніше – вони активно використовують масиви NumPy у своїх методах, що робить NumPy важливою бібліотекою для наукових проектів.

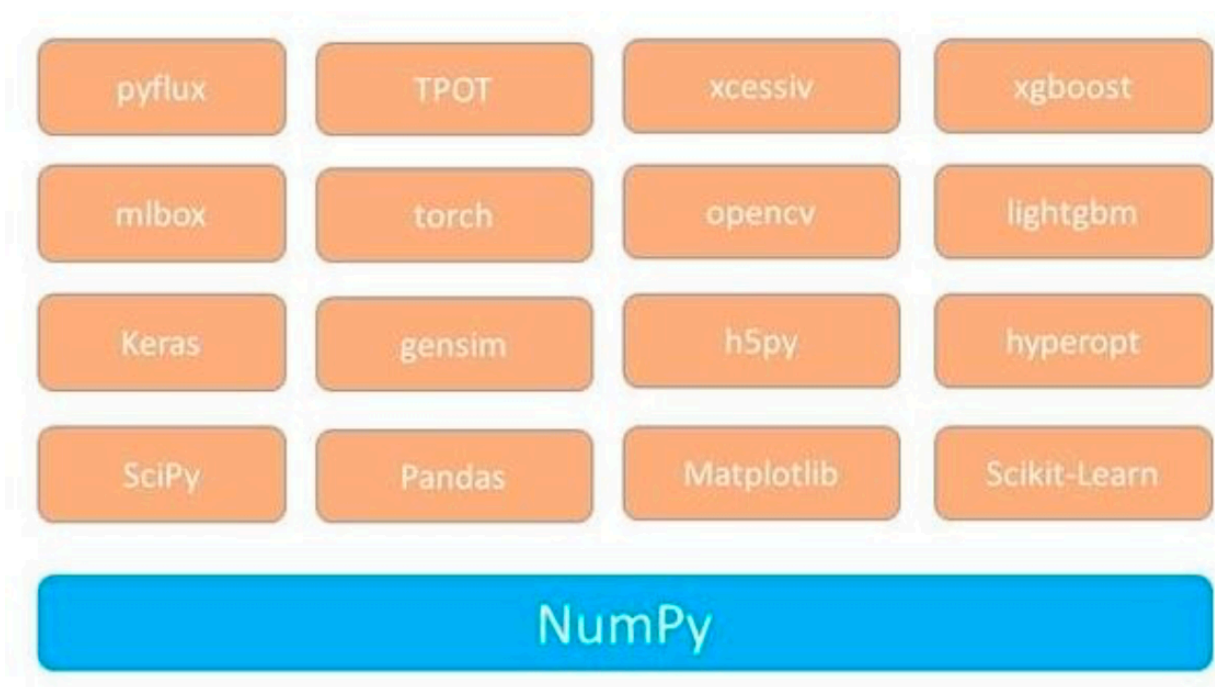


Рис. 3.2. Стек пакетів NumPy

Для числових обчислень в основному працюють з векторами, матрицями та рівняннями. Ви можете маніпулювати ними різними способами, використовуючи різні математичні функції. NumPy ідеально підходить для подібних ситуацій, оскільки дозволяє користувачам ефективно виконувати свої обчислення. Незважаючи на те, що списки Python дуже легко створювати та маніпулювати, вони не підтримують векторизованих операцій. Python не має

елементів фіксованого типу в списках, і, наприклад, цикл не дуже ефективний, оскільки при кожній ітерації тип даних потрібно перевіряти. Однак у масивах NumPy тип даних є фіксованим і підтримує також векторизовані операції. NumPy не просто більш ефективний в операціях з багатовимірним масивом порівняно зі списками Python, він також пропонує багато математичних методів, які ви можете застосувати, як тільки його імпортують. NumPy – це основна бібліотека наукового стека даних Python.

SciPy має міцний зв'язок з NumPy, оскільки він використовує багатовимірні масиви NumPy як базову структуру даних для своїх наукових функцій для лінійної алгебри, оптимізації, інтерполяції, інтеграції, обробки сигналів та зображень та інше. SciPy був побудований на базі масиву NumPy та покращив наукове програмування з його передовими математичними функціями. Тому деякі частини API NumPy були переміщені до SciPy. Цей зв'язок з NumPy робить SciPy більш зручним для передових наукових обчислень у багатьох випадках.

Підсумовуючи все, ми можемо визначити переваги NumPy наступним чином:

- відкритий код і нульова вартість;
- мова програмування високого рівня із зручним синтаксисом;
- більш ефективно, ніж списки Python;
- має більш вдосконалені вбудовані функції і добре інтегрований з іншими бібліотеками.

Люди, яким потрібно працювати з даними та робити аналіз, моделювання чи прогнозування, повинні ознайомитись із використанням NumPy та його можливостями, оскільки це допоможе їм швидко преобразувати та протестувати свої ідеї. Якщо ви працюєте професіоналом, ваша фірма, швидше за все, хоче використовувати методи аналізу даних для того, щоб перейти на крок попереду

своїх конкурентів. Якщо вони зможуть краще зрозуміти отримані дані, вони зможуть краще зрозуміти бізнес, і це призведе до прийняття кращих рішень.

Проте, щоб обробляти дані та отримувати на їхній основі результат, потрібно їх десь зберігати та отримувати в будь-який момент, коли це потрібно.

3.3.3 MySQL

MySQL – це СУБД з відкритим кодом, заснована на структурованій мові запитів (SQL). MySQL доступний у всіх основних операційних системах, включаючи Windows, Linux та MacOS. Він безкоштовний для використання для фізичних осіб та виробничих середовищ за загальною ліцензією GNU; однак, якщо використовується комерційно, потрібна комерційна ліцензія.

MySQL, як і інші реляційні бази даних, зберігає дані в таблицях, стовпцях та рядках. Кожен запис визначається унікальним ідентифікатором. MySQL був розроблений та оптимізований для арени веб-розробок; це, мабуть, найпоширеніша база даних, яка використовується при розгортанні веб-серверів та настільних додатків. Сьогодні MySQL використовує кожен 9 з 10 веб-сайтів в Інтернеті, і це база даних, обрана Facebook, Twitter та Wikipedia.

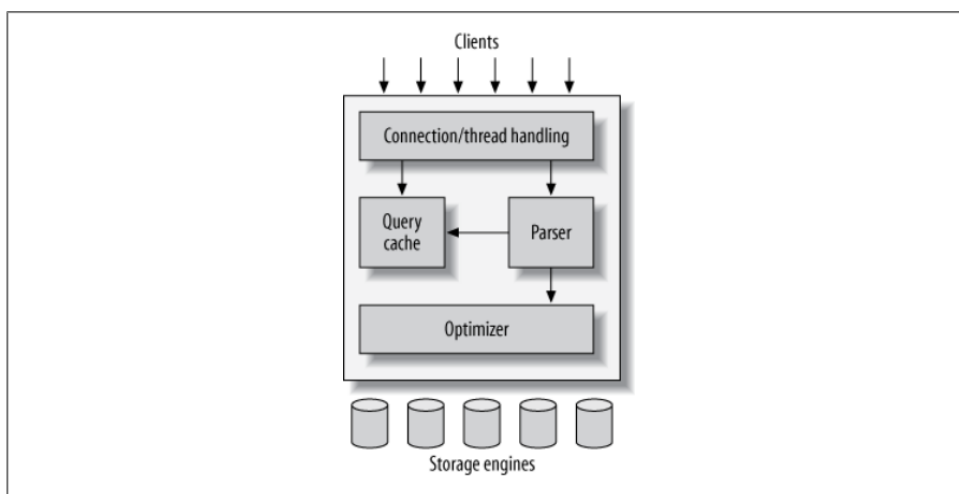


Рис. 3.3. Архітектура MySQL

Архітектура MySQL складається з 3 рівнів (рис. 3.3). Верхній шар забезпечує підключення, автентифікацію, безпеку, яка забезпечується мережевими інструментами до даних

Другий рівень, тобто сервер – це мозок MySQL, він включає розбір запитів, аналіз, оптимізацію, кешування та всі вбудовані функції (наприклад, дати, час, математика та шифрування). Будь-яка функціональність, що надається в системах зберігання даних, працює на цьому рівні: наприклад, збережені процедури, тригери та перегляди.

Третій рівень, тобто двигуни зберігання, вони відповідають за зберігання та отримання всіх даних, що зберігаються в MySQL. MyISAM та InnoDB – часто використовувані двигуни зберігання даних. Сервер MySQL спілкується з механізмом зберігання даних через API у WEB-додатках та через DAO у настільних додатках. DAO абстрагує сутності системи і робить їх відображення на БД, визначаючи загальні методи використання з'єднань, отримання даних, закриття і (або) повернення в Connection Pool.

Процес, який виконується, коли надсилається запит у MySQL виглядає так:

- клієнт відправляє SQL скрипт на сервер;
- сервер перевіряє кеш запитів. Якщо такий запит вже відбувався, він повертає збережений результат із кешу; в іншому випадку він передає SQL скрипт наступному кроку;
- сервер аналізує, попередньо обробляє та оптимізовує SQL в план виконання запитів;
- двигун виконання запитів виконує план, здійснюючи запити в API двигуна зберігання;
- сервер відправляє результат клієнту.

3.4 Розробка та реалізація моделі планування руху

3.4.1 Опис апаратної бази та середовища розробки

Апаратна база та середовища розробки тісно пов'язані з обраним симулятором, який вміщує в собі всі характеристики HD-мапи та надає інформацію для рішення задачі планування маршруту безпілотного автомобіля. Для нашої моделі був обраний CARLA Simulator.

CARLA – автономний симулятор водіння з відкритим кодом. Він був побудований з нуля, щоб надавати модульний та гнучкий API для вирішення цілого ряду завдань, що стосуються проблеми автономного водіння. Однією з головних цілей CARLA є допомогти демократизувати дослідницьку та розробницьку діяльність у сфері автономного управління, слугуючи інструментом, до якого користувачі можуть легко отримати доступ та налаштувати його. Для цього симулятор повинен відповідати вимогам різних випадків використання в рамках загальної проблеми водіння. CARLA заснована на Unreal Engine для запуску моделювання та використовує стандарт OpenDRIVE (1.4 на сьогодні) для визначення доріг та міських налаштувань. Контроль над моделюванням надається за допомогою API, який обробляється в Python та C++, яке постійно зростає в міру розвитку проекту (рис. 3.4).

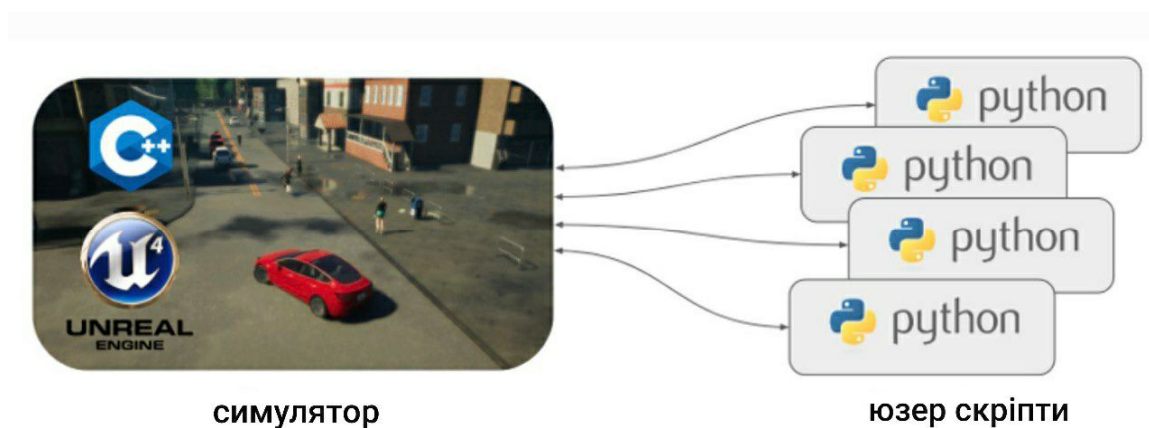


Рис. 3.4. Взаємодія симулятор та програмного коду користувача

Для того, щоб згладити процес розробки, навчання та валідації водійних систем, CARLA перетворилася на екосистему проєктів, побудовану навколо основної платформи. У цьому контексті важливо зрозуміти деякі речі про те, як працює CARLA, щоб повністю зрозуміти її можливості.

Симулятор CARLA складається з масштабованої архітектури клієнт-сервер.

Сервер відповідає за все, що стосується самого моделювання: візуалізація датчиків, обчислення фізики, оновлення світлової та кольорової гами, її дійових осіб та багато іншого. Оскільки він спрямований на реалістичні результати, найкращим чином буде працювати сервер із виділеним графічним процесором, особливо якщо це стосується машинного навчання.

Клієнтська сторона складається з суми клієнтських модулів, що контролюють логіку акторів на сцені та встановлюють умови середовища, в якому рухається безпілотний автомобіль. Це досягається за допомогою використання CARLA API (в Python або C++), шару, який є посередником між сервером і клієнтом, який постійно розвивається для надання нових функціональних можливостей.

Це підсумовує основну структуру симулятора. Проте, щоб зрозуміти CARLA насправді потрібно набагато більше, оскільки в ній співіснує багато різних особливостей та елементів. Деякі з них перераховані нижче, щоб отримати перспективу щодо можливостей, яких може досягти CARLA:

- керівник трафіку – вбудована система, яка бере контроль над транспортними засобами, крім тієї, яка використовується для навчання. Він виступає провідником, наданим CARLA для відтворення міських середовищ із реалістичною поведінкою;
- датчики – транспортні засоби покладаються на них, щоб розповсюджувати інформацію про своє оточення. У CARLA вони є специфічним видом актора, який прикріплений до транспортного

засобу, і отримані ними дані можна зберігати для полегшення процесу та оновлення обчислень на кожному етапі задачі. В даний час проект підтримує різні типи: від камер до радарів, лідарів та багатьох інших.

- диктофон – ця функція використовується для відновлення моделювання поетапно для кожного актора у світі. Він надає доступ до будь-якого моменту часової шкали в будь-якій точці середовища, створюючи чудовий інструмент відстеження;
- відкриті для доступу заготовлені карти – окрім того, що симулятор надає можливість власноруч налаштувати середовище, CARLA надає різні карти для міських налаштувань з контролем погодних умов та бібліотекою креслення з широким набором акторів. Однак ці елементи можна налаштувати та створити нові, керуючись простими вказівками.

Стандартне середовище, яке надається симулятором можна поглянути на зображенні (рис. 3.5):



Рис. 3.5. Стандартна карта у CARLA

Симулятор підтримується на Windows та Linux операційних системах (на разі MacOS не підтримує CARLA). Рекомендовані технічні характеристики включають в собі:

- Чотириядерний процесор Intel або AMD, 2,5 ГГц або швидше;
- Карти серії NVIDIA GeForce 470 GTX або AMD Radeon 6870 HD або новішої версії;
- 8 ГБ оперативної пам'яті;
- ~ 10 Гб місця на жорсткому диску для налаштування тренажера.

Це водночас і є характеристики для нашої апаратної бази, а для середовища розробки підійде будь-який IDE, в якому можна писати та інтерпретувати Python-скріпти для кожного етапу планування руху безпілотного автомобіля.

3.4.2 Опис структури програмного продукту

Структуру написаного програмного забезпечення можна поділити на декілька основних етапів.

Спочатку налаштовується середовище для безпілотного автомобіля у симуляторі, визначивши такий сценарій: безпілотний автомобіль рухається спочатку по прямій дорозі, попереду неї їде інша машина з однаковою швидкістю, а також далі зустрічається знак зупинки та поворот направо на перехресті.

Дана ситуація описує те, з чим стикається кожен із автомобілів, і не тільки безпілотні, кожного дня. Водночас тут беруть участь, як і перехресні, так і смугові елементи, що також показує типовість ситуації, але підвищує складність.

Далі потрібно отримати дані про все середовище, в якому знаходиться автономний транспортний засіб. Сюди входить розташування динамічних (машина, яка їде попереду) та статичних (знак зупинки) об'єктів, дорожня розмітка (поворот направо на перехресті), геометричні та фізичні

					ІТ-62.25.1081.01 ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дат.		

характеристики обох автомобілів, початкова та кінцева точка руху. Дані отримуються через API CARLA Simulator.

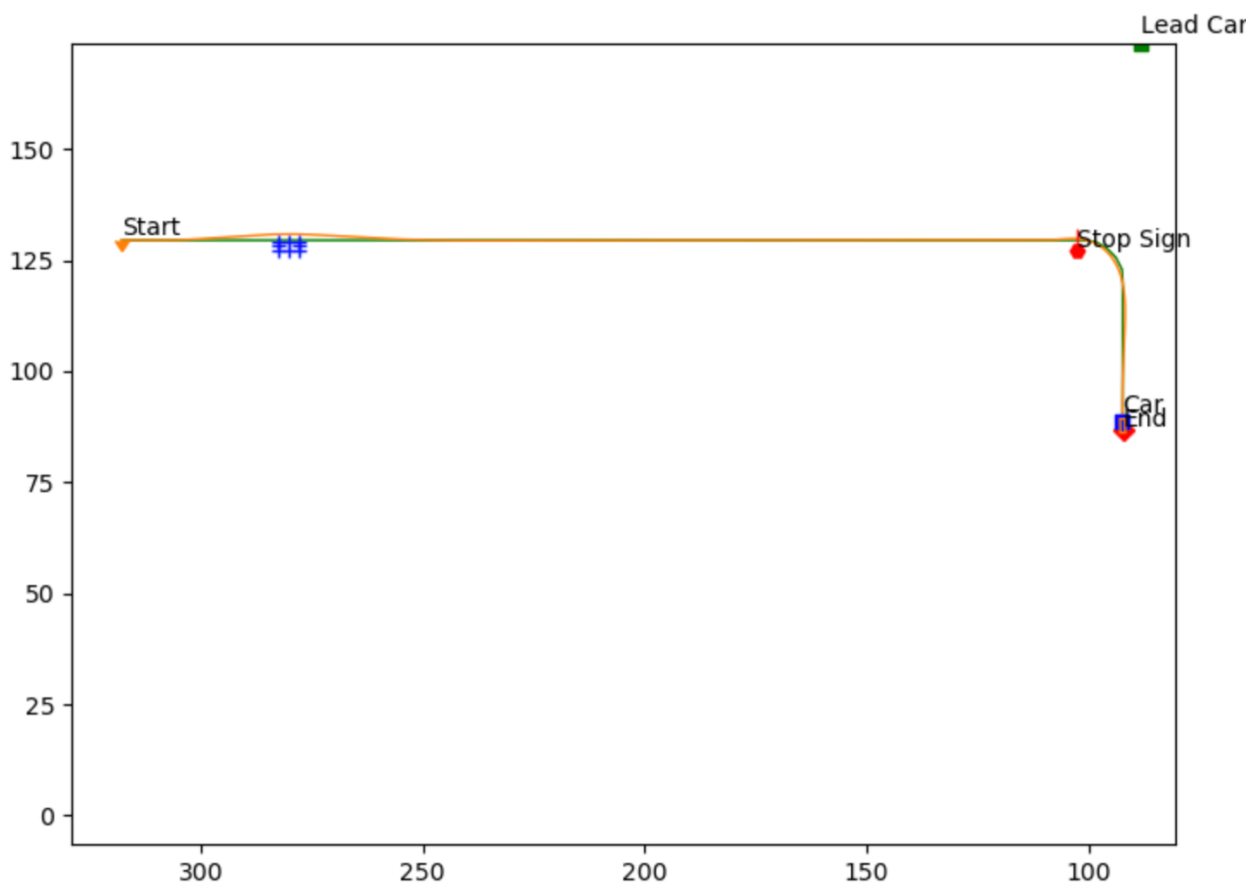


Рис. 3.6. Графічне представлення спіралі руху безпілотного автомобіля

Отримавши дані про саму карту та її компоненти, вони записуються у відповідні таблиці MySQL використовуючи написаний DAO. Таблиці поділені за основними характеристиками, тобто створені окремі таблиці для динамічних та статичних об'єктів, таблиця для кожної із точок на дорозі, через яку може проїхати автомобіль, таблиця для характеристик об'єктів;

На основі даних, які знаходяться у таблицях та графічних об'єктів, які додаються у форматі .png, будується спіраль на GUI-полотні, по якій буде проїжджати машина (рис. 3.6).

Ці етапи прописуються в окремих Python-модулях програмного забезпечення, які запускаються через основний модуль *full_planner.py*. В нього

також входять додаткові скріпти для кожного із етапів задачі планування маршруту, а саме:

- `mission_planner.py` – модуль для планування місії. Оскільки, в нас визначені кінцева та початкова точка руху автомобіля, то нам не потрібно програмно будувати граф маршруту транспортного засобу. Тому призначення даного модуля – це отримати початок та кінець рузу з відповідної таблиці та віддати їх у вигляді результату;
- `time_to_collision_checker.py` – модуль розраховування часу до зіткнення між безпілотним автомобілем та машиною попереду. Фактично головна функція даного модуля – це перевірка чи можуть стикнутись дані динамічні об’єкти між собою на основі їхніх геометричних, фізичних характеристик та їх місцезнаходження у даний час взятих з відповідних таблиць;
- `behavioural_planner.py` – модуль для планування поведінки безпілотного автомобіля. Тут визначені основні переходи станів в програмно-побудованому скінченному автоматі (зупинитись на перехресті, слідувати машині, повернути направо та слідувати визначеній швидкості) на основі місцезнаходження безпілотного автомобіля в даний час та даних, які отримані в модулі перевірки зіткнення машин;
- `local_planner.py` – модуль для планування локалізації, який виконує функцію побудови гладкого маршруту для безпілотного автомобіля, без різких рухів та маневрів. Дані беруться з 3 попередніх модулів, а також, в залежності від моменту часу, вираховується кривизна руху машини за допомогою математичних пакетів NumPy та SciPy.

Все це разом складає планувальник руху автономного транспортного засобу, який при запуску симуляції належно працює матиме та надає функціональний стек планування руху, який дозволяє уникнути як статичних, так і динамічних перешкод під час відстеження центральної лінії смуги, а також обробляти знак зупинки та відповідний поворот на перехресті.



Рис. 3.7. Результат роботи планувальника маршруту безпілотного автомобіля у симуляторі

Кожен результат роботи головного модуля передається на контролер безпілотного автомобіля, який надається CARLA Simulator, і таким чином показує поведінку руху даного транспортного засобу у визначеній ситуації.

Отриманий результат можна поглянути на зображенні (рис. 3.7). Також в стандартному вікні присутні всі обраховані характеристики, як і для симулятора, так і для машини в цілому.

3.4.3 Опис інструкції користувача

Для відображення отриманих результатів потрібно завантажити CARLA Simulator з доступних open-source ресурсів (наприклад, Github або DockerHub). Середовище та ситуація, у якій знаходиться безпілотний автомобіль записується у бінарний файл PlanningProblemMap, тому його потрібно перемістити у папку /Game/Maps завантаженого симулятора. Для запуску карти потрібно ввести команду у відповідному CLI:

- Linux – ./CarlaUE4.sh /Game/Maps/Course4 -windowed -carla-server -benchmark -fps=30
- Windows – CarlaUE4.exe /Game/Maps/Course4 -windowed -carla-server -benchmark -fps=30

Як можна побачити, симулятор надає можливість налаштувати кількість FPS, що може відіграти велику роль, якщо машина, на якій запускається середовище, має недостатні обчислювальні характеристики.

Для запуску основного модуля планувальника потрібно ввести команду у відповідний CLI попередньо встановивши Python:

- Linux – python3 full_planner.py
- Windows – python full_planner.py

Також в даній команді є додатковий параметер plotting, який приймає на вхїж true/false, чия основна функція – це включення/виключення руху безпілотного автомобіля по графічній спіралі.

ВИСНОВКИ ДО РОЗДІЛУ

У третьому розділі були описані апаратна база та симуляційне середовище, які потрібні для побудови планувальника руху безпілотного автомобіля. Водночас були наведені основні технології використання при розробці програмного забезпечення, а також як вони пов'язані між собою. Були показані

					ІТ-62.25.1081.01 ПЗ	Арк. 69
Змн.	Арк.	№ докум.	Підпис	Дат.		

методи як зберегти автономність розробки планувальника та як отримана інформація з симуляції полегшує роботу у його побудові.

Водночас кожен із методів описуються в поетапному вигляді, та прописуються в окремих Python-модулях програмного забезпечення, які запускаються через свій основний модуль. В ньому також міститься основне підключення до бази даних та відповідні модулі реалізації DAO, для кожної із таблиць. БД є реляційною, тому кожна із таблиць пов'язана між собою та бере участь у всьому моделюванні.

Також було розглянуто CARLA Simulator, основні можливості даної програми та як вона може полегшити моделюванню, були описані основні вимоги для запуску та налаштування середовища. В кінці були показані результати роботи та наведена інструкція користувача.

					ІТ-62.25.1081.01 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дат.		

ВИСНОВКИ

У дипломному проекті була розроблена модель планування маршруту для безпілотного автомобіля у симуляційному середовищі. Була розглянута проблема автономності транспортних засобів, розглянуті основні компоненти програмної частини даного виду машин. Також був проведений аналіз існуючих рішень та на основі обраного прототипу було проведене порівняння та виділення основних переваг та недоліків можливої розробленої системи.

Беручи до уваги постановлену задачу було проведене детальне дослідження основних методів рішення проблеми та побудована ієрархічна модель, де описаний кожен етап, з яким так чи інакше стикається автомобіль під час планування автономного руху. Провівши пошук підходів для вирішення проблем кожного етапу, були обрані ті, які можуть найкраще підійти для кожного із рівнів та які можливо реалізувати за допомогою програмного коду. В результаті була конкретизована постановка задачі у покроковому вигляді.

Розглянуті методи вдалось перенести у програмне середовище та були описані основні моменти з точки зору розробки, а саме апаратна база та вибір симуляційного середовища, технологій для математичної обробки інформації та зберігання та оновлення даних. Була збережена автономність написання програмного коду для кожного із етапів планувальника та показано яким чином це можна інтегрувати в симуляцію.

Основним моментом є те, що дана методологія дозволяє показати проблему планування безпілотного автомобіля точково, при тому надана можливість додавати нові методи та оброблювати різні ситуації в залежності від того, з чим може стикнутись транспортний засіб на дорозі.

В ході виконання дипломної роботи побудовано модель планування руху безпілотного автомобіля у симуляційному середовищі, в якій написано програмний код для основних етапів, з якими стикається даний транспортний засіб.

					ІТ-62.25.1081.01 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дат.		71

ПЕРЕЛІК ПОСИЛАНЬ

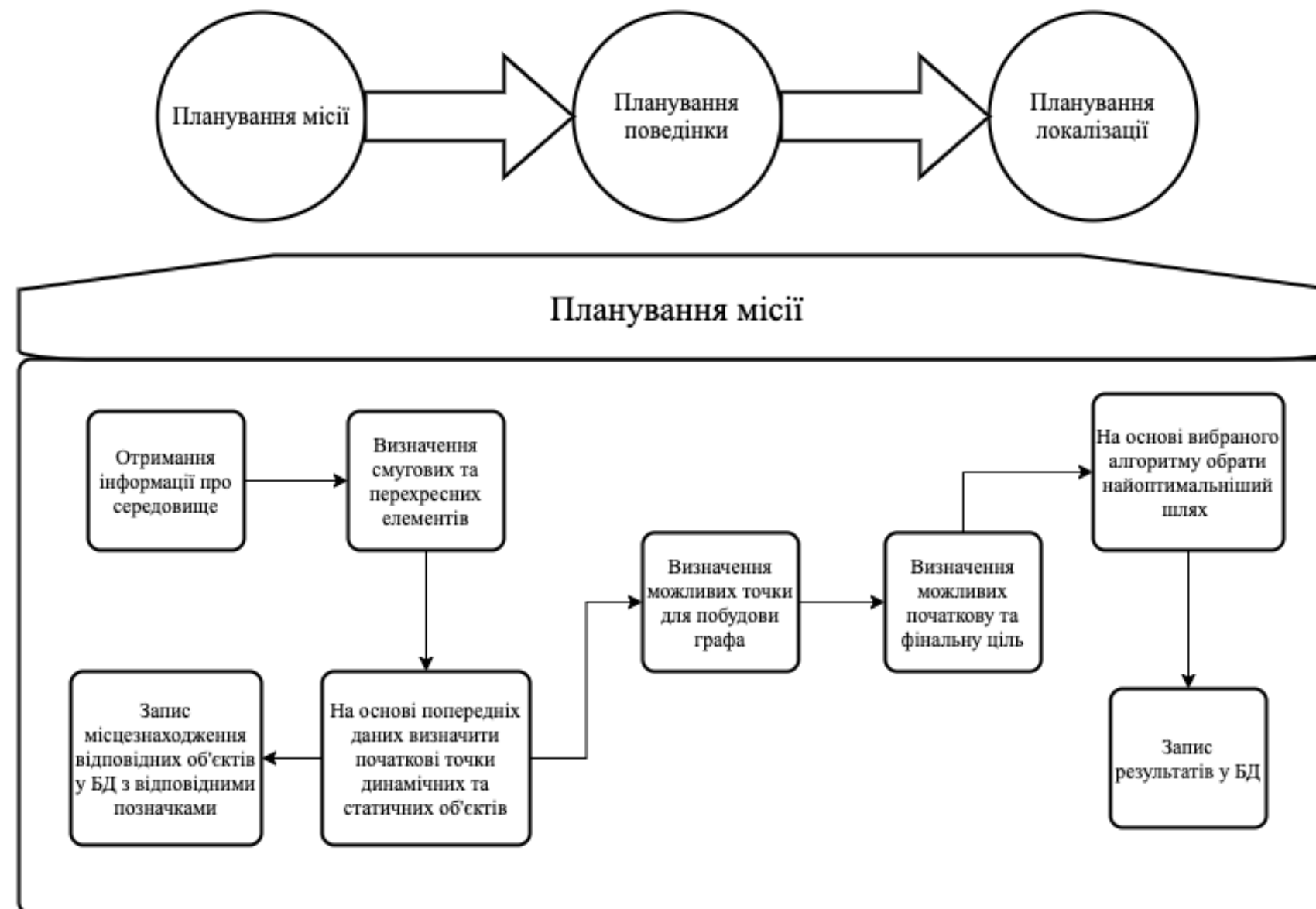
1. A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles [Електронний ресурс] – 2016. – Режим доступу до ресурсу: <https://ieeexplore.ieee.org/abstract/document/7490340>.
2. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions [Електронний ресурс] – 2015. – Режим доступу до ресурсу: <https://www.sciencedirect.com/science/article/pii/S0968090X15003447>
3. Self-Driving Cars: A Survey [Електронний ресурс] – 2019. – Режим доступу до ресурсу: <https://arxiv.org/abs/1901.04407>.
4. A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles [Електронний ресурс] – 2015. – Режим доступу: <https://www.sciencedirect.com/science/article/pii/S0950705115001604>.
5. Self-Driving Cars and the Urban Challenge [Електронний ресурс] – 2008. – Режим доступу до ресурсу: <https://ieeexplore.ieee.org/abstract/document/4475861>.
6. A Review of Motion Planning Techniques for Automated Vehicles [Електронний ресурс] – 2015. – Режим доступу до ресурсу: <https://ieeexplore.ieee.org/abstract/document/7339478>.
7. CARLA Simulator [Електронний ресурс] – 2018. – Режим доступу до ресурсу: <https://carla.org/>.
8. Become a Self-Driving Car Engineer – Udacity [Електронний ресурс] – 2018. – Режим доступу до ресурсу: <https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd013>.
9. Real-Time Motion Planning With Applications to Autonomous Urban Driving [Електронний ресурс] – 2009. – Режим доступу до ресурсу: <https://ieeexplore.ieee.org/abstract/document/5175292>

					ІТ-62.25.1081.01 ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дат.		

ДОДАТКИ

Деталізація ієрархічної моделі планувальника: Етап №1

Розбиття моделі планування маршруту для безпілотного автомобіля на поетапну ієрархію

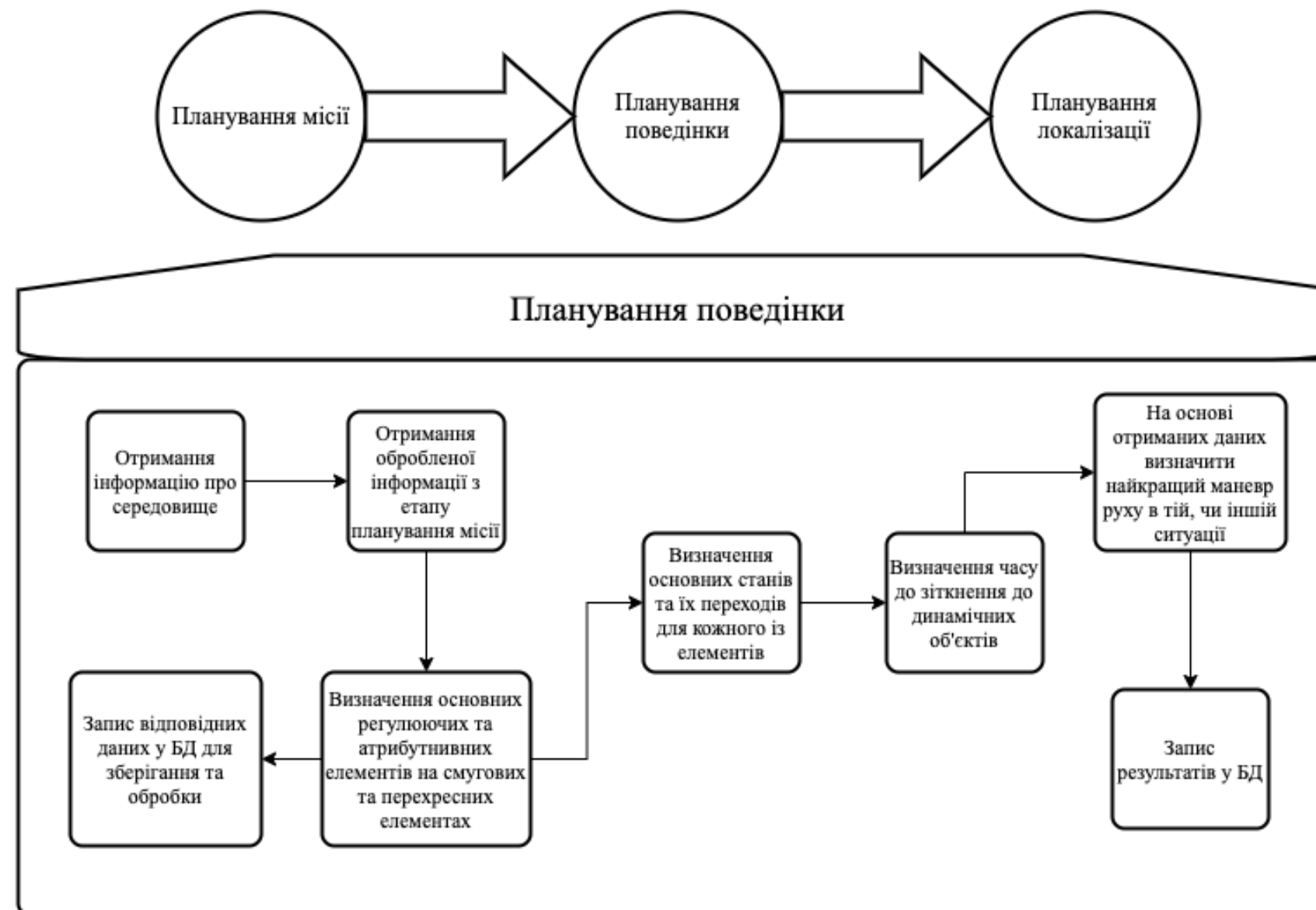


Демонстраційний плакат №1
До дипломної роботи на тему
„Модель системи побудови маршруту для безпілотних автомобілів“

Розробив: студент гр. ІТ-62, Сенишин Б.В.
Прийняв: асистент, Базака Ю.А.

Деталізація ієрархічної моделі планувальника: Етап №2

Розбиття моделі планування маршруту для безпілотного автомобіля на поетапну ієрархію

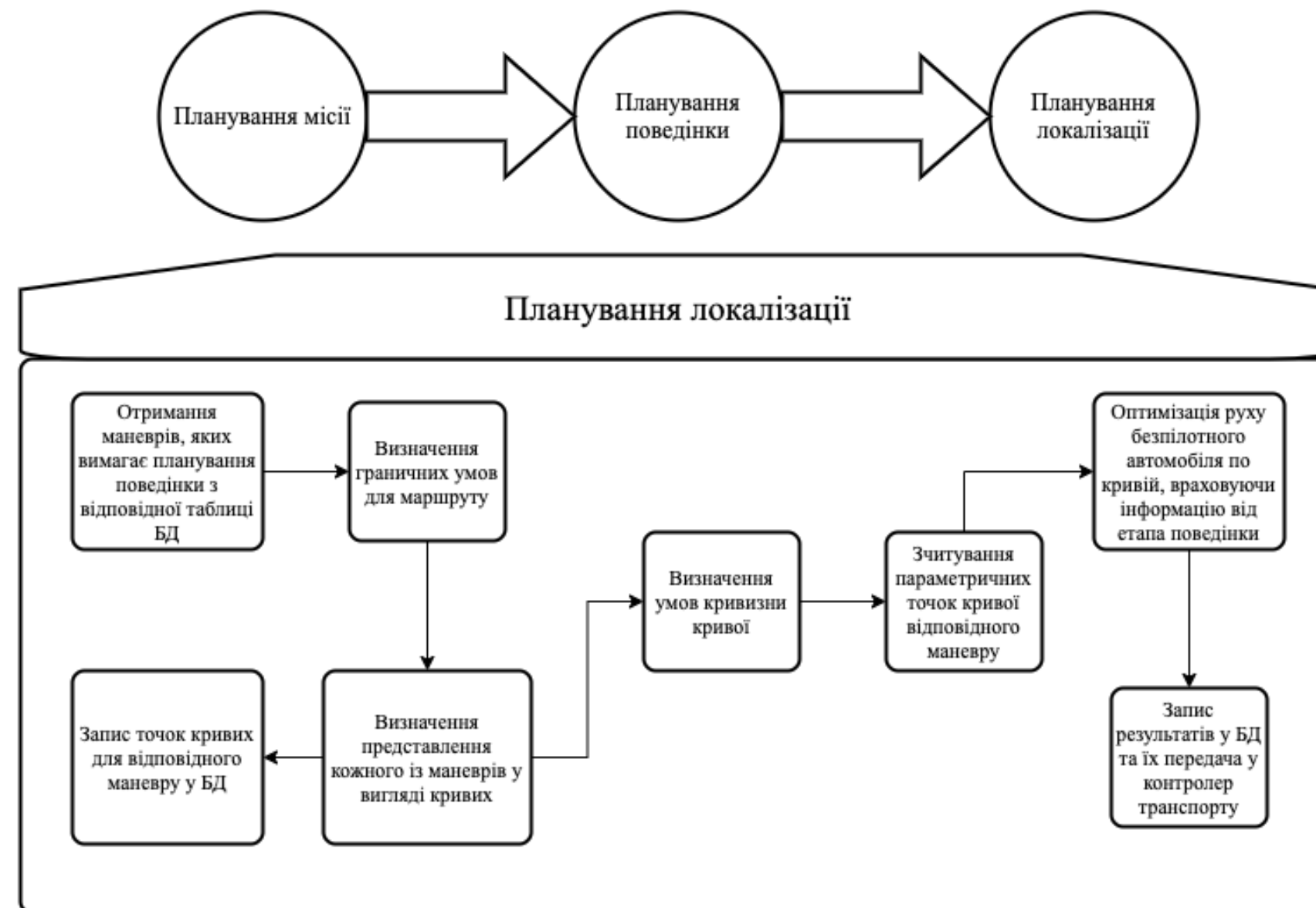


Демонстраційний плакат №2
До дипломної роботи на тему
„Модель системи побудови маршруту для безпілотних автомобілів“

Розробив: студент гр. ІТ-62, Сенишин Б.В.
Прийняв: асистент, Базака Ю.А.

Деталізація ієрархічної моделі планувальника: Етап №3

Розбиття моделі планування маршруту для безпілотного автомобіля на поетапну ієрархію



Демонстраційний плакат №3
До дипломної роботи на тему
„Модель системи побудови маршруту для безпілотних автомобілів“

Розробив: студент гр. ІТ-62, Сенишин Б.В.

Прийняв: асистент, Базака Ю.А.

Власник документу:
Лісовиченко Олег Іванович

ID перевірки:
1003791064

Дата перевірки:
04.06.2020 22:51:32 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
04.06.2020 23:03:36 EEST

ID користувача:
76913

Назва документу: IT-62_Сенишин_OI

ID файлу: 1003804973 Кількість сторінок: 51 Кількість слів: 11348 Кількість символів: 83077 Розмір файлу: 506.19 KB

1.41% Схожість

Найбільша схожість: 0.65% з джерело бібліотеки. ID файлу: 1000070056

Не знайдено жодних джерел з Інтернету

1.41% Текстові збіги по Бібліотеці акаунту

8

Page 53

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів

1