МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Інтегрована система для автономного маршрутного планування і навігації у транспортних системах

Спеціальність 123 Комп'ютерна інженерія

123 – КМР.ПЗ.00 – 605.21810511

Студент _____М. О. Керекеслер

підпис

«__» _____ 202__ р.

Керівник канд. техн. наук, доцент _____ Я.М. Крайник

підпис

«__» _____ 202__ р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри _____ І. М. Журавська

«_____» _____ 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної магістерської роботи

Видано студенту групи 605 факультету комп'ютерних наук

Керекеслеру Максим Олеговичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи
Інтегрована система для автономного маршрутного планування і навігації у транспортних системах

Затверджена наказом по ЧНУ ім. Петра Могили від 08.11.2023 № 226.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваним результатом роботи є: апаратне та програмне забезпечення розпізнавання об'єктів, прокладання маршруту оминаючи перешкоди.

4. Перелік питань, що підлягають розробці
1) огляд технологій для системи з автономним маршрутним плануванням і навігацією;
2) налаштування та підключення сенсорів;
3) розробка апаратної частини системи автономного маршрутного планування та навігації;
4) розробка програмної частини системи автономного маршрутного планування та навігації;

5. Перелік графічних матеріалів

Зображення сенсорів, обчислювача, всієї платформи

6. Завдання до спеціальної частини _

Розглянути основні державні норми України, щодо праці в умовах використання комп’ютерів та екраних пристроїв загалом, щодо вентиляції та кондиціонування, організація повітрообміну, норм шумів та вібрацій. Ознайомитись з правами та нормами праці в умовах використання комп’ютерів та екраних пристроїв загалом.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Д-р біол. наук, професорка Григор'єва Л. І.	Кафедра екології Навчально-наукового медичного інституту ЧНУ ім. Петра Могили	Спеціальна частина з охорони праці та безпеки у надзвичайних ситуаціях

Керівник роботи
_канд. техн. наук, доцент Крайник Ярослав Михайлович_____
(посада, прізвище, ім’я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

_Керекеслер Максим Олегович_____
(прізвище, ім’я, по батькові студента)

_____
(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної магістерської роботи

Тема:_ Інтегрована система для автономного маршрутного планування і навігації у транспортних системах

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КМР	01.09.2023	15.09.2023	*Виконано*
2.	Огляд літератури за темою роботи	15.09.2023	01.10.2023	*Виконано*
3.	Складання календарного плану КМР	01.10.2023	03.10.2023	*Виконано*
4.	Аналіз предметної області	05.10.2023	25.10.2023	*Виконано*
5.	Розробка проєктних рішень	25.10.2023	10.11.2023	*Виконано*
6.	Моделювання системи	10.11.2023	15.11.2023	*Виконано*
7.	Робота з апаратним забезпеченням	15.11.2023	20.12.2023	*Виконано*
8.	Створення програмного забезпечення	20.12.2023	05.01.2024	*Виконано*
9.	Тестування та аналіз	05.01.2024	10.01.2024	*Виконано*
10.	Розробка віддаленого керування	10.01.2024	13.01.2024	*Виконано*
11.	Відгук керівника КМР	13.01.2024	05.02.2024	*Виконано*
12.	Оформлення КМР та презентації	07.02.2024	08.02.2024	*Виконано*
13.	Попередній захист	31.01.2024	11.02.2024	*Виконано*
14.	Рецензування	12.02.2024	18.02.2024	*Виконано*
15.	Захист кваліфікаційної роботи	27.02.2024	27.02.2024	*Виконано*

Розробив здобувач ВО Керекеслер Максим Олегович_____

(*прізвище, ім'я, по батькові)* *(підпис)*

«__» _____ 20__ р.

Керівник роботи Крайник Ярослав Михайлович____ _____

(підпис)

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної магістерської роботи

«Інтегрована система для автономного маршрутного планування і навігації у транспортних системах»

Студент гр. 605 Керекеслер Максим Олегович

Керівник: канд. техн. наук, доцент Крайник Ярослав Михайлович

Кваліфікаційна магістерська робота присвячена розробці системи для автономного маршрутного планування і навігації в транспортних системах. Розглянуто наявні технології для автономної навігації та алгоритми будування. Практичне значення результатів дослідження та розроблення полягає в тому що, на підставі результатів можливе вдосконалення апаратно-програмного комплексу, у засвоєнні навичок процесу проектування, розробки, створення приладу, можливостях застосування розробленого програмно-апаратного комплексу.

Пояснювальна записка кваліфікаційної магістерської роботи складається зі вступу, трьох розділів, висновків та додатків. У вступі визначається актуальність теми, сформульовані мета, об'єкт, предмет та завдання дослідження та розроблення кваліфікаційної магістерської роботи. У першому розділі проводиться аналіз існуючих технологій для автономного маршрутного планування і навігації та обираються необхідні компоненти. У другому розділі розглянуто принцип роботи кожного елементу системи та принцип роботи всієї системи. У третьому розділі описано розробку програмного забезпечення системи з автономним маршрутним плануванням і навігацією. У четвертому розділі проаналізовано результати роботи системи та їх компонентів. У висновках наведено аналіз виконаної роботи та отриманих результатів дослідження та розроблення.

У додатках А та Б наведено програмний код, що використовувався в проекті. В цілому, кваліфікаційна магістерська робота без додатків містить 78 сторінок, 23 рисунка, 27 джерел посилання.

ASTRACT

of the Master's Thesis

" Integrated system for autonomous route planning and navigation in transport systems "

Student 605 gr.: Kerekesler Maxim Olegovich

Supervisor: candidate of technical sciences, associate professor Kraynyk Yaroslav Mykhailovych

The qualification master's thesis is devoted to the development of a system for autonomous route planning and navigation in transport systems. Available technologies for autonomous navigation and building algorithms are considered. The practical significance of the results of research and development is that, based on the results, it is possible to improve the hardware and software complex, in learning the skills of the process of design, development, creation of the device, and the possibilities of using the developed software and hardware complex.

The explanatory note of the qualifying master's thesis consists of an introduction, three sections, conclusions and appendices. The introduction determines the relevance of the topic, formulates the goal, object, subject and task of research and development of a qualifying master's thesis. The first section analyzes the existing technologies for autonomous route planning and navigation and selects the necessary components. The second chapter examines the principle of operation of each element of the system and the principle of operation of the entire system. The third chapter describes the software development of the system with autonomous route planning and navigation. The fourth chapter analyzes the results of the system and its components. The conclusions provide an analysis of the work performed and the obtained results of research and development.

Appendices A and B provide the program code used in the project. In general, a qualification master's thesis without appendices contains 78 pages, 23 figures, 27 reference sources.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

GPS	–	Global Positioning System
GPIO	–	General-Purpose Input/Output
USB	–	Universal Serial Bus
RTK	–	Real-Time Kinematic
HDMI	–	High Definition Multimedia Interface
API	–	Application Programming Interface
SRR	–	Short Range Radar
RPLIDAR	–	Rotary Planar Light Detection and Ranging
ARM	–	Advanced RISC Machine
SSD	–	Single Shot Multibox Detector
YOLO	–	You Only Look Once
CNN	–	Convolutional Neural Network
CUDA	–	Compute Unified Device Architecture
GPU	–	Graphics Processing Unit

ВСТУП

Сучасний швидкий темп життя та постійний розвиток технологій вимагають вдосконалення систем автономного маршрутного планування і навігації для мобільної робототехніки. Особливо актуальним стає застосування інтегрованих систем, що базуються на штучному інтелекті, для ефективного управління рухом автономних роботів, таких як роботи-кур'єри та інші. Інтегровані системи не лише забезпечують оптимальне планування маршрутів, але й здатні адаптуватися до змінних умов навколишнього середовища, підвищуючи ефективність та безпеку руху.

Однією з ключових галузей застосування таких систем є мобільна робототехніка, де роботи-кур'єри виконують важливі завдання, такі як доставка товарів та послуг. Використання штучного інтелекту в цьому контексті дозволяє роботам ефективно взаємодіяти з навколишнім середовищем, уникати перешкод, визначати оптимальні маршрути та швидше адаптуватися до змінних ситуацій.

Автономність системи в контексті автономної навігації визначається її здатністю приймати рішення та виконувати дії без прямого втручання людини. Це означає, що система може самостійно визначати своє місцезнаходження, планувати маршрут, уникати перешкод та реагувати на зміни у середовищі.

У даній роботі розглядається розробка та вдосконалення інтегрованої системи для автономного маршрутного планування та навігації на основі штучного інтелекту для мобільної робототехніки, а також вирішення зазначених вище проблем. Результати цього дослідження відкриють нові можливості для розвитку автономних систем, що забезпечать ефективність та безпеку їхньої діяльності в сучасному світі.

Мета: Розробка інтегрованої системи для автономного маршрутного планування та навігації на базі штучного інтелекту для мобільної робототехніки, що дозволить створення ефективної, безпечної та адаптивної

системи, яка забезпечить оптимальне функціонування автономних роботів, зокрема робіт-кур'єрів, у різноманітних умовах.

Об’єкт: Технології автономної навігації безпілотних наземних транспортних засобів.

Предмет: Система автономної навігації транспортних засобів.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– Проаналізувати сучасні технології автономної навігації;

– Розробити прогруму для управління мікрокомп’ютером Jetson Nano, зокрема управління двигунами для пересування платформи та отримання зображення з камери;

– Створення ефективних алгоритмів, які дозволять знаходити найоптимальніші маршрути в умовах змінного середовища та об'єктів;

– Розробити застосунок для побудови маршруту до цілі на основі інформації місцезнаходження отриманого з gps-модуля;

– Розрахувати живлення всіх компонентів та визначити скільки часу буде робити система.

Практичне значення:

Розроблена система може бути використана в сучасних сервісах доставки та логістиці для підвищення ефективності та конкурентоспроможності. Крім того, вирішення проблем безпеки та оптимізація маршрутів можуть сприяти розвитку та впровадженню автономних роботів у різноманітних сферах, що вимагають автономних транспортних засобів.

Робота пройшла **апробацію** на Всеукраїнській науково-практичній конференції молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія»(31 січня – 2 лютого 2024 р. / ЧНУ імені Петра Могили. Миколаїв) [3].

Публікації: Основні положення та результати роботи було опубліковано у збірнику тез конференції «Інформаційні технології та інженерія» [3].

1 АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА СКЛАДОВИХ СИСТЕМИ

1.1 Сучасні системи автономної навігації

Автономне маршрутне планування і навігація в транспортних системах є актуальною та інноваційною галуззю, яка використовує передові технології для покращення руху транспортних засобів та забезпечення ефективності та безпеки на дорогах.

Існує кілька аналогів та конкуруючих систем для автономного маршрутного планування та навігації в транспортних системах: Waymo, Tesla Autopilot, Uber ATG, Baidu Apollo, Mobileye.

Waymo є проєктом компанії Alphabet Inc. (материнської компанії Google), спрямованим на розробку та впровадження технологій автономного водіння. Однією з ключових складових системи Waymo є датчик LiDAR, який створює точні тривимірні карти навколишнього середовища. Waymo використовує алгоритми машинного навчання для розпізнавання об'єктів, педестріанів, інших транспортних засобів та взаємодії з ними. Система сприйняття Waymo Driver приймає складні дані, накопичені з провідного набору датчиків, і розуміє все навколо, будь то інші транспортні засоби до пішоходів або велосипедистів, використовуючи такі технології, як машинне навчання. У той же час він також звертає увагу на сигнали та знаки, такі як знаки зупинки або перемикання кольорів світлофорів. Машинне навчання дозволяє автомобілям орієнтуватися в складних і делікатних ситуаціях, таких як робота в зонах будівництва, поступка автомобілям екстрених служб і надання місця автомобілям для паралельного паркування[5].

Tesla Autopilot є системою автопілоту для електромобілів Tesla, яка використовує комбінацію камер, радарів та сенсорів для навігації транспортного засобу. Ця система використовує нейронні мережі для розпізнавання дорожніх знаків, інших автомобілів, а також для контролю руху в межах смуги та виконання маневрів. Tesla Autopilot використовує принципи

машинного навчання для адаптації до різних дорожніх умов. Будь-яка Tesla, побудована в 2016 році або пізніше, має все необхідне обладнання, необхідне для автопілота. З 2016 року до кінця 2021 року це означає, що автомобіль оснащений 8 камерами, 12 ультразвуковими датчиками, інструментами для обробки зору, бортовим комп'ютером тощо. Починаючи з 2021 року, Tesla почала видаляти ультразвукові датчики з Model 3 і Model Y, а в 2022 році перехід від датчиків був завершений шляхом видалення їх з Model X і Model S. Замість того, щоб використовувати датчики, схожі на радари, Tesla повністю покладається на камери автомобілів для вимірювання відстані. Навіть без ультразвукових датчиків можна використовувати функції автопілота Tesla, аж до повного автономного водіння[4].

Uber ATG (Advanced Technologies Group) вивчає та розробляє технології для автономного транспорту та таксі. Система Uber ATG використовує лідари, камери та радари для отримання даних про оточуюче середовище. Багато компонентів безпілотних транспортних засобів використовують моделі машинного навчання, що дозволяє їм керувати автомобілем безпечно та точно. Компонент складається з однієї або декількох моделей машинного навчання, і всі компоненти разом утворюють програмне забезпечення для безпілотних транспортних засобів: сприйняття, передбачення, планування руху, керування. Компонент «сприйняття» використовує інформацію з датчиків автомобіля для виявлення акторів у певній сцені за допомогою моделей машинного навчання. Компонент «передбачення» використовує вихідні дані компонента «сприйняття» (тип актора та 3D-координати всіх акторів у сцені), а також карти високої чіткості, щоб передбачити майбутні траєкторії акторів протягом n секунд у заданій сцені за допомогою моделей машинного навчання. Компонент «планування руху» використовує пункт призначення безпілотного транспортного засобу, передбачені траєкторії всіх дійових осіб у сцені, карту високої чіткості та інші механізми для планування шляху транспортного засобу. Компонент «керування» керує колесами безпілотного автомобіля та керує гальмами та акселератором, щоб слідувати шляху,

створеному компонентом планування руху[6].

Apollo, розроблений китайською компанією Baidu, є відкритою платформою для автономних транспортних систем. Apollo використовує сенсори, включаючи лідари та камери, для збору даних про дорожнє середовище. Алгоритми Apollo використовуються для обробки цих даних, визначення оптимального маршруту та керування транспортним засобом[7].

Mobileye, що належить Intel, спеціалізується на системах візуального розпізнавання для автономних транспортних засобів. Вони використовують камери та комп'ютерне зорове розпізнавання для аналізу дорожнього руху, виявлення об'єктів та безпечного керування. Mobileye також використовує алгоритми глибокого навчання для реалізації своїх функціональних можливостей[8].

Ці системи представляють різні технічні підходи та інноваційні рішення для автономного маршрутного планування та навігації. Кожна з них використовує комбінацію сенсорів, алгоритмів машинного навчання та технологій відображення, щоб надавати транспортним засобам здатність ефективно та безпечно функціонувати в реальному світі.

Ці системи оснащені дорогими датчиками та мікрокомп’ютерами. Компанії частіше всього самі розробляють або замовляють різні компоненти, які необхідні для роботи таких систем.

1.2 Навігація в транспортних системах

Системи для самостійного планування маршруту та навігації в транспортних системах в основному ґрунтуються на використанні різноманітних сенсорів, алгоритмів обробки даних та штучного інтелекту для створення безпечного та ефективного руху транспортних засобів. Це особливо актуально в контексті розвитку автономних транспортних засобів (автомобілі, дрони, роботи) та систем громадського транспорту.

Автономна система для маршрутного планування та навігації в транспортних системах повинна відповідати численним вимогам для

ефективної та безпечної роботи.

Основні складові та вимоги систем автономного маршрутного планування і навігації включають:

1) сенсори: Використовуються різноманітні сенсори, такі як камери, лідари, радари, GPS, інерціальні датчики і т.д., для отримання інформації про оточуюче середовище та точного визначення положення транспортного засобу;

2) мапи: Величезна роль у визначенні маршруту та навігації грає використання високодеталізованих цифрових карт, які містять інформацію про дороги, об'єкти, світлофори та інші важливі елементи інфраструктури;

3) обробка даних: Великі обсяги даних від сенсорів обробляються за допомогою алгоритмів машинного навчання та комп'ютерного зору. Це дозволяє системі «розуміти» оточуючий світ, виявляти об'єкти, розпізнавати дорожні знаки та інші елементи інфраструктури;

4) планування маршруту: Алгоритми планування враховують різні фактори, такі як дорожні умови, вартість енергії для електромобілів, безпека та інші обмеження. Мета – забезпечити оптимальний та безпечний маршрут від точки А до точки Б.

5) керування транспортним засобом: Системи автономного транспорту використовують алгоритми керування для ефективного використання рульового управління, гальм та газу на основі інформації, отриманої від сенсорів та обробленої алгоритмами;

6) комунікація з інфраструктурою та іншими транспортними засобами: Системи можуть використовувати комунікаційні протоколи для обміну інформацією з іншими автономними транспортними засобами та інфраструктурою (наприклад, інтелектуальними світлофорами).

Зростання інтересу до автономного транспорту викликає необхідність подальшого вдосконалення цих систем, вирішення питань щодо безпеки, регулювання та інших аспектів.

Системи автономного маршрутного планування та навігації нерідко

використовують системи високоточного позиціонування, які включають в себе не лише GPS, але і інші технології, такі як RTK (Real-Time Kinematic) та DGPS (Differential GPS). Ці технології дозволяють отримувати точне положення транспортного засобу з високою частотою оновлення, що є важливим для точної навігації, особливо в умовах міського середовища[9].

У контексті автономних транспортних систем важливою стає співпраця з автоматизованими системами управління трафіком. Інтеграція з інтелектуальними світлофорами, системами моніторингу дорожньої ситуації та автоматизованими диспетчерськими службами дозволяє оптимізувати рух транспорту, зменшуючи затори та підвищуючи загальну ефективність транспортної системи.

З уведенням автономного транспорту на передовий план виходять питання стосовно енергоефективності та екологічної придатності. Автономні електричні транспортні засоби розглядаються як один із шляхів зменшення викидів CO2 та інших шкідливих речовин. Алгоритми автономного керування можуть оптимізувати шляхи з урахуванням енергетичної ефективності та розподілу навантаження на акумуляторні батареї.

Запровадження автономного транспорту має широкі соціальні та економічні наслідки. Зменшення кількості аварій, поліпшення доступності транспорту для людей з обмеженими можливостями, а також зміна підходів до паркування та використання транспортних інфраструктур - це лише деякі з аспектів, які можуть значно покращити якість життя та рівень безпеки.

Системи для автономного маршрутного планування і навігації є складними інтегрованими системами, які об'єднують аспекти технологій сенсорів, алгоритмів машинного навчання, інформаційних технологій та інфраструктури. Ці системи мають потенціал трансформувати транспортну індустрію, поліпшуючи безпеку, зручність та стійкість до навколишнього середовища.

1.3 Мікрокомп’ютер

Для управління системою автономної навігації потрібен мікрокомп’ютер. Мікрокомп'ютер відповідає за швидку та ефективну обробку сенсорних даних, отриманих від камер, лідарів, радарів та інших датчиків в режимі реального часу. Це важливо для надання актуальної інформації системі маршрутного планування та навігації. Багато алгоритмів машинного навчання вимагають значної обчислювальної потужності для розпізнавання об'єктів, класифікації сценаріїв та прийняття рішень. Мікрокомп'ютер забезпечує виконання цих алгоритмів. Він відіграє роль у керуванні енергоспоживанням автономної системи, оптимізуючи роботу сенсорів, алгоритмів та інших компонентів для максимізації автономії транспортного засобу.

Узагальнюючи, мікрокомп'ютер в цій системі є мозком, який обробляє інформацію, приймає рішення та забезпечує взаємодію з іншими компонентами для досягнення ефективного та безпечного автономного руху.

Для створення системи автономного маршрутного планування та навігації було обрано платформу JetRacer від компанії Nvidia, яка містить модуль Jetson Nano та камеру. Ця платформа створена для розробки багато різних енергоефективних систем на базі ШІ, а також забезпечує багатьма різними можливостями.

Jetson Nano - це мініатюрний одноплатний комп'ютер, розроблений компанією NVIDIA спеціально для застосувань у сфері штучного інтелекту (ШІ) та робототехніки. Основним призначенням Jetson Nano є обробка великої кількості даних в реальному часі і виконання складних алгоритмів ШІ. Розмір модуля Jetson Nano менший, ніж у кредитної картки, становлячи 70 х 45 мм. Jetson Nano забезпечує 472 гігафлопси для швидкої обробки алгоритмів штучного інтелекту, дозволяючи паралельно запускати кілька нейронних мереж і обробляти інформацію від різних високоточних датчиків. Компактний розмір та низьке енергоспоживання роблять Jetson Nano ідеальним для вбудованих систем та мобільних пристроїв[10].

Рис. 1.1 – Мікрокомп’ютер Jetson Nano

Jetson Nano має такі характеристики:

– Графічний процесор: 128-ядерний графічний процесор на основі архітектури NVIDIA Maxwell.

– Процесор: чотирьохядерний ARM A57.

– Пам'ять: 4 ГБ 64-розрядної LPDDR4; 25,6 гігабайт/сек.

– Підтримка ОС: Linux для Tegra.

– Розмір модуля: 70 мм x 45 мм.

Jetson Nano має вбудований вентилятор для забезпечення оптимальної температури пристрою при високих навантаженнях.

Також є різноманітні порти, включаючи HDMI, USB 3.0, USB 2.0, Ethernet та GPIO, що дозволяє підключати різні пристрої та сенсори.

Завдяки високій обчислювальній потужності та компактному дизайну Jetson Nano стає ідеальним вибором для різних застосувань, включаючи системи автономного водіння, робототехніку та візуальне визначення.

Jetson Nano відмінно підходить для використання в різних типах транспортних засобів завдяки своїй компактності та низькому енергоспоживанню. Він може бути легко вбудований у сучасні автомобілі, роботи-кур'єри або інші транспортні системи без значного впливу на їх дизайн чи продуктивність. Jetson Nano підтримує різні розширювані модулі та

інтерфейси, що дозволяє легко змінювати функціональні можливості системи та адаптувати її до різних потреб та вимог користувачів.

Система може бути підключена до системи управління автомобілем через стандартні інтерфейси, такі як CAN (Controller Area Network) або LIN (Local Interconnect Network), що дозволяє обмінюватися даними та командами з іншими системами автомобіля, наприклад, системами управління двигуном, системами стабілізації або системами безпеки.

1.4 Сенсори

Системи автономного маршрутного планування та навігації використовують широкий спектр сенсорів для збору величезної кількості інформації про навколишнє середовище. Ці сенсори є основними будівельними блоками, які дозволяють транспортному засобу ефективно сприймати та взаємодіяти з оточуючим світом.

Системи автономного маршрутного планування та навігації широко використовують різноманітні сенсори для отримання інформації про навколишнє середовище. Камери використовуються для візуального сприйняття, лідари для вимірювання відстаней та створення точного тривимірного зображення оточуючого простору, а радари для виявлення об'єктів за перешкодами та в різних погодних умовах. GPS забезпечує глобальне визначення положення, а інерціальні датчики фіксують зміни в русі.

Камери грають важливу роль у візуальному сприйнятті оточуючого середовища. Високороздільна камера дозволяє системі отримувати зображення дорожнього середовища, розпізнавати дорожні знаки, ідентифікувати інші транспортні засоби та визначати рух пішоходів. Алгоритми комп'ютерного зору використовуються для обробки зображень і визначення об'єктів та їхнього місцезнаходження.

Вибір конкретного типу камери залежить від конкретних потреб системи, умов експлуатації та вимог до точності. У багатосенсорних системах часто використовується комбінація різних типів камер для максимальної

ефективності та робочого діапазону в різних умовах.

В комлекті JetRacer є CSI-камера IMX219. Вона має 800 мегапікселів і кут огляду 160. Ця камера сумісна с Jetson Nano і підходить для отримання зображення для подальшої обробки штучним інтелектом.

Лідари (Light Detection and Ranging) грають ключову роль у системах автономного транспорту, забезпечуючи високоточну тривимірну інформацію про оточуюче середовище. Лідари розрізняються за рядом характеристик, таких як тип лазера, кількість та розташування детекторів, кут огляду та швидкість обробки даних.

Лідар працює на основі використання лазерного випромінювання для вимірювання відстані до об'єктів та створення тривимірної карти оточуючого простору. Основними елементами лідара є лазер, оптична система та детектор. Основні етапи роботи лідара:

1) випромінювання лазера: лідар використовує лазер для випромінювання коротких імпульсів світла в різних напрямках;

2) відбиття від об'єкта: світлові промені стикаються з об'єктами в оточуючому середовищі, і частина світла відбивається від цих об'єктів;

3) прийом сигналу: детектор лідара отримує відбите від об'єкта світло. Час, який затрачає світловий сигнал на шлях від лідара до об'єкта і назад, фіксується;

4) вимірювання часу польоту: лідар вимірює час, який затрачає світловий імпульс на те, щоб повернутися в лідар після відбиття від об'єкта. Цей час вимірюється з високою точністю;

5) розрахунок відстані: відстань до об'єкта обчислюється, використовуючи відомий час польоту світлового імпульсу і знаючи швидкість світла;

6) формування хмари точок: лідар випромінює лазерні промені в різних напрямках, отримуючи відстані до багатьох об'єктів у своєму полі зору. За допомогою цих відстаней та відомого положення лідара формується хмара точок, що представляє тривимірну карту оточуючого простору;

7) обробка та аналіз даних: отримані дані можуть бути оброблені та використані для різних цілей, таких як визначення глибини, розпізнавання об'єктів, створення карт висот та інших додаткових інформацій.

Лідари можуть бути встановлені на різних транспортних засобах, таких як автомобілі, дрони або роботи. Вони дозволяють цим системам отримувати високоточні та деталізовані дані про навколишнє середовище, що є важливим для автономного маршрутного планування та навігації.

Вибір конкретного лідару залежить від потреб конкретної системи, обмежень бюджету та вимог до точності та швидкодії. Багатосенсорні системи часто використовують комбінації різних типів лідарів для оптимальної продуктивності в різних сценаріях.

Механічні лідари використовують обертовий механізм для спрямовування лазерного променя в різні напрямки. Це може бути виконано за допомогою обертового дзеркала або обертової призми. Недоліком таких систем є обмежена швидкість оновлення та можливість виникнення механічних поломок. Однак вони можуть забезпечити великий кут огляду.

Статичні (Фіксовані) Лідари не мають рухомих частин і використовуються для статичних застосувань. Хоча вони мають обмежений кут огляду, вони відрізняються високою точністю і низькими вимогами до обслуговування. Ці лідари часто встановлюються на транспортних засобах для постійного моніторингу навколишнього середовища.

Лідари з кількома лазерами використовують кілька лазерів, які видають промені в різних напрямках. Це дозволяє отримати більше інформації за один оборот та забезпечити більш швидке сканування оточуючого простору.

Фазові лідари вимірюють фазу поверненого лазерного променя, що дозволяє отримувати високу точність вимірювань величини та відстані. Це робить їх особливо ефективними для застосувань, де важлива висока точність даних, таких як картографія або визначення точного положення транспортного засобу.

Лідари з Точковим Детектором (Solid-State LiDAR) - це нове покоління

лідарів без рухомих частин, які використовують жваві точкові детектори. Вони можуть забезпечити високу швидкість сканування та надійність. Спрямовані на ринок масового виробництва, вони мають потенціал знизити вартість і підвищити доступність технології.

Для такої системи можна один із таких лідарів:

Slamtec RPLIDAR A3 360° Laser Scanner має дальність визначення об’єкту 25 м, що дозволить збирати більше даних про навколишнє середовище за рахунок більшої контурної карти. Частота сканування: 10 – 20 Гц. RPLIDAR A3 використовує діапазон, який обертається за годинниковою стрілкою, що дозволяє повністю сканувати навколишнє середовище на 360° і створювати карту місцевості. Частота дискретизації LIDAR безпосередньо визначає, чи зможе робот швидко і точно картографувати. RPLIDAR A3 покращує внутрішню оптичну конструкцію та систему алгоритмів, щоб збільшити частоту дискретизації до 16000 разів/секунду. RPLIDAR A3 має можливість працювати одночасно у двох режимах: посиленому режимі та зовнішньому режимі. У розширеному режимі RPLIDAR A3 використовує максимальний радіус дальності та частоту дискретизації для досягнення оптимальної продуктивності картографування в приміщеннях. У зовнішньому режимі RPLIDAR A3 працює з більш надійною стійкістю до перешкод денного світла. Як білі, так і чорні об'єкти виявляються з однаковою стабільною ефективністю. RPLIDAR A3 використовує принцип дальності лазерної тріангуляції, а завдяки високошвидкісному далекоміру RPVISION 3.0 він вимірює дані про відстань 16000 разів на секунду та зберігає свою чудову продуктивність на великій відстані. RPLIDAR A3 зберігає свою виняткову продуктивність при товщині 4 см компактного розміру. Він ідеально підходить для всіх видів сервісних роботів[12].

Рис. 1.2 – Slamtec RPLIDAR A3

YDLIDAR G2 360° Laser Scanner. Має частоту адаптивного сканування 5-12 Гц. Використовуючи високоякісні підшипники та індивідуальний безщітковий двигун, лідар має стабільну роботу, що значно подовжує термін служби лідара. Передача даних за допомогою послідовного порту рівня 3,3 В (UART) можна з'єднати зовнішню систему та виріб через фізичний інтерфейс. Його висота всього 41 мм. Інтегрована система приводу та система обертання роблять лідар серії G2 більш точним. Лідар має дальність дії 12 м, що значно менше за попередній варіант.

Рис. 1.3 – YDLIDAR G2

Лідари в системах автономного транспорту є невід'ємною частиною для забезпечення точного та швидкого збору даних про навколишнє середовище. Інновації в цій галузі спрямовані на підвищення точності, зменшення вартості та підвищення доступності для широкого спектру застосувань в автономних

транспортних системах.

Радари використовують радіохвильові сигнали для виявлення об'єктів та визначення їхньої відстані та швидкості руху. Вони ефективні в умовах обмеженої видимості та погіршених погодних умов. Радари можуть виявляти та відстежувати інші транспортні засоби, пішоходів та навіть визначати ступінь напруженості дорожнього руху.

Радари в системах автономного транспорту використовуються для виявлення об'єктів, визначення їхньої відстані та швидкості руху. Різноманітні типи радарів можуть бути використані в залежності від конкретних завдань та умов експлуатації. Вибір конкретного радару залежить від потреб конкретної системи, таких як діапазони роботи, точність, частотні характеристики та здатність виявляти об'єкти в різних умовах.

Коротко-діапазонні Радари (Short-Range Radar – SRR) оптимальні для виявлення об'єктів на невеликих відстанях, таких як при паркуванні або русі в умовах міського трафіку. SRR може використовуватися для систем допомоги при паркуванні, системи уникнення зіткнень на невеликих швидкостях тощо.

Середньо-діапазонні Радари (Medium-Range Radar - MRR) призначені для виявлення об'єктів на середніх відстанях, що може бути корисним при русі на швидкості на відкритих трасах та автострадах. MRR може використовуватися для систем адаптивного круїз-контролю та розпізнавання об'єктів.

Довго-діапазонні Радари (Long-Range Radar - LRR) призначені для виявлення об'єктів на великих відстанях і використовуються для систем детекції та виявлення транспортних засобів на великих швидкостях, таких як системи автоматичного гальмування в емергенційних ситуаціях.

Системи MIMO (Multiple Input, Multiple Output). Технологія MIMO використовує багатоантенні системи для відправлення та отримання радіосигналів. Це дозволяє отримувати більше інформації про об'єкти, їхню швидкість та напрямок руху.

Радари для Виявлення Рухомих Об'єктів (Moving Object Detection –

MOD) спеціально призначені для виявлення рухомих об'єктів, таких як пішоходи або велосипедисти, що робить їх ефективними для систем безпеки пішоходів.

Для системи навігації можна обрати радар Batman BM101 mmWave. Цей модуль підключається безпосередньо до комп'ютера Raspberry Pi 4 або NVIDIA Jetson Nano і здатний визначати розташування перешкод, відстань, наближення або віддалення (за допомогою доплерівських даних) від радара в діапазоні 20 метрів або навіть 50 метрів в очікуванні розміру об'єкта з точки зору радіолокаційного поперечного перерізу (RCS); Потім програміст може нанести на карту зону, вільну від перешкод, щоб направити безпілотний наземний транспортний засіб (UGV), сліпу людину або дрон у безпечну зону[13].

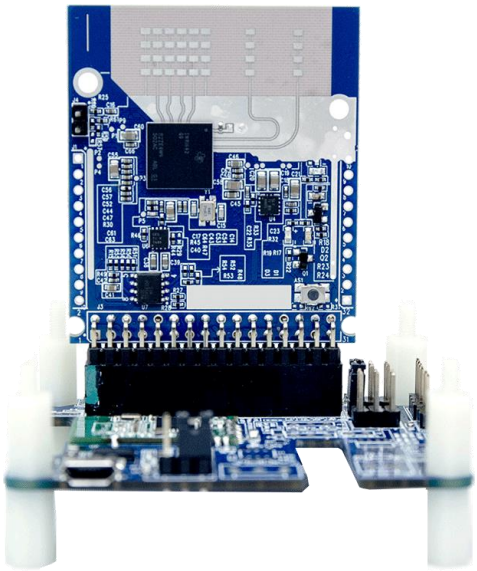


Рис. 1.4 – BM101 mmWave Module

Радари в системах автономного транспорту є важливими для виявлення та вимірювання відстані до об'єктів. Інновації в цій галузі спрямовані на покращення точності, зниження вартості та підвищення надійності, щоб забезпечити ефективну та безпечну роботу автономних транспортних систем.

Ультразвуковий датчик є важливим компонентом в системах автономного маршрутного планування та навігації в транспортних системах. Ці датчики використовують звукові хвилі для вимірювання відстаней до об'єктів навколо транспортного засобу. Ультразвукові датчики генерують

звукові хвилі, які відбиваються від об'єктів у навколишньому середовищі. Датчик затримує час, який звукова хвиля потребує для того, щоб повернутися, і на основі цього вимірює відстань до об'єкта.

Найбільш відомий ультразвуковий сенсор для Arduino це HC-SR04. Цей сенсор має велику популярність, доступність та невелику ціну. Діапазон дальності його виміру становить від 2 до 400 см. На його роботу не має істотного впливу електромагнітні випромінювання та сонячна енергія. Хоча цей датчик не зовсім підходить для Jetson Nano, його всеодно можна підключити за допомогою логічних трансляторів 5 В до 3.3, або можна використовувати Arduino[16].

Ультразвукові датчики зазвичай призначені для вимірювання коротких відстаней, зазвичай від кількох сантиметрів до кількох метрів. Це робить їх ефективними для детектування перешкод, які знаходяться в непрямому полі видимості. Ультразвукові датчики ідеально підходять для детекції ближніх об'єктів, таких як інші транспортні засоби, перешкоди, стіни та інші об'єкти, які можуть з'явитися в обмеженому діапазоні. Наприклад, якщо штучний інтелект не зміг розпізнати об’єкт за допомогою камери, то ультразвуковий датчик вкаже на перешкоду.

Однією з переваг ультразвукових датчиків є їх здатність працювати в режимі реального часу. Вони можуть швидко оцінювати відстані і забезпечувати миттєву інформацію для систем автономного управління. Використання ультразвукових датчиків у системах автономного транспорту дозволяє транспортним засобам ефективно взаємодіяти з оточуючим середовищем та підвищує рівень безпеки та навігації.

GPS(Global Positioning System) визначає географічне положення транспортного засобу за допомогою супутників. Висока точність та глобальний охоплення роблять GPS важливим елементом для визначення глобальної позиції. Однак у міських умовах, де сигнал може бути втрачений чи вражений високими будівлями, система GPS може супроводжуватися

іншими технологіями, такими як інерціальні датчики, для забезпечення більш точного позиціювання.

GPS грає ключову роль в системах автономного транспорту, забезпечуючи точне визначення місцезнаходження транспортного засобу. Однак для автономних систем навігації важливо використовувати не лише GPS, але і інші технології для отримання високоточної інформації про місцезнаходження та оточуюче середовище.

Вибір конкретного GPS залежить від вимог конкретної системи, умов експлуатації та бюджетних обмежень. Багатосенсорні системи часто використовують комбінації різних сенсорів, включаючи GPS, для забезпечення максимальної точності та надійності в різних сценаріях.

Аспекти, які слід враховувати при використанні GPS в автономних системах:

– Висока точність. У сучасних системах автономного транспорту важливо використовувати GPS-приймачі, які забезпечують високу точність визначення місцезнаходження. Деякі приймачі можуть підтримувати диференційне позиціонування (DGPS) або режими Real-Time Kinematics (RTK), що дозволяють досягти високої точності в декілька сантиметрів.

– Багатосистемні приймачі. Системи, які можуть використовувати сигнали не тільки з американської системи GPS, але й інших глобальних навігаційних систем, таких як ГЛОНАСС, Галілео, BeiDou, можуть забезпечити більшу надійність та доступність сигналів в різних областях світу.

– Інтеграція із іншими сенсорами. GPS-дані можуть бути інтегровані із вимірюваннями інших сенсорів, таких як лідари, радари, камери. Це дозволяє системі отримувати комплексну інформацію про місцезнаходження та навколишнє середовище.

– Інерційні Навігаційні Системи (INS). Комбінування GPS з інерційними сенсорами, такими як гіроскопи та акселерометри, може допомогти утримувати точність позиціонування навіть у випадках втрати сигналу GPS (наприклад, в тунелях або міських каньйонах).

– Обробка сигналів. Висока якість GPS-приймачів, здатних до обробки слабких сигналів та працювати в умовах обмеженої видимості (наприклад, під високими будівлями або в густому лісі), є важливим фактором для надійного позиціонування.

– Антенний вибір та розташування. Вибір високоякісної GPS-антени та її правильне розташування на транспортному засобі можуть покращити якість отримуваних сигналів і, відповідно, точність позиціонування.

– Захист від втрати сигналу. Для забезпечення безперебійної роботи важливо використовувати технології, такі як підтримка агрегації сигналів, що дозволяє системі автономного транспорту продовжувати навігацію в умовах, коли сигнал GPS може бути тимчасово втрачений.

В якості GPS підійде модуль SIM7600G-H 4G. Його можна підключити до багатьох різних мікрокомп’ютерів, і він чудово інтегрується з Jetson Nano. Також він має контрольні контакти UART для підключення до хост-плат, таких як Arduino/STM32. Має слот для SIM-карти, підтримує SIM-карту 1,8 В/3 В. В ньому є вбудований транслятор напруги, робочу напругу можна налаштувати на 3,3 В або 5 В за допомогою перемички. Швидкість передачі даних: 300 біт/с ~ 4 Мбіт/с.

Рис. 1.5 – SIM7600G-H 4G Module

Завдяки модулю SIM7600G-H 4G мікрокомп’ютер Jetson Nano може отримувати доступ до мережі LTE, а також може підключатися до супутників, за допомогою яких може отримувати дані про своє місцезнаходження. За

допомогою цього система може планувати подальший маршрут і транслювати дані та зображення з камери. Можливо навіть реалізувати віддалене керування роботом[18].

Інерціальні датчики використовуються в автономних системах для вимірювання руху і змін положення об'єктів в просторі. Основні типи інерціальних датчиків включають гіроскопи, акселерометри та магнітометри. Важливо враховувати, що комбінація цих датчиків може допомогти забезпечити точні та стабільні вимірювання руху та орієнтації в просторі.

Вибір конкретних інерціальних датчиків залежить від вимог конкретної системи, швидкості руху, точності, об'єму та вартості. Зазвичай використовуються комплексні системи, щоб взаємодіяти із зовнішніми сенсорами та забезпечити точне визначення руху транспортного засобу.

Типи інерціальних датчиків та їхні характеристики, які можуть бути використані в системах автономного транспорту:

1. Акселерометри:

Принцип Роботи: Акселерометри вимірюють прискорення транспортного засобу вздовж трьох осей (X, Y, Z).

Застосування: Вони використовуються для визначення змін швидкості та визначення положення транспортного засобу.

2. Гіроскопи:

Принцип Роботи: Гіроскопи вимірюють кутову швидкість об'єкта навколо трьох осей.

Застосування: Використовуються для визначення орієнтації та поворотів транспортного засобу.

3. Магнітнометри:

Принцип Роботи: Магнітнометри вимірюють магнітне поле, що дозволяє визначити орієнтацію транспортного засобу відносно земного магнітного поля.

Застосування: Допомагають визначати напрямок та орієнтацію транспортного засобу.

Gps-модуль може також показувати направлення, але робить це він не

досить точно. Для цієї задачі підійде 9-осьовий інерційний модуль – LSM9DS1. Цей компактний сенсор використовує I2C для зв'язку та дуже простий у використанні.

Усередині чіпа знаходяться три датчика, Одним з них є класичний 3-осьовий акселерометр, який може визначити напрямок (шляхом вимірювання сили тяжіння) або як швидко дошка розганяється в 3D просторі. Інший - 3-осьовий магнітометр, який може визначити, де знаходиться найсильніша магнітна сила, як правило, використовується як компас. Третій - 3-осьовий гіроскоп, який може вимірювати обертання та скручування. Поєднуючи це дані, які можна визначити положення в просторі та визначити сторони світу[19].

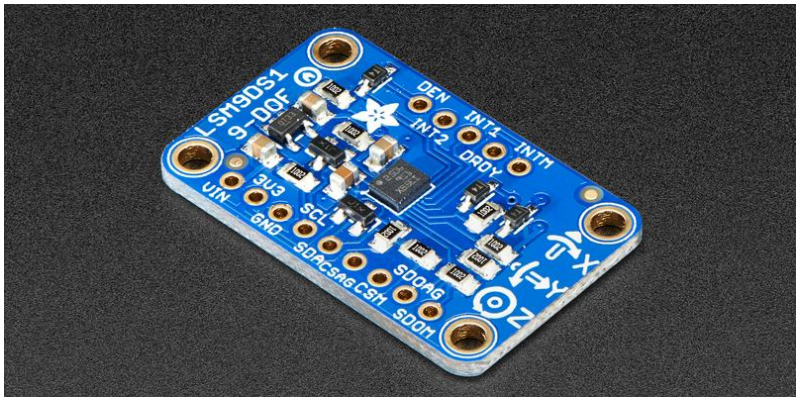


Рис. 1.6 – Модуль LSM9DS1

LSM9DS1 акселерометр має діапазони ±2/±4/±8/±16 g. LSM9DS1 магнітометр має діапазони ±4/±8/±12/±16 гаусів. Гіроскоп LSM9DS1 має діапазон ±245/±500/±2000 dps.

Сенсори в системах автономного маршрутного планування і навігації виконують ключову роль у створенні точної та безпечної транспортної інфраструктури. Їхнє поєднання та взаємодія грають вирішальну роль у забезпеченні транспортних засобів здатністю ефективно адаптуватися до змін у навколишньому середовищі та забезпечувати безпеку та комфорт для пасажирів.

1.5 Картографічні системи та карти

Картографічні системи та мапи є невід'ємною частиною систем автономного транспорту, забезпечуючи важливу інформацію для автономного маршрутного планування та навігації. Детальна та актуальна карта є ключовим елементом для безпечної та ефективної роботи автономних транспортних систем.

Збирати дані для карт можуть різні джерела, такі як GPS, лідари, радари, камери.

GPS використовується для визначення точного положення транспортного засобу на карті. Інтеграція з GPS дозволяє системі в реальному часі відслідковувати рух та орієнтуватися.

Лідари та радари генерують високоточні дані про структуру оточуючого простору, що може бути використано для створення деталізованих карт.

Камери реєструють візуальну інформацію, яка може бути використана для розпізнавання дорожньої знакової системи, пішоходів та інших об'єктів.

Використання супутникових та аерофотознімків дозволяє отримати високоякісні дані про ландшафт та інші географічні особливості.

Існує декілька служб карт:

OpenStreetMap (OSM):

OSM – це глобальна картографічна база даних, яку можна використовувати безкоштовно для створення та оновлення карт для автономного транспорту.

Google Maps та інші картографічні сервіси:

Картографічні сервіси, такі як Google Maps, надають доступ до деталізованих карт та API для використання в системах автономного транспорту.

Карти дозволяють системам автономного транспорту планувати оптимальні маршрути, уникати перешкод та обирати найбезпечніші шляхи. Камери, лідари та радари допомагають системі автономного транспорту виявляти та розпізнавати об'єкти, такі як інші транспортні засоби, пішоходи та

дорожні знаки. Динамічне оновлення карт дозволяє системі адаптуватися до змін у дорожніх умовах, таких як ремонти, затори чи інші тимчасові перешкоди. Детальні та актуальні карти допомагають уникнути небезпек та ризиків на дорозі, підвищуючи загальну безпеку та знижуючи ймовірність аварій.

Усі ці елементи взаємодіють, створюючи комплексну картографічну інфраструктуру для автономного транспорту. Забезпечення точних, оновлюваних та різноманітних даних є критичним аспектом розвитку та безпеки автономних систем, які базуються на навігаційних та маршрутних рішеннях.

1.6 Висновки

Автономні системи для маршрутного планування та навігації в транспортних системах представляють собою складні інженерні рішення, спрямовані на автоматизацію та підвищення ефективності транспортних засобів.

Існує ряд різноманітних систем, які вже розроблені та впроваджені в сучасних автономних транспортних засобах, таких як Waymo, Tesla Autopilot, Uber ATG, Baidu Apollo та Mobileye. Кожна з цих систем використовує комбінацію сенсорів, алгоритмів машинного навчання та інших технологій.

Багато систем спільно використовують камери, лідари, радари, алгоритми машинного навчання та технології взаємодії з оточенням для отримання та обробки інформації про навколишнє середовище. Висока точність сенсорів та надійність функціонування є критичними факторами для безпечної експлуатації системи.

Головним завданням такої системи є знайти шлях до цілі, а також минати перешкоди. Для цього було обрано необхідні комплектуючі: мікрокомп’ютер, камера, лідар, радар, gps-модуль, акселерометр, магнітометр, гіроскоп.

2 РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ

2.1 Платформа JetRacer

Система автономної навігації буде працювати на основі платформи JetRacer. JetRacer — це автономний гоночний автомобіль зі штучним інтелектом, який використовує NVIDIA Jetson Nano. Він складається з багатьох компонентів:

- IMX219 8 Мп HD, ширококутна камера з кутом огляду 160°;
- Oled дисплей 0,91" 128×32 пікселів.
- AC8265 бездротова мережа, дводіапазонний Wi-Fi 2.4G/5G, Bluetooth 4.2.
- Високошвидкісні двигуни 4WD. Диференціали переднього і заднього моста.
- Акумулятор 8,4 В, 18650 × 4 (два паралельно, два послідовно).
- Високошвидкісний двигун з вуглецевою щіткою RC380 Швидкість холостого ходу 15000 об/хв.
- Сервопривід. Крутний момент 6 кг/см.
- Шасі з якісного міцного пластикового корпусу.
- Ударостійка лицьова губка.

В цьому комплекті є плата розширення. В неї вставляється акумулятор, підключається зарядний пристрій і активується захист акумулятора. Потім до плати розширення підключається сервопривід. За допомогою цього сервоприводу відбувається поворот колес. До плати розширення підключається двигун, а також Jetson Nano. Зверху на Jetson Nano встановлюється вентилятор охолодження. На плату розширенння можна встановити антени, щоб підключатися до мережі.

Також встановлюється кронштейн з камерою, яка підключається до Jetson Nano.

Повну збір платформи можна подивитися на офіційному сайті

WaveShare.

Рис. 2.1 – Рухома платформа JetRacer

Самим головним компонентом є Jetson Nano. На цьому маленькому чіпі встановлений 128-ядерний графічний процесор, що використовує архітектуру Maxwell від Nvidia, здатний видавати 472 ГФЛОПС. (Не кажучи вже про 4 ГБ оперативної пам'яті та чотирьохядерний процесор ARM A57.) FLOPS (або FLoating point OPerations per Second) — це обчислювальна потужність процесора, і 472 мільярди з них досить непогано для комп'ютера.

Після збирання набору потрібно підготувати SD-карту , яка повинна бути не менше 64G. На неї потрібно завантажити образ JetRacer за допомогою програми для запису образів Etcher. Після цього потрібно вставити SD-карту в слот для SD-карти в Jetson Nano.

Jetson Nano має 40-контактний GPIO. Порти GPIO (General Purpose Input/Output) розташовані на верхньому краї плати Jetson Nano. Він виглядає як два довгих ряди металевих штифтів, до яких можна прикріпити до плати функціонуючі порти, такі як світлодіоди та перемикачі. Ці контакти можна використовувати для введення та виведення.

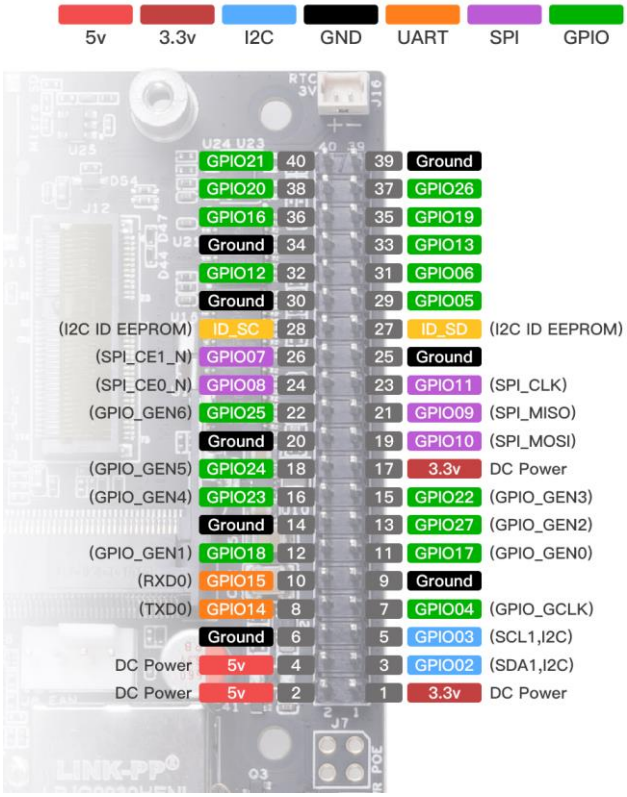


Рис. 2.2 – Опис контактів GPIO Jetson Nano

Усі комунікаційні контакти I2C, крім контактів I2C на роз'ємі, безпосередньо підключені до модуля Jetson Nano. Контакти I2C підключені до проміжного перемикача рівня для зміни напруги від 1,8 В на модулі до 3,3 В на стандартному інтерфейсі I2C. Це контакти 3 і 5 (контакти I2C SDA) і контакти 27 і 28 (контакти I2C SCL). Контакти 8 і 10 - це контакти передавача UART (TX) і приймача (Rx) відповідно.

На платі також передбачено два джерела живлення на 3,3 В (контакти 1 і 17) і два на 5 В (контакти 2 і 4). Роз'єм розширення також має кілька контактів заземлення.

Роз'єм розширення має лише 2 ШІМ-канали, безпосередньо підключені до апаратних ШІМ-контролерів. Він не підтримує жодних програмно згенерованих ШІМ. На жаль, модуль Nano за своєю суттю не налаштований на визначення з'єднання з апаратним забезпеченням ШІМ. Таким чином, для використання апаратного забезпечення ШІМ, система PinMux повинна бути налаштована на забезпечення функціональності ШІМ-контактів.

2.2 Підключення модуля SIM7600G-H 4G

Модуль SIM7600G-H 4G прикріпляєть до Jetson Nano на 40-контактний GPIO. На модуль підключається основна антена та антена GPS.

Це модуль зв'язку 4G/3G/2G і позиціонування GNSS, розроблений для Jetson Nano, він підтримує глобальний LTE CAT4 зі швидкістю до 150 Мбіт/с для передачі даних по низхідній лінії з досить низьким енергоспоживанням. Модуль має багато функцій:

– 40-контактний роз'єм розширення GPIO для підключення Jetson Nano.

– Підтримує комутоване з'єднання, телефонний дзвінок, SMS, пошту, TCP, UDP, DTMF, HTTP, FTP тощо.

– Підтримує позиціонування базових станцій GPS, BeiDou, Glonass, LBS.

– Вбудований інтерфейс USB для тестування AT-команд, отримання даних GPS-позиціонування тощо.

– Контрольні контакти UART для підключення до хост-плат, таких як Arduino/STM32.

– Слот для SIM-карти, підтримує SIM-карту 1,8 В/3 В.

– Вбудований 3,5-мм аудіороз'єм з підтримкою навушників і мікрофона для здійснення телефонних дзвінків.

– Вбудований транслятор напруги, робочу напругу можна налаштувати на 3,3 В або 5 В за допомогою перемички.

– Швидкість передачі даних: 300 біт/с ~ 4 Мбіт/с (за замовчуванням: 115200 біт/с).

Однак головними функціями є GPS та доступ до мережі. Для того щоб отримати доступ до мережі, потрібно вставити SIM-карту в слот для SIM-карти.

Рис. 2.3 – Підключений модуль SIM7600G-H до Jetson Nano

GPS-модуль – це пристрій, який знаходить супутники і встановлює з ними зв'язок, а потім, використовуючи отримані сигнали, визначає поточне місцезнаходження пристрою в просторі. Інформацію про місцеположення модуль надає у форматі координат довготи та широти.

Супутникова система складається з 24 супутників, розташованих в шести орбітальних площинах навколо Землі, при чому в кожній площині міститься по чотири супутники. Супутники обертаються на висоті 20000 км і переміщаються зі швидкістю 14000 км в годину. GPS працює за допомогою методу, що називається трилатерацією. Цей метод використовується для розрахунку місцеположення, швидкості та висоти, отримуючи сигнали з супутників. Його часто помилково сприймають як тріангуляцію, яка вимірює кути, а не відстані. Кожен супутник випромінює унікальний сигнал і параметри орбіти, які дозволяють GPS-приладам декодувати та обчислювати місцеположення супутника. Приймач GPS використовує цю інформацію для точного визначення місцеположення. Фактично GPS вимірює відстань до кожного супутника, враховуючи час, потрібний для отримання переданого сигналу. Для обчислення координат свого місцезнаходження і відстеження руху GPS-модуль повинен бути підключений щонайменше до трьох супутників. Основною причиною похибок визначення GPS-координат є

спотворення сигналу від супутника великими будівлями. У великих містах із високими будівлями сигнал для GPS-модуля може бути складно зловити, бо він відбивається від поверхні будівель. Це призводить до збільшення довжини шляху і, відповідно, до появи похибок у вимірюваннях. Крім того, модуль не може працювати в приміщенні, оскільки не може отримати сигнал.

Існують три типи стартів GPS-приймача: "холодний", "теплий" та "гарячий". Під час "холодного" старту прилад отримує дані альманаху та ефемерідів від супутників, які зберігаються в його пам'яті для подальшої роботи. Альманах містить загальні дані про параметри орбіти супутників, а ефемеріди надають уточнені дані для конкретного супутника у конкретний момент часу. Якщо прилад був вимкнений тривалий час, то під час увімкнення відбувається "холодний" старт, оскільки дані альманаху і ефемерідів застарілі і потребують оновлення. Цей процес триває тривалий час і може зайняти кілька хвилин.

Якщо прилад був увімкнений через більше 30 хвилин після вимкнення, то відбувається "теплий" старт, де дані альманаху ще є актуальними, але дані ефемерідів вже застарілі. У цьому випадку сенсор оновлює дані ефемерідів на основі альманаху. Якщо прилад був увімкнений менше 30 хвилин після вимкнення, він запускається в режимі "гарячого" старту, використовуючи актуальні дані альманаху та ефемерідів, які ще не втратили своєї актуальності. При цьому пристрій запускається дуже швидко і миттєво починає свою роботу.

2.3 Лідар

Light Detection and Range (LiDAR) працює так само, як ультразвукові далекоміри з лазерним імпульсом використовується замість звукових хвиль.

RPLIDAR — це недорогий датчик LIDAR, який підходить для роботизованого застосування SLAM (одночасна локалізація та картографування) у приміщенні. Його можна використовувати в інших програмах, таких як:

– Загальна навігація та локалізація роботів
– Обхід перешкод
– Сканування навколишнього середовища та 3D-моделювання

Лідар можна підключити до Jetson Nano до USB-порту за допомогою USB-адаптеру. Сам лідар можна прикріпити над платформою як показано на рисунку 2.4.

Рис. 2.4 – RPLIDAR на платформі JetRacer

Slamtec RPLIDAR – це 360-градусний лазерний сканер, який використовується в різних додатках, включаючи автономні транспортні засоби. Він працює, випромінюючи імпульс модульованого лазерного світла, яке відбивається від об'єктів і зчитується оптичним приймачем. Час, необхідний для відбиття світла, використовується для обчислення відстані до об'єкта 1. RPLIDAR використовує одне джерело живлення 5 В постійного струму для одночасного живлення двигуна сканування та ядра сканування.

Принцип роботи LIDAR досить простий. Сфокусований світловий промінь спрямовується на об'єкт і датчик шукає його відображення. При

виявленні променя вимірюється його інтенсивність і кут (або фаза). Потім ці значення підключаються до рівняння, яке запускається швидким бортовим комп'ютером для визначення положення та характеристик відбиваючих об'єктів.

RPLidars працюють шляхом обертання лазерного випромінювача та приймача. Лазер випромінює світло, приймач приймає світло. Оскільки приймач знає, коли випромінювалося світло, він може точно виміряти, скільки часу потрібно, щоб світло відбилося від будь-яких об’єктів на шляху світла.

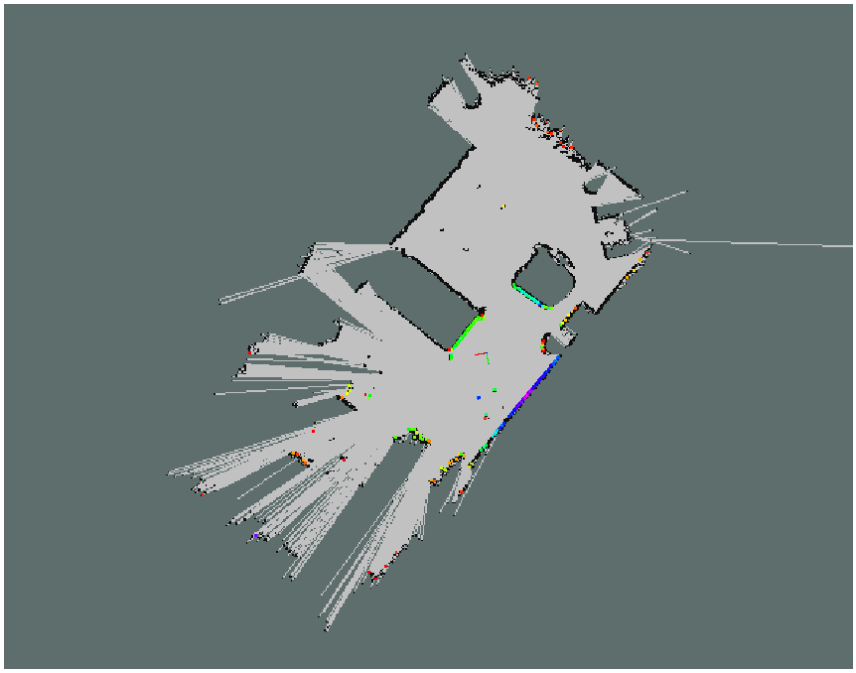


Рис. 2.5 – Приклад роботи RPLIDAR

RPLIDAR має такі особливості:

– Виконує сканування на 360 градусів за годинниковою стрілкою.

– Може виконувати понад 8000 вимірювань даних про відстань за секунду.

– Частота сканування від 2 до 10 Гц. Це означає, що можна виконувати повне сканування на 360 градусів від 2 до 10 разів на секунду, швидкість якого залежить від швидкості двигуна.

– Дальність виявлення 12 метрів.

– Кутова роздільна здатність 1 градус.

– Роздільна здатність відстані 0,2 см.

– SDK та інструменти з відкритим вихідним кодом.

– Інтегрується з ROS (Robot Operating System).

– Включає послідовний перетворювач в USB, що дозволяє підключити сенсор до комп’ютера чи мікрокомп’ютера(Jetson Nano).

2.4 Радар

SRR (Short-Range Radar) – це радари, які призначені для коротких дистанцій, зазвичай від кількох сантиметрів до декількох метрів.

SRR радари генерують короткочасні мікрохвильові сигнали, які випромінюються у визначеному напрямку. Випромінені сигнали взаємодіють з навколишніми об'єктами. Якщо на шляху сигналу є об'єкт, частина енергії відбивається від об'єкта назад до радару. Радар приймає відбиті сигнали та аналізує їх для визначення властивостей об'єктів, таких як відстань, швидкість, розмір та напрямок руху. Зібрані дані використовуються для створення радарного зображення, яке використовується штучним інтелектом для прийняття рішень.

В системах безпеки автомобілів SRR радари використовуються для виявлення можливих колізій, визначення відстані до інших транспортних засобів чи перешкод та автоматичного виконання екстрених заходів, якщо це необхідно.

SRR радари можуть визначати не тільки наявність об'єктів, але і їх характеристики, такі як розмір і швидкість. Це дозволяє визначати, наприклад, чи є об'єкт людина чи інше транспортне засобі.

SRR радари можуть працювати в умовах обмежень видимості, таких як туман, дощ чи сніг, оскільки мікрохвильові сигнали менше схильні до впливу погодних умов, ніж оптичні системи.

Модуль BM101 mmWave може виявляти людину на відстані до 20 м. Виявлення автомобіля відбувається на відстані до 50 м. Куд огляду модуля 120 градусів.

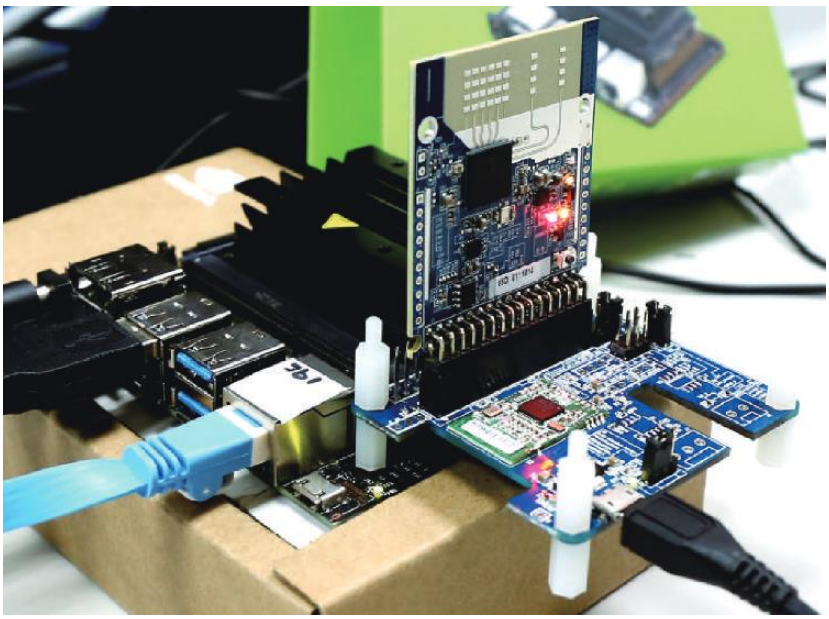


Рис. 2.6 – Підключений модуль BM101 mmWave до Jetson Nano

Модуль BM101 використовує джерело живлення 5 В з мікро USB. Після натискання кнопки RESET жовтий світлодіод почне блимати, вказуючи на нормальну роботу. Для роботи з Jetson Nano треба встановити перемичку J1 на положення 1,2 і перемичку J12 положення 1,2.

Вхід PIN12 регулює швидчкість передачі необролених даних. При значенні High швидкість дорівнює 921600. При значенні Low швидкість дорівнює 115200. PIN23 є UART виходом для виводу інформації і працює при напрузі 3,3 В. Піни 3 і 5 є пінами напруги 5В і заземлення відповідно. Також є ще інтерфейси UART, SPI, I2C.

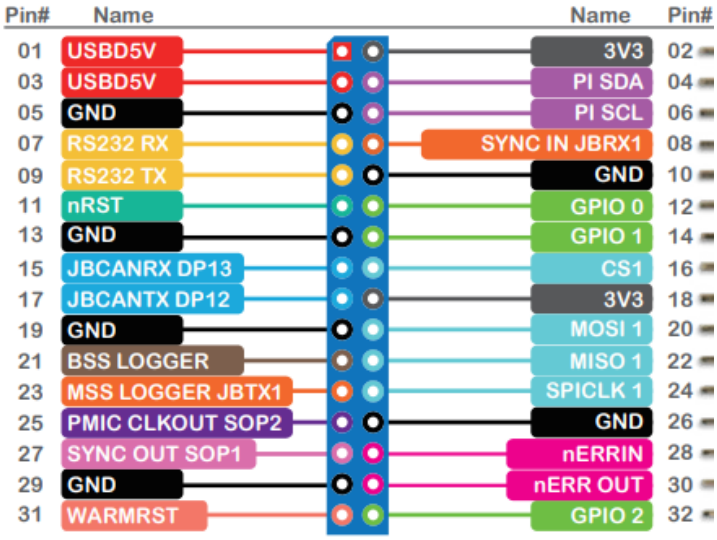


Рис. 2.7 – GPIO модуля BM101

2.5 Ультразвуковий датчик

Ультразвукові датчики використовують ультразвукові хвилі для визначення відстані від датчика до об'єктів в їхньому оточенні. Цей сенсор використовує спеціальний перетворювач для відправлення та отримання ультразвукових імпульсів, які надають інформацію про віддаленість об'єкта. Високочастотні звукові хвилі відбиваються від кордонів, утворюючи чіткі ехо-сигнали.

Для визначення відстані між сенсором і об'єктом, сенсор вимірює час, який пройшов між моментом випромінювання звуку передавачем і його приходом до приймача.

Ультразвук – це звукова хвиля з високою частотою, яка перевищує межі чутного спектру людського слуху. Зважаючи на те, що звичайно прийнято вважати чутний діапазон частот від 16 Гц до 20 кГц, термін "ультразвук" зазвичай вказує на акустичні хвилі з частотою вище 20 кГц.

Для ефективної роботи ультразвукового сенсора використовуються дві ключові компоненти: передавач і приймач, які зазвичай розташовані дуже близько один до одного. Коли приймач і передавач знаходяться в непосередній близькості, звук поширюється по більш прямій лінії від передавача до

виявленого об'єкта і знову до приймача, що сприяє зменшенню похибок у вимірюваннях.

З передавача виходять акустичні хвилі, структура яких подібна до світла від ліхтарика, а не лазера, тому необхідно враховувати їх поширення та кут променя. Залежно від того, наскільки далеко акустичні хвилі виходять з передавача, область виявлення збільшується в поперечному та вертикальному напрямках.

Якщо промінь ультразвукового сенсора є вузьким, то дальність виявлення збільшиться, оскільки енергія ультразвукового імпульсу буде більш сфокусованою і матиме змогу пройти далі, перш ніж розсіятися до рівнів, які не придатні для використання. З іншого боку, широкий промінь буде розповсюджувати енергію по широкій зоні, що призводить до зменшення дальності виявлення.

Ультразвукові датчики є важливою частиною сенсорного арсеналу для систем, які потребують визначення відстані та уникнення перешкод на низьких діапазонах відстаней.

З підключенням ультразвукового датчика HC-SR04 до Jetson Nano можуть виникнути деякі проблеми, оскільки HC-SR04 толерантний лише до 5 В, тому потребує логічних трансляторів від 5 В до 3.3 при підключенні контактів GPIO до Jetson Nano. Для цього можна використовувати I2C конвертер що зображений на рисунку 2.7.

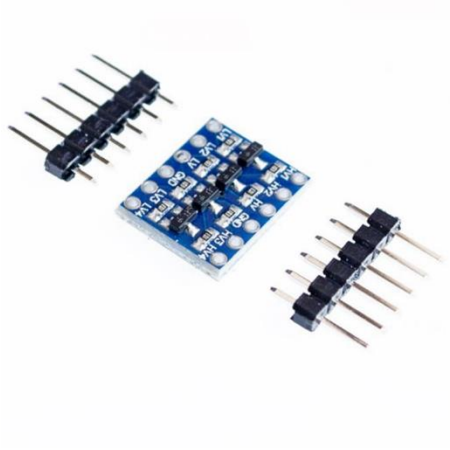


Рис. 2.8 – Logic Level Converter від 5V до 3.3V

Однак до цього потрібна ще реалізація на стороні програмного забезпечення.

Виклик ультразвукового датчика безпосередньо з Jetson Nano може бути не ідеальним, оскільки Jetson Nano не виконує обробку в реальному часі. Це призводить до того, що вимірювання відстані стає ненадійним. Краще використовувати мікроконтролер Arduino Nano для вимірювання відстані. І вже з Arduino передавати дані на Jetson Nano по USB з’єднанню. Однак при такому способі Arduino буде займати багато місця, однак вимірювання буде кращим.

Ультразвуковий далекомір HC-SR04 має такі технічні параметри:

– Напруга для живлення: 5В;

– Робочий параметр сили струму: 15 мА;

– Сила струму в пасивному стані < 2 мА;

– Оглядовий кут – 15 °;

– Вимірювальний кут - 30 °;

Сенсор має 4 піни: контакт живлення 5 В, Trig – сигнал входу, Echo – сигнал виходу, GND – заземлення.

Послідовність дій для отримання даних така:

– подаємо імпульс тривалістю 10 мкс на виведення Trig;

– всередині далекоміра вхідний імпульс перетворюється на 8 імпульсів частотою 40 кГц і посилається вперед через випромінювач T;

– коли надіслані імпульси доходять до перешкоди, вони відбиваютьсяі приймаються приймачем R, в результаті отримуємо вихідний сигнал на виведенні Echo;

– отриманий сигнал переводиться в відстань контролером.

2.6 Компас на базі модуля LSM9DS1

LSM9DS1 - це універсальна система з датчиком руху в чіпі. Він містить 3-осьовий акселерометр, 3-осьовий гіроскоп і 3-осьовий магнітометр - 9DOF(degrees of freedom) в одній мікросхемі.

Модуль LSM9DS1підключається до Jetson Nano за допомогою I2C інтерфейсу. SCL підключається до піну 5, а SDA до піну 3.

I2C — це протокол послідовного зв'язку, де дані передаються біт за бітом по одному проводу, який називається лінією SDA. Протокол є синхронним, що означає, що вихідні біти синхронізуються з вибіркою бітів за допомогою тактового сигналу. Цей тактовий сигнал розділяється між Master (головним пристроєм) та Slave (підпорядкованим пристроєм), при цьому Master завжди контролює тактовий сигнал.

Гіроскоп може вимірювати кутову швидкість, тобто «з якою швидкістю і по якій осі я обертаюся?». Кутові швидкості вимірюються в градусах в секунду - зазвичай їх скорочують до ДПС або °/с. LSM9DS1 може вимірювати ± 2000 DPS, хоча цей масштаб також можна встановити на 245 або 500 DPS, щоб отримати точнішу роздільну здатність.

Акселерометр вимірює прискорення, яке показує, як швидко змінюється швидкість - «з якою швидкістю я прискорююся або сповільнююся?». Прискорення зазвичай вимірюється в м/с^2 (метрів в секунду в секунду) або g (сила тяжіння [близько 9,8 м/с^2]).Якщо об'єкт сидить нерухомо, він відчуває прискорення близько 1 г до землі. LSM9DS1 вимірює його прискорення в g, і його шкала може бути встановлена на ± 2, 4, 8 або 16 g.

Є магнітометр, який вимірює потужність і напрямок магнітних полів. LSM9DS1 вимірює магнітні поля в одиницях Гаусах (Gs) і може встановити свою шкалу вимірювання на ± 4, 8, 12 або 16 Gs.

Вимірюючи ці три властивості, ви можна отримати багато знань про рух і орієнтацію об'єкта.

LSM9DS1 вимірює кожну з цих властивостей руху в трьох вимірах. Це означає, що він виробляє дев'ять фрагментів даних: прискорення в x/y/z, кутове обертання в x/y/z і магнітна сила в x/y/z. LSM9DS1 Breakout має мітки, що вказують орієнтації осей акселерометра та гіроскопа, які мають відношення правої руки один до одного.

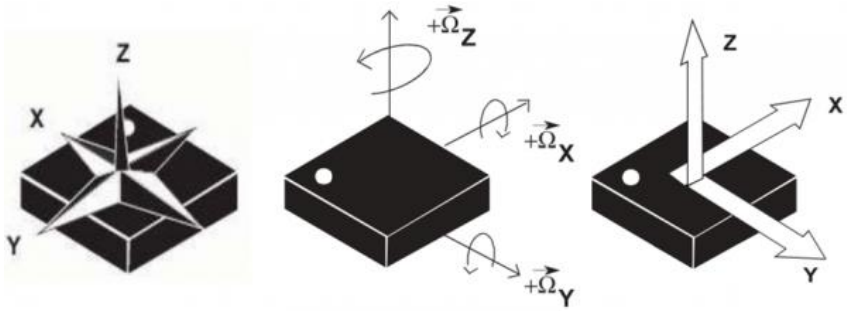


Рис. 2.9 – Орієнтації осей LSM9DS1

Одна половина пристрою піклується про всі функції гіроскопа і акселерометра, а інша половина управляє виключно магнітометром.

Модуль LSM9DS1 має 13 пінів:

– GND: земля;
– VDD: джерело живлення. Напруга повинна регулюватися між 2,4 В і 3,6 В;
– SDA: SPI: MOSI. I2C: Серійні дані;
– SCL: Послідовний годинник;
– DEN: Включення даних гіроскопа;
– INT1, INT2: програмовані переривання для акселерометра і гіроскопа. Їх можна налаштувати на оповіщення про перевищення/заниження порогових значень;
– INTM: Програмоване переривання для магнітометра. Можна налаштувати на оповіщення про перевищення порогових значень;
– RDY: Доступне переривання, що вказує на нові дані магнітометра;
– CS M: Цей контакт вибирає між I2C і SPI на магнітометрі;
– CS AG: Цей контакт вибирає між I2C і SPI на акселі/гіроскопі;
– SDO M: У режимі SPI це вихід даних магнітометра (SDO_M);
– SDO AG: У режимі SPI це виведення даних акселерометра та гіроскопа.

LSM9DS1 - це пристрій на 3,3 В, тому він чудово підходить для Jetson Nano.

2.7 Висновки

Була розглянута широка палітра компонентів, які спільно створюють інтегровану систему для автономного маршрутного планування та навігації. Кожен з елементів грає свою унікальну роль у забезпеченні функціональності системи, а їх взаємодія формує потужний апаратний фундамент.

Мікрокомп'ютер Jetson Nano виступає в ролі центрального мозку системи, обробляючи дані з усіх сенсорів і виконуючи алгоритми планування маршруту. GNSS GPS модуль SIM7600G-H забезпечує точне геопозиціювання і орієнтацію системи. Лідар Slamtec RPLIDAR відповідає за створення точної тривимірної карти навколишнього середовища, необхідної для ефективного маршрутного планування.

SSR радар BM101 mmWave дозволяє виявляти об'єкти на коротких відстанях і забезпечує безпеку системи у відносно обмеженому просторі. Ультразвуковий датчик HC-SR04 служить для визначення відстаней на низьких діапазонах та уникнення перешкод. Акселерометр, гіроскоп та магнітометр LSM9DS1 відповідають за вимірювання руху і орієнтації системи.

Ця інтегрована апаратна система створена для оптимального використання різноманітних сенсорів та обладнання, забезпечуючи високий рівень точності та надійності в автономному маршрутному плануванні та навігації. Узгоджена робота всіх компонентів сприяє створенню ефективної системи.

3 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

3.1 Підготовка системи Jetson Nano

Linux - це сімейство операційних систем на основі ядра Linux, яке виникло у 1991 році від розробки фінського програміста Лінуса Торвальдса. Linux базується на принципах відкритого коду, що означає, що користувачі мають доступ до вихідного коду операційної системи. Це забезпечує свободу модифікацій, розповсюдження та використання. Існує велика кількість дистрибутивів Linux, таких як Ubuntu, Fedora, Debian, CentOS, і багато інших. Кожен з них має свою філософію, особливості та ряд пакетів.

Ядро Linux відповідає за обробку завдань ядра, керування ресурсами, драйвери пристроїв та взаємодію з апаратним забезпеченням. Воно є ключовим компонентом операційної системи. Linux підтримує мультизадачність, дозволяючи виконувати одночасно кілька завдань.

Jetson Nano використовує операційну систему NVIDIA JetPack, яка ґрунтується на Ubuntu Linux. Операційна система Jetson Nano спеціально адаптована для вбудованих систем та має підтримку апаратного забезпечення, що входить у склад платформи Jetson Nano.

JetPack є набором SDK (Software Development Kit), який містить інструменти для розробки та оптимізації програм для платформи Jetson. Він включає в себе бібліотеки для машинного навчання, бібліотеки комп'ютерного зору, інструменти для налагодження та розгортання програм.

Операційна система використовує файлову систему ext4, що дозволяє ефективно управляти файлами та ресурсами пам'яті. Також вона включає в себе системні бібліотеки та драйвери, які оптимізовані для роботи з апаратною частиною Jetson Nano.

Спочатку потрібно завантажити образ JetRacer на SD-карта, розміром не менше 64 Гб. Образ можна завантажити за допомогою програми Etcher. Після чого SD-карта вставляється в Jetson Nano.

Потрібно підключити Jetson Nano до ПК через мікро USB. Далі в браузері треба перейти за адресою 192.168.55.1:8888. Після цього запитає пароль, за замовчанням пароль jetson. Після введення пароля перекидає на сторінку Jupyter Lab, саме тут в подальшому буде відбуватися робота з Jetson Nano. Є ще і інший варіант, можна підключити Jetson Nano до монітора або телевізора за допомогою HDMI кабеля і підключити до мікрокомп'ютера мишку та клавіатуру.

Спочатку потрібно підключити Jetson Nano до wi-fi. Для цього потрібно відкрити термінал і ввести в нього наступну команду:

sudo nmcli device wifi list

Ця команда показує список доступних мереж.

```
jetson@nano-4gb-jp451:~$ sudo nmcli device wifi list
[sudo] password for jetson:
IN-USE  SSID           MODE   CHAN  RATE        SIGNAL  BARS  SECURITY
*       NETMIX_GUSHPIT  Infra  10    130 Mbit/s  77      ▂▄▆_  WPA2
jetson@nano-4gb-jp451:~$
```

Рис. 3.1 – Результат виконання команди nmcli device wifi list

Далі можна підключитися до мережі. Для цього використовується команда:

sudo nmcli device wifi connect <ssid_name> password <password>

Після цього можна перевірити підключення за допомогою команди ifconfig.

Коли Jetson Nano підключено до мережі, то тепер можна від'єднати мікро USB. На OLED-дисплеї, який знаходиться на платі розширення, тепер відображається адреса. За допомогою неї тепер можна підключатися до Jetson Nano бездротово.

За замовчуванням python буде вже встановлений в системі, однак якщо це не так, то можна встановити його вручну за допомогою наступних команд:

cd $HOME

git clone https://github.com/NVIDIA-AI-IOT/jetracer

```
cd jetracer
sudo python3 setup.py install
```

Jetson Nano має два режими роботи: MAXN, 5V. Вони відрізняються тим, що в першому режимі мікрокомп'ютер використовує всю свою потужність і при цьому споживає багато струму. В такому випадку батарея швидко розрядиться. Другий режим не дозволяє споживати більше струму, ніж може подати акумуляторна батарея.

Щоб перевірити режим треба ввести команду:

```
sudo nvpmodel -q
```

Щоб перемикнути режим MAXN використовується команда:

```
sudo nvpmodel -m0
```

Щоб перемикнути режим 5V використовується команда:

```
sudo nvpmodel -m1
```

3.2 Управління компонентами

Для написання програмного забезпечення правління компонентами та обробки даних використовується мова програмування Python. Python, мова програмування високого рівня, стала незамінним інструментом у сфері розробки імплементації алгоритмів штучного інтелекту, обробки даних, роботи з сенсорами та створення алгоритмів для планування маршрутів.

Python відомий своєю простотою та зрозумілістю синтаксису, що робить його ідеальним вибором для швидкого прототипування та розробки рішень в галузі штучного інтелекту. Інтеграція з різноманітними бібліотеками та фреймворками, такими як TensorFlow, PyTorch, OpenCV та ROS, надає безмежні можливості для розробки складних систем.

Python використовується для обробки зображень та відео, отриманих з камер, лідарів та інших сенсорів. За допомогою бібліотеки OpenCV, можна виконувати такі операції, як зчитування, аналіз та обробка зображень у реальному часі.

Для розробки програмної частина знадобляться наступні бібліотеки:

– jetracer: вбудована бібліотека, яка використовується для управління моторами;

– Jetson.GPIO: використовується для управління різними модуля через контаки GPIO;

– serial: бібліотека для отримання даних з послідовного порту;

– Jetson-Nano--LSM9DS1: бібліотека, для простого використання модуля LSM9DS1, також ця бібліотека використовує бібліотеку smbus;

– rplidar-roboticia: бібліотека, що написана для зручного використання лідарів від SLAMATEC;

– mmWave: використовується для керування присторями mmWave, в тому числі і Short Range Radar.

Для того щоб встановить бібліотеку на python використовується команда:

```
pip install <lib_name>
```

Для управління платформою використовується уже встановлена бібліотека jetracer. Щоб керувати колесами та моторами потрібно імпортувати бібліотеку:

```
from jetracer.nvidia_racecar import NvidiaRacecar
```

NvidiaRacecar реалізує клас Racecar, тому має два атрибути: throttle та steering. Ми можемо призначити цим атрибутам значення в діапазоні [-1, 1].

Клас NvidiaRacecar має два значення steering_gain і steering_bias, які можна використовувати для калібрування керування.

Остаточне значення керування обчислюється за допомогою рівняння:

$$y = a \times x + b, \tag{3.1}$$

де a – car.steering_gain,

b – car.steering_offset,

x – car.steering,

y – це значення, записане в драйвер двигуна.

Ці значення можна відрегулювати так, щоб при значенні 0 платформа рухалася вперед, а при значенні 1 переміщувалася повністю праворуч, а -1 – повністю ліворуч.

Throttle також має значення підсилення, яке можна використовувати для керування швидкості. Потужність throttle обчислюється як:

$$y = a \times x, \tag{3.2}$$

де a – car.throttle_gain,

x – car.throttle,

y – це значення, записане на регулятор швидкості.

Загалом для регулювання напрямку треба змінювати значеннння car.steering. Для того щоб регулювати швидкісь треба змінювати значення car.throttle. Для тестування це значення можна виставити на 0,5.

Для отримання даних з ультразвукового датчика HC-SR04 використовується Arduino. Arduino підключається до Jetson Nano через USB. Для того щоб отримати дані з serial monitor використовується бібліотека serial.

Після того як Arduino було підключене до Jetson Nano треба перевірити порт за допомогою команди:

ls /dev/tty*

Результатом виконання команди буде багато адрес, серед яких повинно бути /dev/ttyUSB0 або /dev/ttyACM0.

Отримання даних відбувається за допомогою:

```
import serial
ser = serial.Serial(port='/dev/ttyUSB0', baudrate=9600)
while True:
    if ser.in_waiting > 0:
        # Зчитуємо дані з порту
        data = ser.readline().decode('utf-8').strip()
```

Однак для зчитування даних з датчиків потрібно написати прошивку для Arduino:

```
#include <HCSR04.h>
```

```
UltraSonicDistanceSensor distanceSensor(8, 9);
void setup () {
    Serial.begin(9600);
}
void loop () {
    // Every 500 miliseconds, do a measurement using the sensor and print the distance in centimeters.
    float data = distanceSensor.measureDistanceCm();
    Serial.println(data);
    delay(100);
}
```

Спочатку імортується бібліотеку HCSR04, потім ініціалізуються піни 8 та 9, до яких підключено trigger та echo. Функція Serial.begin(9600) використовується для передачі та отримання даних з комп'ютера чи інших пристроїв через вбудований USB-порт зі встановленою швидкістю передачі даних. Далі за допомогою функції distanceSensor.measureDistanceCm() отримуються значення з датчика і потім передається в serial monitor.

Отримання даних з GPS також відбувається через serial. Також використовується бібліотека Jetson.GPIO.

Для включення модуля використовується функція:

```
def powerOn():
      print('SIM7600X is starting:')
      GPIO.setmode(GPIO.BCM)
      GPIO.setwarnings(False)
      GPIO.setup(6,GPIO.OUT)
      time.sleep(0.1)
      GPIO.output(6,GPIO.HIGH)
      time.sleep(2)
      GPIO.output(6,GPIO.LOW)
      time.sleep(20)
      ser.flushInput()
      print('SIM7600X is ready')
```

В цій функції ініціалізуються GPIO та встановлює потрібні параметри. Записується високий рівень на GPIO для увімкнення модема.

Для отримання позиції використовується наступна функція:

```
def getGpsPosition():
        rec_null = True
      answer = 0
      print('Start GPS session...')
      rec_buff = ''
      sendAt('AT+CGPS=1,1','OK',1)
      time.sleep(2)
      while rec_null:
            answer = sendAt('AT+CGPSINFO','+CGPSINFO: ',1)
            if 1 == answer:
                  answer = 0
                  if ',,,,,,,,' in rec_buff:
                        print('GPS is not ready,wait 10 seconds')
                        rec_null = False
                        time.sleep(10)
            else:
                  print('error %d'%answer)
                  rec_buff = ''
                  sendAt('AT+CGPS=0','OK',1)
                  return False
            time.sleep(1.5)
```

Ця функція починає сесію GPS, надсилаючи відповідні команди AT та отримуючи інформацію про місцезнаходження.

Запускається цикл, що очікує відповідь від модема. В тілі цикла відправляється команда AT+CGPS=1,1 для запуску GPS сесії. Далі очікується відповідь, перевіряється, чи готовий GPS. В разі готовності надсилається команда AT+CGPSINFO для отримання даних про місцезнаходження. Це повторюється кожні 1,5 секунди.

Настпупна функція відповідає за надсилання команди AT до модема через послідовний порт та очікування відповіді:

```
def sendAt(command,back,timeout):
ser.write((command+'\r\n').encode())
time.sleep(timeout)
if ser.inWaiting():
     time.sleep(0.01 )
     rec_buff = ser.read(ser.inWaiting())
if rec_buff != '':
     if back not in rec_buff.decode():
          print(command + ' ERROR')
          print(command + ' back:\t' + rec_buff.decode())
          return 0
     else:
          print(rec_buff.decode())
          return 1
else:
     print('GPS is not ready')
     return 0
```

Функція приймає три параметри:

– command: команда AT, яку потрібно надіслати.

– back: очікувана відповідь на команду.

– timeout: час очікування відповіді.

Відправляється команда AT через послідовний порт. В середині циклу перевіряється наявність відповіді в буфері. Якщо відповідь містить очікуваний рядок back, повертається 1, інакше виводиться помилка та повертається 0.

Вимикається модуль функцією:

```
def powerDown():
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(6,GPIO.OUT)
print('SIM7600X is loging off:')
GPIO.output(6,GPIO.HIGH)
time.sleep(3)
GPIO.output(6,GPIO.LOW)
```

```
time.sleep(18)
```

Тут записується високий рівень на GPIO для вимкнення модема. Далі йде затримка на 3 секунди, потім знижується рівень GPIO. Після цього відбувається ще одна затримка на 18 секунд для повного вимкнення модема.

Для управління модулем LSM9DS1 використовується бібліотека Jetson-Nano--LSM9DS1. Ця бібліотека вже має налаштування для роботи модуля та Jetson Nano. Також повинна бути встановлена бібліотека smbus. Сам модуль підключається через перший i2c канал:

SCL – Pin 5

SDA – Pin 3

Спочатку створюється екземпляр акселерометра:

```
device = accelerometer.Accelerometer()
```

Дані акселерометра отримуються у форматі кортежу(x,y,z) за допомогою:

```
device.readAccData()
```

Дані гіроскопа отримуються так само, але за допомогою функції:

```
device.readGyroData()
```

Дані магнітометра отримуються функцією:

```
device.readMagData()
```

Щоб налаштувати Slamtec RPLIDAR A3 та отримувати з нього дані використовується бібліотека rplidar-roboticia.

Спочатку створюється об'єкт класу лідара та йде підключення до самого лідара:

lidar = PyRPlidar(port="/dev/ttyUSB0", baudrate=256000, timeout=3)

В значенні port вказується порт, до якого підключений лідар. Значення baudrate містить в собі швидкість читання даних.

RPLIDAR використовує для зв'язку нетекстовий протокол на основі пакетів двійкових даних з хост-системами. І всі пакети, передані по каналу інтерфейсу, поділяються уніфіковані формати пакетів. Сеанс зв'язку завжди ініціалізується головною системою, тобто MCU, ПК тощо. Сам RPLIDAR не надсилатиме жодних даних автоматично після ввімкнення.

Якщо пакет даних надсилається з хост-систем до RPLIDAR, такий пакет називається Request. Коли RPLIAR отримує запит, він надсилає хост-системі дані пакет називається Response. RPLIDAR почне виконувати пов'язані операції, необхідні хост-системі, лише після отримання запиту.

Для отримання стану пристроя використовується команда `get_health()`. Коли основна система виявляє деякі потенційний ризик, який може спричинити апаратний збій у майбутньому, повернуте значення стану буде "Попередження". Але сенсор все одно може працювати як зазвичай. Коли датчик знаходиться в стані зупинки, значення status буде "Error". У разі статусів попереджень або помилок буде повернуто ненульовий код помилки.

Для отримання сканування використовується:

```
for i, scan in enumerate(lidar.iter_scans()):
    print('%d: Got %d measurments' % (i, len(scan)))
    print(scan)
    if i > 10:
        break
```

Функція `lidar.iter_scans()` має два параметри: max_buf_meas та min_len. Перший параметр містить в собі максимальну кількість вимірювань, що зберігаються всередині буфера. Другий параметр містить в собі мінімальну кількість вимірювань у скануванні.

В циклі переглядається перелік вимірювань. Кожне вимірювання є кортежем з наступним форматом: якість, кут, відстань.

Для отримання даних з модуля BM101 mmWave використовується бібліотека mmWave.

Спочатку імпортується клас srradar з самої бібліотеки. Також треба імпортувати Jetson.GPIO:

```
from mmWave import srradar
import Jetson.GPIO as GPIO
```

Далі потрібно підключитися до serial monitor:

```
port = serial.Serial("/dev/ttyTHS1",baudrate = 921600, timeout = 0.5)
```

Також створюється об’єкт класа srradar:

```
srr = srradar.SRR(port)
```

Далі можна отримати дані:

```
(dck,v1,v2,v3,v4) = srr.tlvRead()
```

dck – data check. Приймає значення 1 або 0. Значення 1 – дані готові, 0 – дані не готові. Ця змінна містить в собі інформацію про наявність даних.

v1 – Виявлений об’єкт: містить дальність, кут, (X,Y) координати та доплерівську інформацію об’єктів, які бачить пристрій mmWave.

v2 – Об'єкт кластера: містить кількість виявлених об'єктів, (X,Y) координати та (X,Y) розмір.

v3 – TRACK Дані: містить кількість виявлених об'єктів, (X,Y) координати, (X,Y) швидкість для доплера, (X,Y) розмір, дальність та доплерівську інформацію об’єктів, які бачить пристрій mmWave.

v4 – повертає масив float для системи допомоги при паркуванні.

3.3 Альтернативи отримання даних

Багато з модулів, такі як gps, лідар, радар, компас, коштують дорого. Деякі з них можна замінити звичайним мобільним телефоном. Наприклад мобільний телефон має свій gps, акселерометр, гіроскоп, магнітометр. Таким чином телефон можна використовувати для отримання даних про місцезнаходження. Є декілька способів передачі даних з мобільного телефону на пристрій linux: bluetooth, usb, wi-fi з’єднання. Перші два способи потребують складного програмного забезпечення, тому вирішено передавати дані за допомогою клієнта та сервера. Також телефон може стати точкою доступа і Jetson Nano отримає доступ до інтернету.

Для початку потрібно створити застосунок для телефону, який буде отримувати дані gps, акселерометра, гіроскопа, магнітометра. Також застосунок запускає TCP-сервер і далі всім підключеним клієнтам буду відправляться дані про місцезнаходження.

Застосунок для Android створюється в програмном середовищі розробки Andoid Studio. Мовою програмування є Java.

Спочатку в Android Studio створюється пустий проект. В ньому обирається мінімальна версія SDK. Для того щоб застосунок підходив для більшості пристроїв обрано API 23, ця версія підтримується на 98,2% пристроїв. В якості мови конфігурації збірки обрано Kotlin DSL.

В файлі build.gradle.kts слід вказати залежності:

```
dependencies {
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.8.0")

implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation("androidx.navigation:navigation-
fragment:2.5.3")
    implementation("androidx.navigation:navigation-ui:2.5.3")
    implementation ("com.google.android.gms:play-services-
location:18.0.0")
    implementation ("com.google.code.gson:gson:2.8.8")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-
core:3.5.1")
}
```

Спочатку створюємо клас GPSManager який дозволяє запитати дозвіл на доступ до GPS, а потім отримувати координати через функцію requestLocationUpdates(). Якщо дозвіл не наданий, він запросить його.

```
public void requestLocationUpdates() {
        if (ActivityCompat.checkSelfPermission(activity,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
            // Перевірка дозволу на доступ до GPS. Якщо дозволу
немає, потрібно запросити його.
```

```
                ActivityCompat.requestPermissions((Activity)
activity,    new    String[]{Manifest.permission.ACCESS_FINE_LOCATION},
REQUEST_LOCATION_PERMISSION);
            } else {
                // Якщо є дозвіл, запускаємо отримання координат.

fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, Looper.myLooper());
            }
        }
```

Для отримання напрямку на основі даних з акселерометра та магнітометра створюємо клас HeadingCalculator. В цьому класі використовується SensorManager, який отримує дані з акселерометра та магнітометра мобільного телефона. В функції startListening() отримуються акселерометр, магнітометр та запускаються слухачі для отримання даних:

```
    public void startListening() {
            Sensor                    accelerometer                    =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            Sensor                    magnetometer                     =
sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
            sensorManager.registerListener(this,       accelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
            sensorManager.registerListener(this,        magnetometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
```

Наступна функція отримані дані з використанням синусів та косинусів перетворює в градуси та зберігає ці дані:

```
    public void onSensorChanged(SensorEvent event) {
            if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                System.arraycopy(event.values, 0, accelerometerValues,
0, 3);
            }      else      if      (event.sensor.getType()      ==
Sensor.TYPE_MAGNETIC_FIELD) {
```

```
                System.arraycopy(event.values, 0, magnetometerValues,
0, 3);
            }

            if (SensorManager.getRotationMatrix(rotationMatrix, null,
accelerometerValues, magnetometerValues)) {
                SensorManager.getOrientation(rotationMatrix,
orientationValues);
                float azimuthInRadians = orientationValues[0];
                float        azimuthInDegrees        =        (float)
Math.toDegrees(azimuthInRadians);

                // Обчислення напрямку (heading) в градусах (від 0 до
360 градусів)
                float heading = azimuthInDegrees;
                if (heading < 0) {
                    heading += 360f;
                }
                azimuth = heading;
                // Отриманий напрямок (heading) знаходиться в змінній
'heading'
            }
        }
```

Уже в класі MainActivity використовуються ці два створені класи, а також створюється та запускається сервер. В цьому класі ініціалізується основна робота застосунку. Створюється кнопка яка буде запускати сервер. При повторному натисканні на кнопку сервер закривається. Коли сервер запустився буде показуватися IP адреса та порт, за допомогою яких клієнт зможе підключитися. Також запускаються слухачі для отримання орієнтації в просторі та місцезнаходження.

```
    protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
```

```
            gpsManager = new GPSManager(this);
            gpsManager.requestLocationUpdates();
            stopServerButton = findViewById(R.id.stopServerButton);
            startServerButton = findViewById(R.id.startServerButton);
            ipTextView = findViewById(R.id.ipTextView);
            portTextView = findViewById(R.id.portTextView);
            sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
            locationHelper = new HeadingCalculator(sensorManager);
            locationHelper.startListening();
            startServerButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    startServer();
                    startServerButton.setVisibility(View.GONE);
                    stopServerButton.setVisibility(View.VISIBLE);
                }
            });
            stopServerButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    stopServer();
                    startServerButton.setVisibility(View.VISIBLE);
                    stopServerButton.setVisibility(View.GONE);
                    ipTextView.setVisibility(View.GONE);
                    portTextView.setVisibility(View.GONE);
                }
            });
        }
```

В функції startServer() відбувається запуск сервера та очікується з'єднання з клієнтом. Якщо клієнт приєднався то йому відправляються дані з датчиків:

```
      while (isServerRunning) {
          // Перевірка, чи попередній клієнт від'єднався
          try {
             float azimuth = locationHelper.getAzimuth();
             runOnUiThread(new Runnable() {
                @Override
                public void run() {
                   gpsManager.requestLocationUpdates();
                }
             });
             double latitude = gpsManager.getLatitude();
             double longitude = gpsManager.getLongitude();
             ServerData serverData = new ServerData(latitude, longitude,
azimuth);
             Gson gson = new Gson();
             // Отправка азимуту клієнту
             String jsonData = gson.toJson(serverData);
             outputStream.write(jsonData.getBytes());
             Thread.sleep(1000);
          }catch (SocketException e) {
             // Помилка при з'єднанні, вважати, що клієнт від'єднався
             try {
                clientSocket.close(); // Закрити з'єднання
             } catch (IOException ex) {
                ex.printStackTrace();
             }
             // Очікувати нове з'єднання
             clientSocket = serverSocket.accept();
             outputStream = clientSocket.getOutputStream();
             } catch (IOException e) {
                e.printStackTrace();
             }
          }
          serverSocket.close();
```

Клієнту надсилаються дані формату json, які містять довготу, широту та азимут.

Після цього весь проект потрібно скомпілювати та зібрати в Android Studio. Буде створений apk файл, який потрібно перенести на мобільний телефон та запустити його. Застосунок буде встановлений та готовий для роботи. Він представляє собою одну кнопку, яка запускає та зупиняє сервер. Адреса сервера буде написана над кнопкою після запуску.

В ролі клієнта виступає Jetson Nano. TCP-клієнт для Jetson Nano та отримання даних:

```
def get_data():
    global recording, path, seconds
    while True:
        s = socket.socket()
        host = '192.168.0.101'
        port = 9999
        s.settimeout(5)
        try:
            s.connect((host, port))
            while True:
                string = ''
                while recording:
                    data = s.recv(1024)
                    if not data:
                        print("Сервер закрив з'єднання")
                        break
                    try:
                    decoded_data = json.loads(data.decode('utf-8'))
                        string += str(seconds) + ' : ' +
str(decoded_data['azimuth'])
                    except:
                        pass
            s.close()
        except ConnectionRefusedError:
```

```
                print("Не вдалося підключитися до сервера. Спробуємо
ще раз через 5 секунд.")
                time.sleep(5)
            except Exception as e:
                print("Сталася помилка:", e)
```

В цьому коді відбувається підключення до сервера. Після підключення отримуються дані json. Якщо дані не отримуються, то виникає помилка що означає що сервер закрив з'єднання.

Використовуючи API Google Maps можна отримати координати точки призначення. Маючи координати місцезнаходження платформи за допомогою цього API можна прорахувати маршрут до точки призначення. Цей маршрут буде складатись з координат точок, по яким буде рухатися транспорт.

Спочатку отримуються координати місця призначення за його назвою:

```
geocode_result = gmaps.geocode(place)
```

Отримання місця знаходження пристрою за допомогою координат з gps:

```
reversegeocode_result=gmaps.reverse_geocode((prev_lat,prev_long))
```

Розрахування маршруту до місця призначення з допомогою google maps:

```
results=gmaps.directions(reverse_geocode_result[0]['formatted_add
ress'],geocode_result[0]["formatted_address"],mode="driving",arrival_t
ime=datetime.now()+timedelta(minutes=0.5))
```

Маючи координати точки до якої наразі повинен рухатися транспорт, можна визначити напрямок руху. За допомогою бібліотеки Geodesic визначаємо азимут між двома точками:

```
def get_azimuth(lat1,lat2,long1,long2):
  brng = Geodesic.WGS84.Inverse(lat1,long1,lat2,long2)['azi1']
  if brng < 0:
  brng = 360 + brng
  return brng
```

Таким чином транспорт буде рухатися від однієї точки до іншої. Для того щоб транспорт не збивався з маршруту, постійно буде розраховуватися напрямок між точкой призначення і поточним мізцезнаходженням.

3.4 Розпізнавання об’єктів

Розпізнавання об'єктів - це важлива складова автономних систем, що дозволяє ідентифікувати та класифікувати об'єкти на зображеннях або відео. Одним з найефективніших підходів до цього завдання є застосування нейронних мереж, які навчаються розпізнавати об'єкти на основі великої кількості прикладів.

Нейронні мережі - це математичні моделі, які імітують роботу людського мозку. Вони складаються з нейронів, які з'єднані між собою шляхом ваг, та шарів, які обробляють вхідні дані та генерують вихід. Існує кілька типів нейронних мереж, зокрема зворотні мережі, згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та їх поєднання.

Нейронні мережі працюють шляхом подачі вхідних даних на вхідний шар, після чого вони проходять через різні шари мережі, кожен з яких обробляє та витягує корисну інформацію. У процесі тренування нейронної мережі, ваги між нейронами налаштовуються так, щоб мережа правильно класифікувала вхідні дані.

У звичайній нейронній мережі є три типи шарів:

Вхідні шари: Це шар, на якому ми вводимо нашу модель. Кількість нейронів у цьому шарі дорівнює загальній кількості ознак у наших даних (кількість пікселів у випадку зображення).

Прихований шар: вхідні дані з вхідного шару потім подаються в прихований. Прихованих шарів може бути багато, залежно від моделі та розміру даних. Кожен прихований шар може мати різну кількість нейронів, яка, як правило, більша, ніж кількість ознак. Вихідні дані з кожного шару обчислюються шляхом матричного множення вихідних даних попереднього шару з вагами цього шару, що навчаються, а потім додаванням зсувів, що навчаються, з подальшою функцією активації, яка робить мережу нелінійною.

Вихідний шар: Вихідні дані з прихованого шару потім подаються в логістичну функцію, таку як сигмоїд або софтмакс, яка перетворює вихідні дані кожного класу в оцінку ймовірності кожного класу.

Дані подаються в модель, і вихідні дані з кожного шару виходять з наведеного вище кроку, який називається feedforward, потім ми обчислюєця похибка за допомогою функції похибки, деякі поширені функції помилок - це перехресна ентропія, помилка втрати квадрата тощо. Функція помилок вимірює, наскільки добре працює мережа. Після цього ми розповсюджуємо назад у модель, обчислюючи похідні. Цей крок називається зворотним поширенням, який в основному використовується для мінімізації втрат.

Згорткова нейронна мережа (CNN) — це розширена версія штучних нейронних мереж, яка переважно використовується для вилучення функції з сіткоподібного матричного набору даних. Наприклад, візуальні набори даних, такі як зображення або відео, де шаблони даних відіграють велику роль.

Згорткова нейронна мережа складається з кількох рівнів, таких як вхідний рівень, згортковий шар, агрегувальні шари та повністю пов'язані шари.

Згортковий шар застосовує фільтри до вхідного зображення для вилучення об'єктів, шар «об'єднання» зменшує дискретизацію зображення, щоб зменшити обчислення, а повністю зв'язаний шар робить остаточне з’єднання.

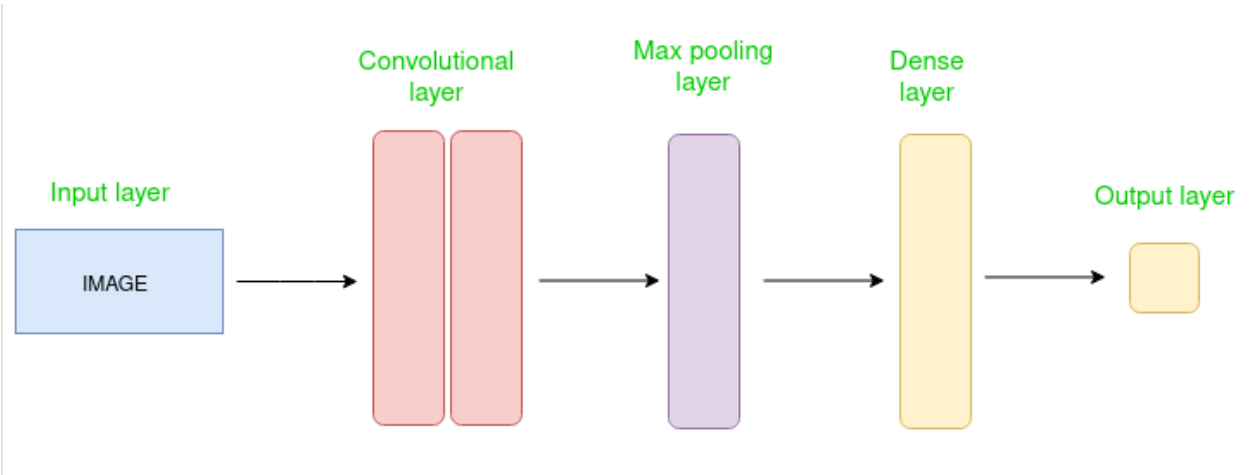


Рис. 3.2 – Архітектура CNN

Зображення представляється у вигляді кубоїда що має свою довжину, ширину (розмір зображення) і висоту (тобто канал, оскільки зображення

зазвичай мають червоний, зелений і синій канали). Далі береться невелика ділянка цього зображення і запускається на ньому невелика нейронна мережа, яка називається фільтром або ядром, з K-виходами. Тепер ця нейронна мережа проводиться по всьому зображенню, в результаті ми отримаємо інше зображення з різною шириною, висотою і глибиною. Замість каналів R, G і B тепер у нас більше каналів, але менша ширина і висота. Ця операція називається згорткою. Якщо розмір патча збігається з розміром зображення, це буде звичайна нейронна мережа.

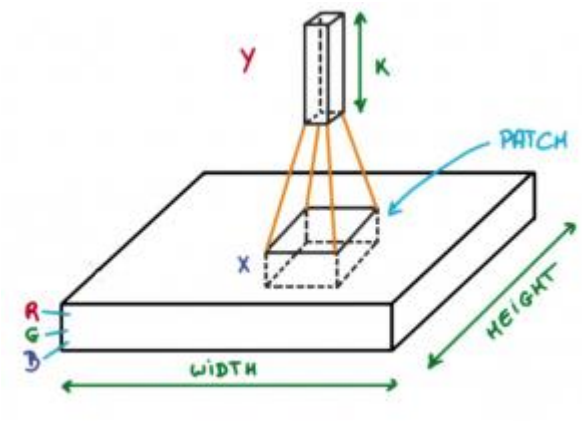


Рис. 3.3 – Принцип роботи згорткового шару

Основна функція агрегувального шару полягає в зменшенні розміру об'єму, що робить обчислення швидкими, зменшує пам'ять, а також запобігає перенавчанню. Двома поширеними типами шарів є максимальне агрегування та середнє агрегування. Наприклад, максимізаційне агрегування використовує максимальне значення з кожного з кластерів нейронів попереднього шару. Іншим прикладом є усереднювальне, що використовує усереднене значення з кожного з кластерів нейронів попереднього шару.

Повноз'єднані шари з'єднують кожен нейрон одного шару з кожним нейроном наступного шару.

YOLO (You Only Look Once) відноситься до згорткових нейронних мереж (CNN). Він є однією з найпопулярніших архітектур для розпізнавання об'єктів в реальному часі. YOLO використовує згорткові шари для виявлення об'єктів на зображеннях та відео. Його особливість полягає в тому, що він

розділяє зображення на сітку і застосовує одну модель для всіх клітин цієї сітки, що дозволяє значно прискорити процес розпізнавання об'єктів.

YOLO працює дуже швидко, оскільки вона використовує один покроковий процес для виявлення об'єктів, у порівнянні з іншими архітектурами, які можуть вимагати кількох проходів через зображення.

Остання версія — YOLOv8, яка була розроблена компанією Ultralytics. На python є зручна бібліотека, яка називається ultralytics. Ця бібліотека надає зручний інструментарій для навчання та роботи нейронної мережі.

Для навчання нейронної мережі знадобиться dataset з різними об’єктами. Існує кілька джерел, де можна знайти готові датасети для YOLO або інших моделей об'єктного виявлення:

– COCO (Common Objects in Context): Це один з найпопулярніших датасетів для об'єктного виявлення. Він містить більш як 200 тисяч зображень та понад 80 категорій об'єктів.

– Open Images Dataset: Цей датасет містить мільйони зображень, розмічених з різними класами об'єктів.

Тренування відбувається за допомогою наступного коду:

```
from ultralytics import YOLO
model = YOLO('yolov8n.pt')
results = model.train(data='coco128.yaml', epochs=100, imgsz=640)
```

Спочатку завантажується претренована модель. В третьому рядку вказується файл yaml датасету, кількість епох для тренування та розмір зображення(всі зображення будуть підганятися під цей розмір).

Тренування буде відбуватись швидше на GPU. Для того щоб викоритсовувати GPU, в pytorch повино бути встановлене CUDA. Щоб встановити CUDA треба зайти на офіційний сайт pytorch і там обрати конфігурацію своєї системи і після цього з’явиться спосіб для встановки.

Щоб використати GPU треба додати такий код:

```
import torch
model.to('cuda')
```

З використанням GPU нейронна мережа буде працювати ефективніше.

Запуску YOLO та отримання даних виконується за допомогою наступного коду:

```
from ultralytics import YOLO
import cv2
import torch
from ultralytics.utils.plotting import Annotator
model = YOLO('yolov8m.pt')
model.to('cuda')
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)
while True:
    _, img = cap.read()
    results = model.predict(img)
    for r in results:
        annotator = Annotator(img)
        boxes = r.boxes
        for box in boxes:
            b = box.xyxy[0]#get box coordinates in (left, top, right, bottom) format
            c = box.cls
            annotator.box_label(b, model.names[int(c)])
    img = annotator.result()
    cv2.imshow('YOLO V8 Detection', img)
    if cv2.waitKey(1) & 0xFF == ord(' '):
        break
cap.release()
cv2.destroyAllWindows()
```

В цьому коді використовується opencv для отримання зображення з камери, а також виведення зображення з уже обробленими даними. Запускається цикл, в кожній ітерації якого отримується зображення з камери та оброблюється нейронною мережею. З даних отримуються координати точок на зображенні, з яких формується квадрат, в середині якого знаходиться

виявленний об’єкт. Також в даних містяться індекс об’єкта та відсоток на скільки впевнена в типі цього об’єкта нейронна мережа.

Annotator використовується для того щоб на основі отриманих даних намалювати прямокутник на зображенні для того щоб побачити що розпізнала нейронна мережа.

Розмір зображення встановлено 640×480 пікслеів, так як нейронна мережа навчалася на зображеннях з таким розміром.

Зазвичай ці дані представлені у вигляді списку з кортежів або структур. За допомогою цієї інформації ми можемо впевнено визначити та відобразити об'єкти на відео та виконати додаткові дії, наприклад, відстежування об'єктів або подальший аналіз.

Слід зазначити про проблеми з версіями. Бібліотека ultralytics доступна на версії python вище 3.8, в той час як на Jetson Nano встановлена версія 3.6.9. Можна встановити новішу версію, наприклад 3.10. Однак виникає проблема з pytorch cuda. Встановити на Jetson Nano можна лише уже скомпільовані версії, які можна завантажити з сайта nvidia. Однак версія JetPack також повинна співпадати. І на Jetson Nano є лише версія Jetpack 4, яка має версію pytorch 1.8.0, в той час коли для ultralytics треба версія більше 2.0.0. І встановити потрібну скомпільовану версія pytorch не можливо.

Тут є два способи вирішення цієї проблеми:

– Використовувати нейронну мережу для розпізнавання об’єктів SSD mobile.

– Зробити клієнт та сервер. Jetson Nano відправляє на сервер зображення. Зображення буде оброблюватися нейронною мережею на сервері. Отримані дані відправляються назад до Jetson Nano.

SSD mobile працює на Jetson Nano і з нею не виникають проблеми з версіями. Однак вона розпізнає об’єкти трохи гірше за YOLO.

В другому способі можуть виникать збої зі зв’язком і з цим буде виникать затримка передачі відео. Однак при хорошому зв’язку об’єкти розпізнаються дуже добре.

Обидва способи мають свої переваги та недоліки, тому можна використовувати два способи одночасно. Наприклад при поганому зв’язку використовувати перший спосіб, а при хорошому зв’язку використовувати другий спосіб.

3.5 Висновки

У цьому розділі було детально розглянуто програмну частину інтегрованої системи для автономного маршрутного планування та навігації у транспортних системах. Розглянуто керування кожним з модулів системи, зокрема моторами та колесами, ультразвуковим датчиком, GPS, акселерометром, магнітометром, гіроскопом, лідаром та радаром.

Для керування моторами та колесами використовуються відповідні сигнали керування для кожного з них, забезпечуючи потрібний рух та поворот рухомого транспорту. Ультразвуковий датчик дозволяє виявляти перешкоди та уникати них, а GPS забезпечує інформацію про місцезнаходження рухомого транспорту.

Акселерометр, магнітометр та гіроскоп надають дані про орієнтацію транспорта у просторі, що допомагає в навігації та виборі оптимального маршруту. Лідар та радар використовується для виявлення об'єктів навколо автомобіля та уникнення зіткнень.

Крім того, розглянуто розпізнавання об'єктів з камери за допомогою нейронної мережі YOLO. Ця технологія дозволяє ефективно та точно виявляти об'єкти на зображеннях та відео, що допомагає в реалізації системи автономного маршрутного планування та навігації.

Загалом, програмна частина системи виконує важливі функції керування та обробки даних, що дозволяє забезпечити ефективну та безпечну автономну навігацію в транспортних системах.

4 АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Результати роботи компонентів та їх особливості

Одним з основних критеріїв ефективності нейронної мережі є її здатність точно виявляти та класифікувати об’єкти на зображенні. Висока точність дозволяє системі надійно реагувати на навколишнє середовище та уникати небезпеки.

Нейронна мережа YOLO, проявила себе добре в багатьох ситуаціях, забезпечуючи швидке та точне розпізнавання об'єктів на зображеннях. Вона здатна ефективно виявляти різноманітні об'єкти, включаючи автомобілі, пішоходів, дорожні знаки тощо, що є важливим для задач автономного водіння та інших застосувань.

Рис. 4.1 – Результат роботи YOLO

Проте, важливо відзначити, що можуть траплятися труднощі в певних умовах, таких як низька якість зображення, обмеженість освітлення або велика кількість фонового шуму. В таких умовах точність розпізнавання може знизитися, а навіть стати недостатньою для ефективного використання системи.

Також, варто відзначити, що нейронні мережі мають тенденцію до недооцінки або неправильного розпізнавання деяких об'єктів, особливо якщо вони виявляються на зображеннях у великій кількості або знаходяться у складних сценах.

Також слід відзначити те, що оскільки YOLO не може працювати на Jetson Nano і через це обробка даних з зображення відбувається на сервері, інколи можуть виникать затримка пов’язана з поганим зв’язком. В цьому випадку система може переключитися на розпізнавання об’єктів за допомогою моделі SSD MobileNet, яка оброблює дані з зображення на Jetson Nano.

SSD MobileNet демонструє високу швидкість роботи та низьку вимогливість до обчислювальних ресурсів, що робить його ідеальним вибором для реалізації вбудованих та мобільних систем розпізнавання об'єктів. Він ефективно впорався з багатьма завданнями розпізнавання об'єктів у реальному часі, забезпечуючи швидке та точне виявлення об'єктів на зображеннях.

Рис. 4.2 – Результат роботи SSD Mobile Net

Проте, SSD MobileNet має деякі обмеження порівняно з YOLO. Зокрема, він може бути менш точним у виявленні дрібних об'єктів або об'єктів зі

складною формою.Також, він може виявляти труднощі з обробкою складних сцен або зображень з великою кількістю фонового шуму.

Ультразвуковий датчик надійно виявляє неподвижні об'єкти із високою точністю, дозволяючи уникати зіткнень та забезпечувати безпеку руху в обмежених просторах.

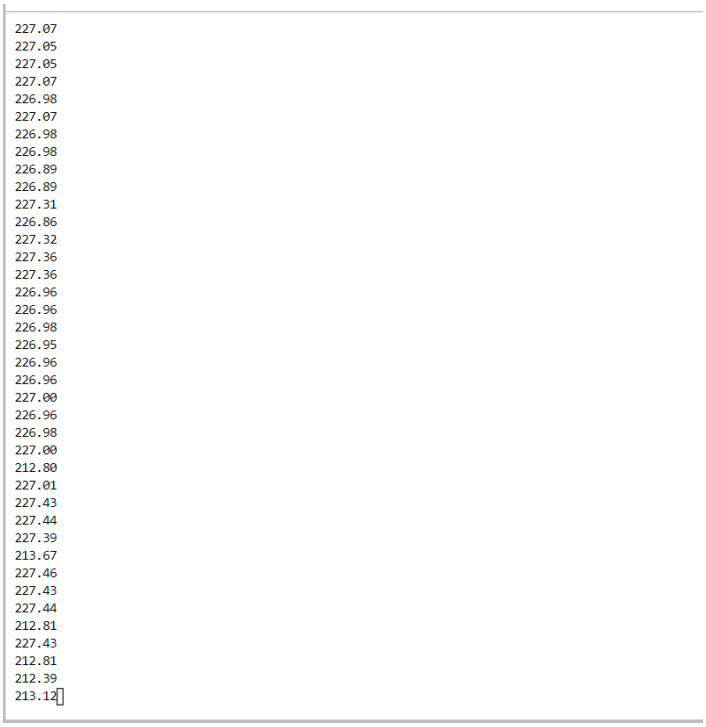


Рис. 4.3 – Значення отримані з ультразвукового датчика

Особливості роботи ультразвукового датчика включають його високу швидкість відгуку та можливість використання в реальному часі. Він працює на основі принципу відбиття звукових хвиль від об'єктів та вимірювання часу їхнього повернення, що дозволяє визначати відстань до об'єктів.

Ультразвуковий датчик може мати проблеми, наприклад, при вимірюванні відстаней до м'яких або непрозорих поверхонь, а також у випадку відбиття сигналу від різних матеріалів або при наявності водяного пару в повітрі. Також важливо враховувати обмежену дальність дії ультразвукового датчика, яка може бути недостатньою для виявлення об'єктів на великих відстанях. При близькій дистанції(до 2 метрів) дистанція виявляєтсья точно, при більш великих дистанціях(більше 2 метрів) точність падає і датчик надає

різні показники. Це залежить більш від якості дачтика. Великий розкид значень слід узагальнювати і брати вищі значення. Також, якщо перешкоди немає, то датчик буде давати значення -1.

Також слід зазначити роботу Arduino, якщо плата не оригінальна, то можу виникать труднощі з отриманням даних з serial monitor. І тому спочатку треба виконати підключення до порта декілька раз.

Компас забезпечує високу точність визначення азимуту та може ефективно коригувати орієнтацію в реальному часі, що дозволяє точно навігуватися та вибирати оптимальний маршрут.

Важливою особливістю є необхідність калібрування компаса для забезпечення точності його роботи та уникнення помилок у визначенні орієнтації.

Проте, компас може стикатися з проблемами, таких як сильні магнітні поля або металеві конструкції у навколишньому середовищі. У таких ситуаціях може виникати ризик помилкового визначення орієнтації або неправильного визначення напрямку руху, що може призвести до недостатньо точної навігації та вибору маршруту. Модуль слід причепити на платформі подалі від металевих деталей, інакше показники будуть не точними.

GPS забезпечує швидку та точну інформацію про географічні координати транспорта. Важливо враховувати, що точність визначення місцезнаходження може залежати від кількості видимих супутників, а також від наявності перешкод, що перешкоджають сигналу.

У місцях з високими будівлями або густим лісом, сигнал GPS може бути блокований або викривлено, що призводить до неточності визначення місцезнаходження. Також, у глибоких долинах або підземних тунелях сигнал може бути втрачений зовсім, що може ускладнити навігацію.

Також треба враховувати холодний та гарячий старт.

При холодному старті GPS-приймач запускається після тривалого періоду вимкнення або відсутності сигналу. Під час холодного старту приймач повністю втрачає свої попередні дані про супутники та їхні орбіти. У такому

випадку, для отримання першого фіксації місцезнаходження приймач повинен здійснити повний пошук та залочення на сигнали супутників. Час, необхідний для холодного старту, може бути значною кількістю хвилин.

При гарячому старті GPS-приймачу вже відомі деякі дані про супутники та їхні орбіти. Оскільки приймач має попередні дані, для отримання першого фіксації місцезнаходження він може швидше залочитися на сигнали супутників. Час, необхідний для гарячого старту, зазвичай значно менший, порівняно з холодним стартом, і може бути декілька секунд або навіть менше.

Таким чином, при першому включені пристрою знадобиться деякий час для встановлення зв’язку з супутниками.

```
Azimuth: 2.676133
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 2.4167542
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 0.079464406
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 1.6844838
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 2.6182783
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 2.013781
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 2.2966723
Longitude: 31.000024
Latitude: 47.628725

Azimuth: 2.4236248
Longitude: 31.000024
Latitude: 47.628725
```

Рис. 4.4 – Значення напрямку та координат

4.2 Навігація за допомогою штучного інтелекту

Маючи вхідні дані з усіх компонентів можна створити штучний інтелект, який на основі цих даних буде будувати маршрут. Першим кроком у

створенні нейронної мережі для будування маршруту є підготовка вхідних та вихідних даних. Для цього потрібно зібрати дані з усіх необхідних датчиків:

– Дистанція з ультразвукового датчика.
– Координати об'єктів на зображенні та їх клас.
– Поточне значення швидкості та повороту.
– Напрямок руху та бажаний напрямок руху.

Необхідно за допомогою контролера вручну керувати рухомою платформою, оминаючи перешкоди та рухатися до місця призначення. При цьому потрібно щоб дані записувалися в реальному часі. Для кожного параметра повинен призначатися час, для того щоб вони не переплуталися.

Дані повинні бути відформатовані та підготовлені для подальшого використання у навчанні моделі. Наприклад, дистанція з ультразвукового датчика може представлена у вигляді одного числа, а інформація про об'єкти на зображенні може бути у вигляді координат та класів об'єктів, значення швидкості та повороту може бути від -1 до 1, напрямок руху визначається числом(всього 360 градусів). Також важливо розділити дані на тренувальний та валідаційний набори для подальшої оцінки продуктивності моделі.

Далі необхідно створити модель нейронної мережі з декількома вхідними шарами, кількома прихованими шарами та вихідними шарами. Архітектура моделі:

– Вхідні шари: Для кожного типу вхідних даних (ультразвуковий датчик, координати об'єктів, поточна швидкість, поточний поворот, напрямок руху, бажаний напрямок руху) використовується окремий вхідний шар.

– Повнозв'язані шари: Вхідні дані перетворюються за допомогою повнозв'язаних шарів, які реалізують функції активації ReLU. Шари Dense з активацією ReLU використовуються для взаємодії між різними типами вхідних даних та внутрішніми представленнями моделі.

– Вихідні шари: Модель має два вихідні шари: один для передбачення швидкості іншій для передбачення повороту. Для передбачення швидкості використовується вихідний шар з одним нейроном та активацією Sigmoid. Для

передбачення повороту використовується вихідний шар з одним нейроном та активацією Tanh.

Кожен тип вхідних даних (ультразвуковий датчик, координати об'єктів, поточна швидкість, поточний поворот, напрямок руху, бажаний напрямок руху) подається на відповідний вхідний шар моделі. Вхідні дані перетворюються за допомогою повнозв'язаних шарів з активацією ReLU для створення внутрішнього представлення. Внутрішні представлення даних об'єднуються та подаються на вихідні шари. Вихідні шари генерують передбачення для швидкості та повороту.

Шари Dense (або повнозв'язані шари) - це найпоширеніший тип шарів у нейронних мережах. Вони складаються з набору нейронів, кожен з яких пов'язаний з кожним нейроном попереднього та наступного шару. У шарах Dense кожен вхідний сигнал з попереднього шару зважується вагами та передається через функцію активації до наступного шару.

ReLU - це функція активації, яка використовується для нелінійної активації шарів нейронних мереж. Вона визначається як:

$$f(x) = \max(0, x), \tag{4.1}$$

де x – вхідне значення.

Функція ReLU активує нейрони з від'ємними значеннями на нуль, тим самим здійснюючи нелинійне перетворення сигналу та дозволяючи нейронним мережам навчитися складним не лінійним залежностям у вхідних даних.

Модель будується наступним чином:

```
def build_model():
    ultrasonic_distance_input = Input(shape=(1,))
    object_info_input = Input(shape=(None, 5))
    current_speed_input = Input(shape=(1,))
    current_turn_input = Input(shape=(1,))
    movement_dir_input = Input(shape=(1,))
    des_movement_dir_input = Input(shape=(1,))
```

```
        distance_flat = layers.Flatten()(ultrasonic_distance_input)
        current_speed_flat = layers.Flatten()(current_speed_input)
        current_turn_flat = layers.Flatten()(current_turn_input)
        movement_dir_flat = layers.Flatten()(movement_dir_input)
        des_move_dir_flat= layers.Flatten()(des_movement_dir_input)
        merged_inputs = layers.concatenate([ultrasonic_distance_flat,
    layers.GlobalAveragePooling1D()(object_info_input),
                                            current_speed_flat,
                                            current_turn_flat,
                                            movement_dir_flat,
                                            des_move_dir_flat])
        x = layers.Dense(64, activation='relu')(merged_inputs)
        x = layers.Dense(64, activation='relu')(x)
        output_speed = layers.Dense(1, activation='sigmoid')(x)
        output_turn = layers.Dense(1, activation='tanh')(x)
    model=Model(inputs=[ultrasonic_distance_input,object_info_input,c
urrent_speed_input,current_turn_input,movement_direction_input,desired
_movement_direction_input], outputs=[output_speed, output_turn])
        return model
```

Тепер, коли є побудована модель, потрібно навчити її на основі даних. Тренування моделі відбувається наступним чином:

```
    history = model.fit(
        x=[X_train[:, 0], X_train[:, 1], X_train[:, 2], X_train[:, 3],
X_train[:, 4], X_train[:, 5]],
        y=[y_train_speed, y_train_turn],
        validation_data=(
            [X_val[:, 0], X_val[:, 1], X_val[:, 2], X_val[:, 3],
X_val[:, 4], X_val[:, 5]],
            [y_val_speed, y_val_turn]
        ),
        epochs=10)
```

В функції fit вказуються вхідні та вихідні дані для тренування, а також для валідації. Також вказується кількість епох. Кількість епох визначає,

скільки разів весь тренувальний набір даних буде проходитися через нейронну мережу під час тренування. Кожен цикл, коли весь тренувальний набір даних пройшов через мережу один раз, називається епохою.

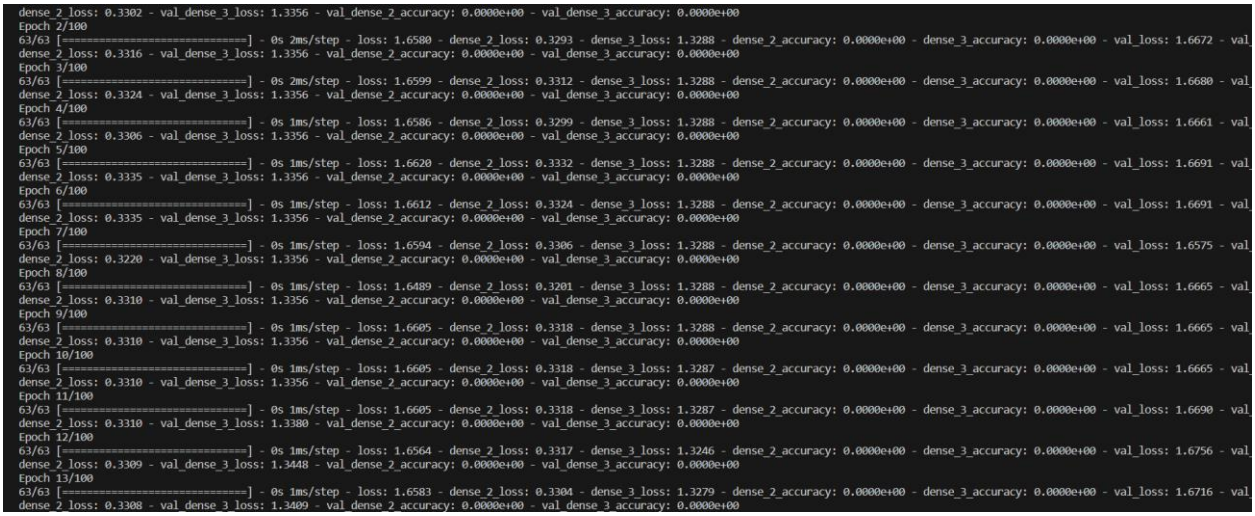


Рис. 4.5 – Навчання нейронної мережі

Загалом мережа навчалась на не великій кількісті даних і потребує більше часу та даних для навчання. Нейронна мережа пройшла 100 епох і тренерувалася на більш чим 1000 даних. Точність швидкості: 0.0971272. Точність повороту: 0.0894122. Точність є досить низькою і модель вимагає подальшого навчання. Додавання більше даних для тренування може допомогти моделі краще узгодитися з патернами у вхідних даних і покращити її точність.

4.3 Безпека та експлуатація системи

Перед кожним використанням системи слід перевірити стан всіх її компонентів, зокрема, Jetson Nano, GPS, модуль компаса, камера та ультразвуковий датчик. Бути впевненим, що всі підключення щільні та надійні, а всі системи працюють належним чином.

Завжди треба мати можливість ручного керування транспортом. У разі будь-яких непередбачених ситуацій або втрати зв'язку з системою навігації завжди треба бути готовим взяти керування транспортом на себе.

Необхідно уникати використання системи автономної навігації та маршрутного планування у складних або небезпечних умовах, таких як сильний дощ, туман, снігопад, погана якість доріг або обмежена видимість. Завжди треба дотримуватися правил безпеки на дорозі та пам'ятати, що система може не завжди розпізнати всі небезпеки на шляху.

Після того як система була увімкнена, треба зачекати деякий час поки увімкнуться всі компоненти за запуститься усе необхідне програмне забезпечення. Модулю GPS треба час для підключення до супутників.

Важливо щоб система завжди була підключена до мережі, тому використовувати систему слід в населених пунктах в місцях, де є хороший зв’язок. Оскільки система ще не є досконалою, необхідно уникати місць які можуть перешкоджувати доступу до зв’язку з мережами.

4.4 Висновки

Кожен з компонентів виявив хорошу точність у виконанні своїх функцій. GPS забезпечує точне визначення місцезнаходження, модуль компаса – правильну орієнтацію, а камера та ультразвуковий датчик – виявлення перешкод на шляху. Однак у кожного з компонентів також є свої недоліки, які варто враховувати під час експлуатації системи.

Наприклад, GPS може виявляти неточності у зоні згущення будівель або високих гірських масивів, модуль компаса може бути чутливим до електромагнітних перешкод, а камера та ультразвуковий датчик можуть мають обмежену точність у поганих погодних умовах або з обмеженою видимістю.

Описана модель нейронної мережі, яка використовує дані з цих компонентів, дозволяє системі ефективно управляти платформою та уникати перешкод на шляху. Це підвищує рівень автономності та безпеки руху системи.

Ретельне дотримання правил безпеки та інструкцій з експлуатації допомагає забезпечити надійну та безпечну роботу системи, зменшуючи ризик виникнення аварійних ситуацій та зберігаючи цілісність обладнання.

ВИСНОВКИ

У даній магістерській роботі була розроблена інтегрована система для автономного маршрутного планування та навігації на базі штучного інтелекту для мобільної робототехніки, що дозволить створення ефективної, безпечної та адаптивної системи, яка забезпечить оптимальне функціонування автономних роботів, зокрема робіт-кур'єрів, у різноманітних умовах.

У першому розділі було розглянуті аналоги, вимоги та складові системи. Було з’ясовано, що для роботи системи використовуються камери, лідари, радари, алгоритми машинного навчання та технології взаємодії з оточенням для отримання та обробки інформації про навколишнє середовище. Важливі для безпечної роботи системи є висока точність сенсорів та надійність їх функціонування.

У другому розділі було розглянуто особливості та принципи роботи кожного компонента, способи їх підключення. Кожен компонент виконує важливу функцію у забезпеченні роботи системи.

У третьому розділі було розглянуто програмну частину та керування кожним з модулів системи, зокрема моторами та колесами, ультразвуковим датчиком, GPS, акселерометром, магнітометром, гіроскопом, лідаром та радаром.

У четвером розділі було продемонстровано роботу компонентів, виявлено їхні недоліки. Була описана модель нейронної мережі, яка використовує дані з цих компонентів.

У загальному можна зазначити, що система автономної навігації та маршрутного планування представляє собою потужний інструмент для розвитку автоматизованих транспортних систем. Вона забезпечує не поганий рівень автономності та безпеки руху, що дозволяє використовувати її в різних областях, включаючи автомобільну промисловість, логістику, робототехніку та інші сфери. Однак система все ще вимагає вдосконалення апаратної та програмної частини.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. López, J., Sanchez-Vilariño, P., Cacho, M. D., & Guillén, E. L. Obstacle avoidance in dynamic environments based on velocity space optimization. Robotics and Autonomous Systems. 2020. DOI: 10.1016/j.robot.2020.103569.

2. Hertz L, Krogh A, Palmer RG. Вступ до Theory of Neural Computing Addison Wesley. 1991.

3. Керекеслер М. О., Крайник Я. М. Інтегрована система для автономного маршрутного планування і навігації у транспортних системах. Інформаційні технології та інженерія : Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів : тези доп., 31 січня – 2 лютого 2024 р. / ЧНУ імені Петра Могили. Миколаїв, 2024. С. 79–82.

4. What is Tesla Autopilot and how does it work?. *Pocket-lint* : вебсайт. URL: https://www.pocket-lint.com/what-is-tesla-autopilot-features-what-does-it-do/ (дата звернення: 12.12.2023).

5. Waymo's Self-Driving Cars: How They Work. *Lifewire* : вебсайт. URL: https://www.lifewire.com/waymo-self-driving-cars-4171314 (дата звернення: 12.12.2023).

6. Learning Infrastructure and Versioning Control Platform for Self-Driving Vehicles. *Uber* : вебсайт. URL: https://www.uber.com/en-UA/blog/machine-learning-model-life-cycle-version-control/ (дата звернення: 12.12.2023).

7. Apollo 3.0: Entering the New Era of Autonomous Driving. *Medium* : вебсайт. URL: https://medium.com/apollo-auto/apollo-3-0-entering-the-new-era-of-autonomous-driving-d781bc769cef (дата звернення: 12.12.2023).

8. How Autonomous Vehicles Work: the Self-Driving Stack. *Mobileye* : вебсайт. URL: https://www.mobileye.com/blog/autonomous-vehicle-day-the-self-driving-stack/ (дата звернення: 13.12.2023).

9. Real-Time Kinematic and Differential GPS. *E-education* : вебсайт. URL: https://www.e-education.psu.edu/geog862/node/1828/ (дата звернення: 13.12.2023).

10. Get Started With Jetson Nano Developer Kit. *Nvidia* : вебсайт. URL: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit (дата звернення: 14.12.2023).

11. Waveshare JetRacer AI Kit. *Waveshare* : вебсайт. URL: https://www.waveshare.com/wiki/JetRacer_AI_Kit (дата звернення: 14.12.2023).

12. About Slamtec RPLIDAR, MAPPER and Slamware. *Seeedstudio* : вебсайт. URL: https://www.seeedstudio.com/blog/2019/08/05/all-you-need-to-know-about-slamtec-rplidar-mapper-and-slamware/ (дата звернення: 14.12.2023).

13. mmWave Radar for Automotive and Industrial Applications. URL: https://www.ti.com/content/dam/videos/external-videos/2/3816841626001/5675916489001.mp4/subassets/Mmwave_webinar_Dec2017.pdf/ (дата звернення: 14.12.2023).

14. Kargin, A., & Ivaniuk, O. Autonomous robot motion control situational planning model. *Advanced Information Systems*. 2020. Vol. 4. No. 3. P. 41–51. DOI:10.20998/2522-9052.2020.3.05.

15. Oliveira, M., Castro, A., Madeira, T., Pedrosa, E., Dias, P., & Santos, V. A ROS framework for the extrinsic calibration of intelligent vehicles: A multi-sensor, multi-modal approach. Robotics and Autonomous Systems. 2020. DOI: 10.1016/j.robot.2020.103558.

16. How HC-SR04 Ultrasonic Sensor Works & Interface It With Arduino. URL: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/ (дата звернення: 14.12.2023).

17. Khnissi, K., Ben Jabeur, C., & Seddik, H. A smart mobile robot commands predictor using recursive neural network. *Robotics and Autonomous Systems*. 2020. Vol. *131*. DOI: 10.1016/j.robot.2020.103593.

18. SIM7600G-H 4G / 3G / 2G / GNSS Module for Jetson Nano, LTE CAT4, Global Applicable. URL: https://www.waveshare.com/sim7600g-h-4g-for-jetson-nano.htm (дата звернення: 18.12.2023).

19. LSM9DS1 Breakout Hookup Guide. URL: https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide/all (дата звернення: 18.12.2023).

20. Introduction to Convolution Neural Network. URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (дата звернення: 18.12.2023).

21. Ultralytics YOLOv8 Docs. URL: https://docs.ultralytics.com/models/yolov8/ (дата звернення: 20.12.2023).

22. Getting Started with the Low-Cost RPLIDAR Using NVIDIA Jetson Nano. URL: https://collabnix.com/getting-started-with-the-low-cost-rplidar-using-jetson/ (дата звернення: 24.12.2023).

23. How to Use GPIO Pins on Jetson Nano Developer Kit. URL: https://maker.pro/nvidia-jetson/tutorial/how-to-use-gpio-pins-on-jetson-nano-developer-kit (дата звернення: 24.12.2023).

24. Training & evaluation with the built-in methods. URL: https://keras.io/guides/training_with_built_in_methods/ (дата звернення: 28.12.2023).

25. Keras 3 API documentation / Layers API / Layer activation functions. URL: https://keras.io/api/layers/activations/ (дата звернення: 28.12.2023).

26. Three Ways to Build Machine Learning Models in Keras. URL: https://machinelearningmastery.com/three-ways-to-build-machine-learning-models-in-keras/ (дата звернення: 28.12.2023).

27. Training & evaluation with the built-in methods. URL: https://www.tensorflow.org/guide/keras/training_with_built_in_methods/ (дата звернення: 28.12.2023).

ДОДАТОК А

ДОВІДКА

про перевірку на унікальність пояснювальної записки кваліфікаційної магістерської роботи

на тему: «Інтегрована система для автономного маршрутного планування і навігації у транспортних системах»

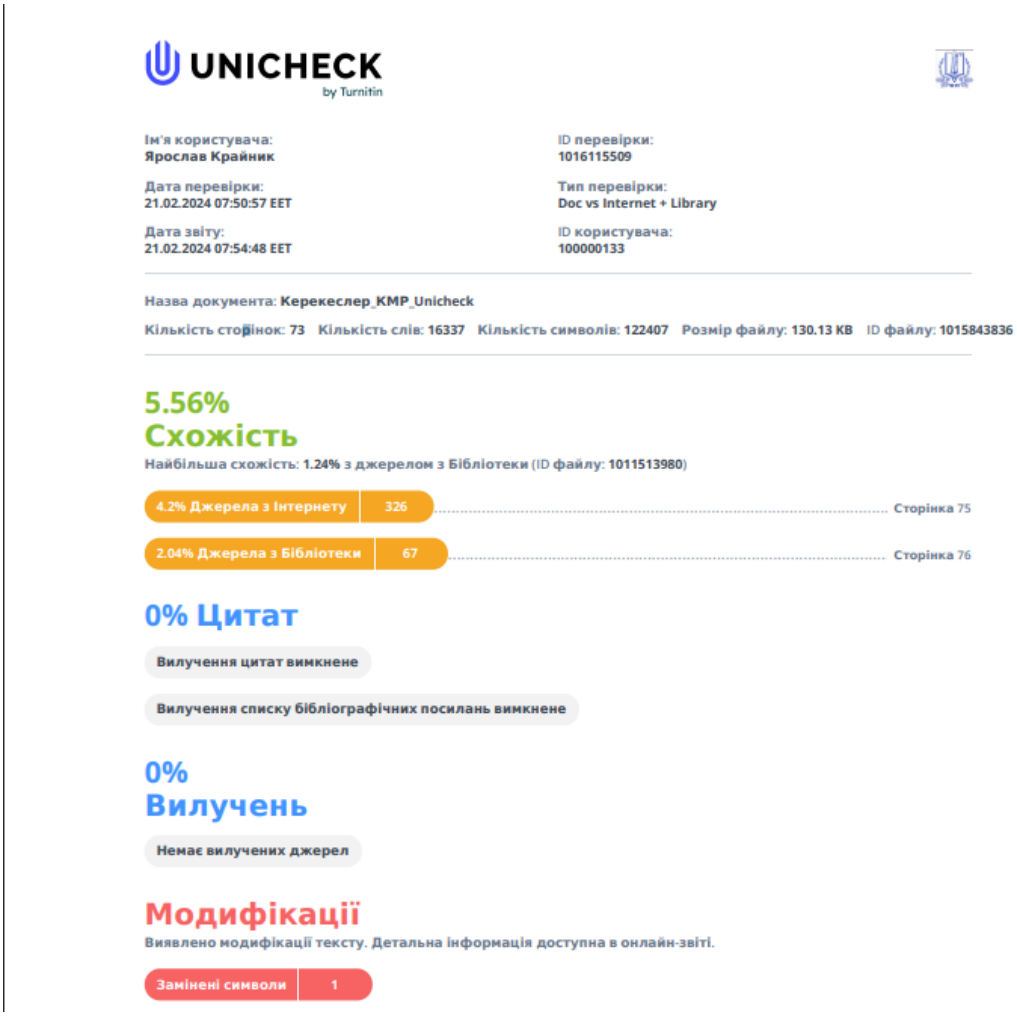
студента спеціальності 123 «Комп'ютерна інженерія», групи 605м

Керекеслер Максим Олегович

прізвище, ім'я, по-батькові

Перевірку тексту здійснено сервісом: онлайн-сервіс Unicheck

Результат перевірки тексту кваліфікаційної роботи: схожість складає 5,56 %.



UNICHECK
by Turnitin

Ім'я користувача: Ярослав Крайник
ID перевірки: 1016115509
Дата перевірки: 21.02.2024 07:50:57 EET
Тип перевірки: Doc vs Internet + Library
Дата звіту: 21.02.2024 07:54:48 EET
ID користувача: 100000133

Назва документа: Керекеслер_КМР_Unicheck
Кількість сторінок: 73 Кількість слів: 16337 Кількість символів: 122407 Розмір файлу: 130.13 KB ID файлу: 1015843836

5.56%
Схожість
Найбільша схожість: 1.24% з джерелом з Бібліотеки (ID файлу: 1011513980)

4.2% Джерела з Інтернету 326 Сторінка 75
2.04% Джерела з Бібліотеки 67 Сторінка 76

0% Цитат
Вилучення цитат вимкнене
Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень
Немає вилучених джерел

Модифікації
Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.
Замінені символи 1

Студент: Керівник:

_____ М. О. Керекеслер _____ Я. М. Крайник
підпис ініціали, прізвище підпис ініціали, прізвище

Дата: «___» _____ 2024 р.

ДОДАТОК Б

Лістінг коду для розпізнавання

```
from ultralytics import YOLO
import cv2
import torch
from ultralytics.utils.plotting import Annotator  # ultralytics.yolo.utils.plotting is deprecated
print(torch.cuda.is_available())
model = YOLO('yolov8m.pt')
model.to('cuda')

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

while True:
    _, img = cap.read()

    results = model.predict(img)

    for r in results:

        annotator = Annotator(img)

        boxes = r.boxes
        for box in boxes:

            b = box.xyxy[0]  # get box coordinates in (left, top, right, bottom) format
            c = box.cls
            annotator.box_label(b, model.names[int(c)])

    img = annotator.result()
    cv2.imshow('YOLO V8 Detection', img)
    if cv2.waitKey(1) & 0xFF == ord(' '):
        break

cap.release()
cv2.destroyAllWindows()
```

ДОДАТОК В

Лістінг коду для отримання даних

```
import socket, json, time, serial, cv2, asyncio
from aiohttp import ClientSession, WSMessage, WSMsgType
import numpy as np
from threading import Thread
from jetracer.nvidia_racecar import NvidiaRacecar
IP = '192.168.0.105'
car = NvidiaRacecar()
car.throttle_gain = 0.6
error = 0.15
car.steering_offset = 0.18
car.steering = 0
speed = 0.4
gas = False
back = False

capture_device = 0
capture_width = 1640
capture_height = 1232
capture_fps = 30
width = 640
height = 480
quality = 60
azimuth = 0
long = 0
lat = 0

ultrasonic_data = 0
def get_coord_azimuth():
    global azimuth
    global long
    global lat
    while True:
        s = socket.socket()
        host = '192.168.0.106'
        port = 9999
        s.settimeout(5)
        try:
            s.connect((host, port))
            while True:
                string = ''
                while True:
                    data = s.recv(1024)
                    if not data:
                        print("Сервер закрив з'єднання")
                        break
                    decoded_data = json.loads(data.decode('utf-8'))
                    print('Azimuth: '+str(decoded_data['azimuth']))
```

```
                print('Longitude: '+str(decoded_data['longitude']))
                print('Latitude: '+str(decoded_data['latitude'])+ '\n \n')
                azimuth = decoded_data['azimuth']
                long = decoded_data['longitude']
                lat = decoded_data['latitude']

            s.close()
        except ConnectionRefusedError:
            print("Не вдалося підключитися до сервера. Спробуємо ще раз через 5 секунд.")
            time.sleep(5)
        except Exception as e:
            print("Сталася помилка:", e)

def get_range():
    ser = serial.Serial(port='/dev/ttyUSB0', baudrate=9600)
    while True:
        if ser.in_waiting > 0:
            data = ser.readline().decode('utf-8').strip()
            print(data)

capture = 'nvarguscamerasrc sensor-id=%d ! video/x-raw(memory:NVMM), width=%d, height=%d, format=(string)NV12, framerate=(fraction)%d/1 ! nvvidconv ! video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! videoconvert ! appsink' % (
                capture_device, capture_width, capture_height, capture_fps, width, height)

async def receive_data():
    async with ClientSession() as session:
        async with session.ws_connect('http://{}:8080/ws'.format(IP)) as ws:
            print('connected')
            while True:
                msg = await ws.receive()
                if msg.type == WSMsgType.TEXT:
                    data = msg.json()
                    print("Received data from server:", data)
                elif msg.type in (WSMsgType.CLOSED, WSMsgType.CLOSING):
                    break

def main():
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(receive_data())

async def send_video():
    cap = cv2.VideoCapture(capture, cv2.CAP_GSTREAMER)  # Відкриваємо камеру за допомогою OpenCV
    while True:
        try:
            async with ClientSession() as session:
                async with session.ws_connect('http://{}:8080/ws'.format(IP)) as ws:
                    while True:
                        ret, frame = cap.read()  # Зчитуємо кадр з камери
                        if not ret:
```

```
                print('0camera')
                break
            #frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #gray
            # Перетворюємо кадр в формат байтів
            height, width = frame.shape[:2]

            frame = frame[:height-100, :]
            encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), quality]
            _, buffer = cv2.imencode('.jpg', frame, encode_param)
            data = buffer.tobytes()
            #print(len(frame.tobytes()))
            #print(len(data))

            # Відправляємо кадр на сервер через WebSocket
            await ws.send_bytes(data)

    except Exception as e:
        print(f"Client 1: {str(e)}")
        await asyncio.sleep(5)  # Затримка перед повторним підключенням
def drive():
    global gas
    global back
    global speed
    while 1:
        for event in events:
            if event.code == "ABS_X":
                car.steering = ((event.state - 127)/127) + error
            if event.code == "ABS_GAS":
                if event.state > 0:
                    gas = True
                if event.state == 0:
                    gas = False
            if event.code == "ABS_BRAKE":
                if event.state > 0:
                    back = True
                if event.state == 0:
                    back = False

            if gas and not back:
                car.throttle = speed
            if back and not gas:
                car.throttle = -speed*2
            if gas and back:
                car.throttle = 0
            if not gas and not back:
                car.throttle = 0

if __name__ == '__main__':
    t1 = Thread(target = main)
    t1.start()
    loop = asyncio.get_event_loop()
    loop.run_until_complete(send_video())
```

ДОДАТОК Г

Лістінг коду для навчання нейронної мережі

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, Input, Model
def build_model():
    # Вхідні дані
    ultrasonic_distance_input = Input(shape=(1,))
    object_info_input = Input(shape=(None, 5))  # Кількість об'єктів не відома
    current_speed_input = Input(shape=(1,))
    current_turn_input = Input(shape=(1,))
    movement_direction_input = Input(shape=(1,))
    desired_movement_direction_input = Input(shape=(1,))

    ultrasonic_distance_flat = layers.Flatten()(ultrasonic_distance_input)
    current_speed_flat = layers.Flatten()(current_speed_input)
    current_turn_flat = layers.Flatten()(current_turn_input)
    movement_direction_flat = layers.Flatten()(movement_direction_input)
    desired_movement_direction_flat = layers.Flatten()(desired_movement_direction_input)

    merged_inputs = layers.concatenate([ultrasonic_distance_flat,
                                layers.GlobalAveragePooling1D()(object_info_input),
                                current_speed_flat,
                                current_turn_flat,
                                movement_direction_flat,
                                desired_movement_direction_flat])

    x = layers.Dense(64, activation='relu')(merged_inputs)
    x = layers.Dense(64, activation='relu')(x)

    # Вихідні дані
    output_speed = layers.Dense(1, activation='sigmoid')(x)  # Вихідна швидкість (від 0 до 1)
    output_turn = layers.Dense(1, activation='tanh')(x)  # Вихідний поворот (від -1 до 1)

    # Побудова моделі
    model = Model(inputs=[ultrasonic_distance_input, object_info_input,
                        current_speed_input, current_turn_input,
                        movement_direction_input, desired_movement_direction_input],
                outputs=[output_speed, output_turn])
    return model

model = build_model()
model.compile(optimizer='adam', loss=['mse', 'mse'], metrics=['accuracy'])
# Виведення структури моделі
model.summary()
history = model.fit(x=X_train_data,y=y_train_data,
    validation_data=(X_val_data, y_val_data), epochs=100, batch_size=128)
```

ДОДАТОК Д

Публікації за темою роботи

Міністерство освіти і науки України
Чорноморський національний університет
імені Петра Могили

«Інформаційні технології та інженерія»

Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів

ТЕЗИ

31 січня – 2 лютого 2024 року

Миколаїв – 2024

даних;

3) наявність інтерфейсів та периферії. Мікроконтролер ESP32 має ряд інтерфейсів та периферійних пристроїв, які сприяють легкій інтеграції з різними сенсорами та пристроями;

4) ефективність та низьке енергоспоживання. ESP32 дозволяє ефективно використовувати енергію, що важливо для багатьох застосувань IoT, де пристрої можуть працювати на батарейках;

5) спільнота та підтримка. Існує велика активна спільнота розробників, що робить ESP32 привабливим вибором, адже можна знайти багато ресурсів та допомоги в Інтернеті.

ESP32 є потужним та гнучким рішенням для розробки систем керування віддаленими IoT-пристроями.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Getting Started with ESP8266 NodeMCU Development Board. URL: https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/ (Last accessed: 30.12.2023).

2. ESP32 Pinout Reference: Which GPIO pins should you use? URL: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/ (Last accessed: 30.12.2023).

УДК 629.1.07

Керекеслер М. О., Крайник Я. М.

Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна

ІНТЕГРОВАНА СИСТЕМА ДЛЯ АВТОНОМНОГО МАРШРУТНОГО ПЛАНУВАННЯ І НАВІГАЦІЇ У ТРАНСПОРТНИХ СИСТЕМАХ

Автономні роботи досліджувалися протягом тривалого періоду, і зростання попиту на автоматизовані рішення для різних життєвих сфер прискорило вивчення цієї області. Такі проблеми пов'язані з домашніми завданнями (роботи-пилососи та газонокосарки), застосунками безпеки (спостереження, розвідка) та промисловими операціями (виробництво та логістика) тощо. За останні кілька років ринок онлайн-покупок значно зріс, і тепер роздрібні та поштові компанії активно працюють над тим, щоб зробити автономну доставку роботами реальністю.

Безпілотні мобільні роботи найчастіше можуть використовуватися в сегменті логістичних транспортних перевезень.

Використання безпілотних транспортних засобів є ефективним, оскільки ці пристрої можуть значно знизити вартість доставки у порівнянні з участю людей, що призводить до загального зниження собівартості надання послуг та сприяє поширенню таких технологій.

Автономні транспортні системи, такі як роботи-кур'єри, можуть пересуватись як наземним шляхом, так і в повітрі. Автономні наземні роботи та дрони мають ряд своїх переваг та недоліків.

Переваги наземних автономних транспортних систем:

– стабільність. Наземні транспортні системи мають стійку базу на поверхні, що сприяє їхній стабільності в різних погодних умовах та на різних типах доріг;

– ефективність в міському середовищі. Для міських територій наземні транспортні системи можуть бути більш практичним та ефективним в порівнянні з повітряними, оскільки вони можуть легко взаємодіяти з існуючою дорожньою інфраструктурою;

– більша вантажопідйомність. Наземні транспортні засоби, можуть мати велику вантажопідйомність, що робить їх ефективними для перевезення великих обсягів товарів;

– безпека. Ані вантаж, ані транспортний засіб не призведуть до травматичних наслідків для оточуючих, якщо виникне аварійна ситуація.

Недоліки наземних автономних транспортних систем:

– трафік та затори. Наземні транспортні системи часто стикаються з проблемами трафіку та заторами, особливо у великих містах, що може призводити до затримок у доставці;

– залежність від інфраструктури. Наземні транспортні системи вимагають розвиненої дорожньої інфраструктури, також їхні можливості обмежені якістю та доступністю доріг.

Переваги повітряних автономних транспортних систем:

– прямий шлях. Повітряні транспортні системи можуть прокладати прямий шлях від точки А до точки Б, уникаючи дорожніх завад та заторів;

– швидкість. Повітряні транспортні засоби можуть працювати на великих швидкостях, забезпечуючи швидку та ефективну доставку;

– можливість функціонування поза інфраструктурою. Вони не обмежені традиційною дорожньою інфраструктурою, що дозволяє їм об'їжджати перешкоди і швидко пересуватися в разі надзвичайних ситуацій.

Недоліки повітряних автономних транспортних систем:

– безпека. Висока ймовірність конфліктів та безпекових питань, таких як зіткнення в повітрі, потенційна небезпека для пішоходів та інших учасників руху;

– енергоспоживання. Повітряні транспортні системи зазвичай витрачають більше енергії через необхідність підтримання повітряного підйому та інших аеродинамічних факторів.

Штучний інтелект (ШІ) грає ключову роль у вдосконаленні систем автономного маршрутного планування та навігації в транспортних системах, привносить інновації та покращення у ряд аспектів. Використання штучного інтелекту в системах для автономного маршрутного планування та навігації розширює можливості та покращує ефективність автономного транспорту, роблячи його більш адаптивним, безпечним та ефективним у великому спектрі умов. Штучний інтелект використовує алгоритми машинного навчання для аналізу величезних обсягів даних про трафік. На основі цього аналізу система може прогнозувати поточні та майбутні умови руху, допомагаючи у визначенні оптимальних маршрутів. Системи автономного маршрутного планування можуть використовувати машинне навчання для визначення контексту навколишнього середовища, враховуючи інші транспортні засоби, пішоходів, дорожні знаки та інші елементи. Алгоритми штучного інтелекту дозволяють системам для автономного маршрутного планування адаптуватися до змінних умов на дорозі, таких як ремонти, аварії або погодні умови. Це забезпечує більшу ефективність та безпеку руху.

Автономне маршрутне планування і навігація в транспортних системах в основному грунтуються на використанні різноманітних сенсорів, алгоритмів обробки даних та штучного інтелекту для створення безпечного та ефективного руху транспортних засобів. Науково-дослідні установи часто самостійно розробляють або замовляють спеціальні компоненти для створення автономних роботів-кур'єрів. Це включає власні материнські плати обчислювача, камери та сенсори, що призначені для оптимальної ефективності та енергоефективності. Перед початком процесу збирання робота необхідно визначити, яким характеристикам повинні відповідати комплектуючі, і в якому ціновому діапазоні вони мають знаходитися. Зазвичай високоякісні комплектуючі коштують дорого, тому часто проводиться аналіз доступних бюджетних аналогів. Порівнявши альтернативи, обирається оптимальний варіант, який відповідає всім необхідним критеріям.

Для створення системи автономного маршрутного планування та навігації було обрано платформу JetRacer від компанії Nvidia, яка містить модуль Jetson Nano та камеру (рис. 1). Ця платформа створена для розробки багатьох різних енергоефективних систем на базі ШІ, а також забезпечує багатьма різними можливостями.

Рисунок 1 – Платформа JetRacer

Jetson Nano представляє собою компактний і потужний комп'ютер, який здатний виконувати паралельний запуск кількох нейронних мереж в застосунках для класифікації зображень, розпізнавання об'єктів та сегментації. Це ідеальне рішення для швидкого прототипування нового продукту на основі штучного інтелекту та швидкого введення його на ринок. Розмір модуля Jetson Nano менший ніж у кредитної картки, та становить 70 мм х 45 мм. Система на модулі (англ. System-on-a-Module, SoM), яка готова до використання, забезпечує високу продуктивність при впровадженні штучного інтелекту на пристроях різних галузей, від "розумних міст" до робототехніки. Jetson Nano забезпечує обчислювальну потужність в 472 гігафлопси для швидкої обробки алгоритмів штучного інтелекту, дозволяючи паралельно запускати кілька нейронних мереж і обробляти інформацію від різних високоточних датчиків.