Міністерство освіти та науки України

Рівненський державний гуманітарний університет

Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «бакалавр»

на тему:

«3-D симулятор для управління автономними транспортними засобами»

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія
програмного забезпечення»
Губенко Олександр Русланович

Науковий керівник:
викл. Копелюк В.О.

Рівне, 2025

Зміст

1. Вступ

1.1. Актуальність теми

Сучасні реалії розвитку технологій поступово наближають нас до того, що автомобілі можуть самостійно рухатися без постійної участі людини. Це питання сьогодні активно вивчається науковцями та інженерами у всьому світі. Основною причиною такої уваги є зростання потреби у безпечнішому, ефективнішому та доступнішому транспорті.

Однією з головних проблем, що спонукає до таких розробок, є збільшення кількості дорожньо-транспортних пригод, перевантаження доріг та складність у керуванні автомобілем у великих містах. Автономні машини можуть частково або повністю вирішити ці питання, оскільки здатні приймати рішення набагато швидше, ніж людина, і не страждають від втоми чи неуважності.

Проте створення повноцінного самокерованого транспортного засобу вимагає ретельного тестування в різних умовах. Для цього часто використовуються симулятори, які дозволяють моделювати ситуації з максимальною точністю, не ризикуючи ні технікою, ні життям людей. Однак більшість із доступних систем вимагають серйозного обладнання, що обмежує можливості невеликих команд чи навчальних проєктів. Тому тема розробки легкого, адаптивного 3D-симулятора, який можна використовувати на звичайному комп’ютері, залишається дуже актуальною, особливо для закладів освіти та дослідницьких лабораторій.

1.2. Мета дослідження

Метою бакалаврської роботи є створення програмного інструменту у вигляді 3D-симулятора, за допомогою якого можна віртуально протестувати алгоритми автономного керування транспортним засобом, та розробка апаратної частини у вигляді фізичної моделі з мікроконтролером ESP32. Планується поєднати симуляційне середовище та реальний пристрій для вивчення ефективності автономної навігації в різних ситуаціях.

1.3. Завдання дослідження

Для досягнення поставленої мети було сформульовано такі задачі:

- Розглянути існуючі програмні платформи, які використовуються для симуляції безпілотних авто;
- Визначити ключові технічні вимоги до майбутнього симулятора;
- Розробити архітектуру симулятора з використанням середовища Unity;
- Реалізувати базову функціональність: виявлення перешкод, реакція на об'єкти, навігація;
- Створити прототип фізичної автономної машинки, використовуючи доступні компоненти;
- Забезпечити взаємодію між мобільним додатком та мікроконтролером;
- Провести тестування розроблених компонентів у віртуальному та фізичному середовищі;
- Порівняти отримані результати та зробити висновки щодо переваг запропонованого підходу;
- Запропонувати шляхи покращення системи та можливості її використання в освітньому середовищі.

1.4. Об’єкт дослідження

Об’єктом дослідження є процес імітації автономного руху транспортного засобу у віртуальному середовищі та його тестування, а також створення практичної моделі, яка буде автономною.

1.5. Предмет дослідження

Предмет дослідження становлять програмні та апаратні рішення, які забезпечують самостійне керування транспортним засобом, зокрема алгоритми ухилення від перешкод, планування маршруту та взаємодії з оточенням як у симуляторі, так і на реальному пристрої.

2. Огляд сучасних підходів до моделювання автономних транспортних засобів

2.1. Вступ

Сучасні дослідження в галузі автономного транспорту неможливо уявити без етапу віртуального моделювання. Реальні випробування безпілотних автомобілів пов’язане з ризиками, досить дороговартісне та необхіднісю спеціальної інфраструктури. Тому перед виїздом на дороги більшість розробок перевіряються у цифрових середовищах, які дозволяють безпечно протестувати алгоритми, виявити помилки, проаналізувати дані та допрацювати логіку керування.

Симулятори імітують дорожні умови, дії інших учасників руху, роботу сенсорів та інші чинники, з якими стикається транспортний засіб. Завдяки цьому розробники отримують змогу відлагодити систему, не ризикуючи обладнанням та не витрачаючи зайві ресурси.

2.2. Наявні симуляційні платформи для автономних транспортних засобів

У світі вже розроблено декілька програмних рішень, які дозволяють моделювати поведінку автономних автомобілів. Набільш поширені серед них є:

- **CARLA**

Це симулятор з відкритим кодом, зосереджений на міські умови. Підтримує підключення різних сенсорів (LiDAR, GPS, камери), дозволяє створювати складні дорожні сценарії. Має хорошу графіку, але потребує потужного обладнання.

- **AirSim**

Розроблений компанією Microsoft, цей симулятор дозволяє проводити тестування як з наземними транспортними засобами, так і з дронами.

Використовує рушій Unreal Engine, що забезпечує реалістичну фізику. Підтримує мови програмування Python і C++, має відкритий API.

- **LGSVL (тепер SVL Simulator)**

Рішення з підтримкою систем ROS і Apollo, створене спеціально для тестування програм автопілота. Можна налаштовувати різні сенсори та дорожні сценарії, проте середовище потребує певної технічної підготовки для запуску.

- **SUMO**

Цей симулятор більше спрямований на моделювання загального трафіку в містах. Він не забезпечує реалістичної візуалізації, але добре справляється з обчисленням різних маршрутів та моделюванням транспортних потоків. Підходить для аналітики та сценарного тестування.

2.3. Порівняння характеристик існуючих симуляційних платформ

Для зручності ключові характеристики платформ можна звести до таблиці:

Таблиця 2.1

Симулятор	Графіка	Сенсори	Інтеграція	Обчислювальні вимоги	Підтримка сценаріїв
CARLA	Висока	LiDAR, камери, GPS	Python, ROS	Високі	Міський рух, ДТП
AirSim	Висока	Камери, IMU, GPS	Python, C++	Високі	Дрони, автомобілі
LGSVL	Середня	LiDAR, RADAR, камери	ROS, Apollo	Середні	Міський рух, тестування автопілота
SUMO	Низька	Немає	Python, C++	Низькі	Трафік, міський рух

2.4. Проблеми та обмеження

Попри широку функціональність, згадані симулятори мають і свої недоліки:

- Високе навантаження на систему. Потужна графіка та складна фізика змушують користувачів використовувати сильні ПК або сервери.
- Складність налаштування. Деякі платформи мають високий поріг входу, що ускладнює роботу з ними студентам або невеликим командам.
- Недостатня деталізація. Деякі рішення, як-от SUMO, не моделюють сенсори або фізику руху з достатньою точністю, що ускладнює симуляцію реальних умов.

2.5. Висновки до розділу 2

Симулятори є незамінною складової підготовки до запуску автономних транспортних засобів. Кожна з розглянутих платформ має свої переваги, проте в умовах обмежених ресурсів їх застосування може бути не завжди доцільним. Через це виникає потреба у розробленні більш легкого та зручного симулятора, який би поєднував базову функціональність зі зрозумілим інтерфейсом та можливістю запуску на звичайному комп’ютері.

3. Проектування та розробка 3D-симулятора для автономного транспортного засобу

3.1. Вступ до розділу

Розробка симулятора, котрий імітує поведінку автономного автомобіля, є одним із найважливіших етапів створення системи керування без участі людини. Такий інструмент дозволяє виконувати випробування в умовах, наближених до реальних, але без ризику пошкодити техніку чи наражати на небезпеку інших. У цьому розділі детально описано структуру, компоненти та принципи функціювання симулятора, створеного в середовищі Unity.

3.2. Опис архітектури симулятора

Симулятор розроблено на базі популярного рушія Unity. Це середовище часто використовують для створення ігор, але воно також підходить для інженерного моделювання, зокрема автономного транспорту. Головна ідея полягала в тому, щоб поєднати візуальне відображення, фізику руху, віртуальні сенсори й алгоритми управління в єдину функціональну систему.

3.2.1. Чому було обрано Unity

Unity — це безкоштовна (у базовій версії) платформа, яка підтримує створення тривимірного середовища, симуляцію фізичних процесів та написання скриптів. Вона дозволяє:

- швидко розгорнути інтерактивну сцену;
- підключити камери, сенсори, керовані об’єкти;
- протестувати алгоритми у візуальному середовищі.

Окрім того, Unity підтримує мову програмування C#, яка є зручною для реалізації логіки руху та взаємодії з об’єктами на сцені.

3.2.2. Основні складові системи

Графіка:

Створено 3D-сцену з дорогами, перешкодами та автомобілями. Усі об'єкти були розміщені вручну або завантажені з Asset Store.

Фізика:

Застосовано стандартні компоненти Rigidbody (для динаміки) і WheelCollider (для коліс). Це дозволило наблизити поведінку віртуального авто до реального, з урахуванням прискорення, гальмування, інерції.

Сенсори:

Віртуальні датчики (лідар, радар, камери, ультразвукові сенсори) були реалізовані за допомогою променів (Raycast), які імітують сканування простору.

Алгоритми керування:

Реалізовано автономне керування з використанням простого планування траєкторії (наприклад, A*), а також обробку ситуацій, коли на шляху виникають перешкоди. Керування виконується через скрипти.

Інтерфейс:

Сцену доповнено елементами інтерфейсу (UI) для зручного запуску/зупинки симуляції, перегляду статусу та параметрів руху.

3.3. Переваги використання Unity у цій розробці:

1. Підтримка реалістичної графіки:

● Unity має потужні інструменти для створення високоякісної графіки та візуалізацій, що дозволяє створювати реалістичне середовище для тестування.

● Підтримка текстур, освітлення, анімації та реалістичної взаємодії з об'єктами на сцені.

2. Широкі можливості для фізичного моделювання:

● Вбудована фізична модель Unity дозволяє моделювати реалістичні фізичні взаємодії транспортних засобів, їх динаміку і реакцію на різні умови.

3. Підтримка сенсорів та алгоритмів AI:

● Легко інтегруються різні типи сенсорів (камери, LiDAR, RADAR тощо), а також алгоритми автономного управління.

● Можливість використання методів машинного навчання для покращення алгоритмів прийняття рішень.

4. Гнучкість та налаштування:

● Unity підтримує розширення та налаштування під конкретні вимоги симуляції, що робить його ідеальним інструментом для створення складних тестових середовищ.

● Можливість налаштування умов симуляції: погода, трафік, дорожні умови.

5. Інтерфейс для розробників та зворотний зв'язок:

● Легкий доступ до інтерфейсу для тестування алгоритмів, візуалізації результатів і швидкої налагодження роботи алгоритмів автономного управління.

3.4. Недоліки використання Unity для симуляції:

1. Високі вимоги до обчислювальних потужностей:

● Реалістичні графіка та фізика можуть вимагати значних обчислювальних потужностей, особливо при масштабних симуляціях з великою кількістю об’єктів і сценаріїв.

● Для досягнення високої продуктивності потрібні потужні комп'ютери або сервери.

2. Обмеження в моделюванні складних фізичних явищ:

● Хоча Unity надає потужні інструменти для фізичного моделювання, в деяких випадках може бути важко реалізувати дуже складні або специфічні фізичні ефекти (наприклад, моделювання поведінки інтермодальних транспортних засобів).

3. Потрібено мати досвід в програмуванні:

● Для реалізації складних алгоритмів автономного управління та інтеграції сенсорів розробникам необхідно володіти знаннями програмування (особливо на мові C#), що може бути додатковою складністю для початківців.

4. Можливі неточності в сенсорних моделях:

● Моделювання сенсорів, таких як LiDAR і камери, може мати деякі неточності порівняно з реальними даними, що може впливати на результати тестування.

3.5. Висновки

Unity виявився зручним інструментом для створення 3D-симулятора автономного транспортного засобу (автомобіля). Попри деякі обмеження, рушій дозволяє реалізувати необхідний функціонал, візуалізувати роботу алгоритмів та швидко вносити зміни. Такий підхід особливо корисний для навчання, тестування логіки та підготовки до реалізації фізичної моделі автономного транспортного засобу.

4. Реалізація 3D-сцени та сенсорних систем у Unity:

4.1. Побудова віртуально середовища:

Для того щоб симулятор міг якнайточніше відтворювати ситуації, з якими може зіткнутися автономний транспорт у реальному житті, в середовищі Unity було створено спеціальну тривимірну сцену. Основна увага приділялася простоті й чіткості елементів, щоб фокус лишався на роботі алгоритмів керування.

4.1.1. Відкриття Unity та створення проєкту:

● Відкриваємо Unity Hub, та вибираємо останню версію Unity, сумісну з середовищем, натискаєм "Create New Project".

● Вибираємо шаблон 3D (Core), оскільки він забезпечує необхідну основу для тривимірної симуляції.

4.1.2. Налаштування базових параметрів:

● Роздільна здатність: Заходимо в Edit > Project Settings > Player і встановлюємо роздільну здатність відповідно до цільової платформи. Це особливо важливо для продуктивності та візуальної чіткості симуляції.

● Фізика: У Edit > Project Settings > Physics налаштовуємо гравітацію, щоб вона відповідала реальним умовам (-9.81 по осі Y).

● Освітлення: Переходимо до Window > Rendering > Lighting Settings і налаштовуємо освітлення.

4.2. Розробка середовища

4.2.1. Знаходимо та завантажуємо ресурси з Unity Asset Store:

4.2.2. Налаштовуємо фізичні параметри

4.2.3. Розміщуємо об'єкти у сцені

4.2.4. Створюємо поверхню для об’єкта керування

4.2.5. Додаємо перешкоди

4.2.6. Налаштовуваємо камери для симуляції огляду

4.3. Реалізація датчиків

Особлику увагу приділено створенню сенсорної системи, що імітує роботу пристроїв, які встановлюються на автономні транспортні засоби у фізичному світі. Під час розробки віртуального середовища, я застосовув датчики, таких як ультразвукові сенсори, камеру, лідар та радар, для збору інформації про навколишнє середовище. Unity дає змогу реалізувати це за допомогою компонентів і C#-скриптів.

4.4.1. Ультразвукові сенсори

Імітація ультразвукових сенсорів виконана за допомогою променевих трасувань (Raycast). Віртуальний сенсор випускає "промінь" у визначеному напрямку, і якщо він потрапляє на об'єкт, фіксується відстань до нього.

Параметри налаштовуються окремо:

- Кут огляду,
- Максимальна дистанція,
- Частота оновлення.

4.4.2. Камера (віртуальне зображення)

Для моделювання зору автономного авто використовується компонент **Camera,** який фіксує зображення перед транспортом. У майбутньому це дозволить додати обробку зображення (наприклад, через OpenCV) або використати його для глибших досліджень.

4.4.3. Додаткові сенсори

Передбачено можливість додавання інших типів датчиків — наприклад, симуляції лідару (через кілька одночасних променів), або GPS (визначення координат авто в сцені).

4.4. Реалізація логіки руху та уникнення перешкод

Створюємо C#-скрипт для управління рухом автомобіля, який буде обробляти команди прискорення, гальмування та повороту. За основу рошуку шляху беремо алгоритм пошуку A*.

Алгоритм пошуку A* («A зірочка» або англ. «A star») — відноситься до евристичних алгоритмів пошуку. Цей метод використовується щоб знайти найкоротший шлях між двома вершинами графу з додатніми вагами ребер. Він був описаним в 1968 р. Пітером Хартом, Нільсом Нільсоном та Бертрамом Рафаелєм. Алгоритм застосовує допоміжну функцію (евристику), аби спрямувати напрямок пошуку та зменшувати його тривалість. Алгоритм повний в тому сенсі, що завжди знаходить оптимальне рішення, якщо воно є.

Для реалізації автономного керування транспортним засобом з трьома ультразвуковими датчиками попереду та інтегрувати їх з алгоритмом A* для навігації, треба виконати наступні кроки:

- Додати датчики спереду автомобіля в Unity.
- Виконати симуляцію ультразвукових датчиків за допомогою Raycast.
- Змінити траєкторію руху при виявленні перешкод.
- Інтегрувати датчики з алгоритмом A* для динамічної побудови маршруту з урахуванням перешкод. (код виконання наведений в дод.1 та дод.2)

4.5. Додаткові налаштування

Після запуску симулятора на екрані відображається:

- рух транспортного засобу;
- виявлені перешкоди;
- реакції на сенсорні сигнали;
- поточний статус системи (швидкість, напрямок, стан керування).

Завдяки цьому можна бачити, як машина «поводиться» в умовах, які швидко змінюються, і як вона пристосовується до ситуацій, що виникають на шляху.

4.6. Висновки до розділу

У цьому розділі було показано, як у середовищі Unity можна створити повноцінну симуляцію автономного авто з базовим набором сенсорів і реакцією на перешкоди. Отримана система дозволяє проводити тестування без апаратного втручання й стане гарною основою для перевірки алгоритмів автономного керування перед їх впровадженням у фізичну модель.

5. Створення апаратної частини автономного транспортного засобу на базі ESP32

5.1. Вибір платформи для реалізації фізичної моделі

Для побудови фізичної моделі автономного транспортного засобу було обрано мікроконтролер ESP32. Він є одним із найпопулярніших рішень серед розробників завдяки поєднанню доступної ціни, добрих обчислювальних можливостей та вбудованих модулів Wi-Fi та Bluetooth.

Переваги ESP32 у цьому проєкті:

- Підтримка великої кількості цифрових і аналогових входів/виходів;
- Наявність двох ядер, що дозволяє одночасно обробляти декілька задач (наприклад, керування мотором та обробку даних із сенсорів);
- Сумісність із Arduino IDE, що спрощує програмування;
- Можливість з’єднання з мобільним додатком для віддаленого керування.

5.2. Загальна схема системи фізичної моделі

Конструкція, що складається з декількох основних блоків:

- Живлення – два Li-Ion акумулятори типу 2S: один для живлення двигунів, інший для ESP32 і сенсорів, знижений до 5 В через модуль типу LM2596;
- Основний контролер – плата ESP32 DevKit;
- Привод двигуна – використовується модуль BTS7960 для управління основним мотором;
- Керування поворотом – реалізовано через модуль L298N, що керує поворотним механізмом;
- Сенсори – підключено чотири ультразвукових датчиків (US-025), сенсор відстані VL53L0X, гіроскоп MPU6050, компас QMC5883L, GPS-модуль NEO-6M (або мобільний GPS, якщо модуль не ловить супутники);
- Інтерфейс зв’язку – мобільний додаток спілкується з ESP32 через Wi-Fi.

5.3. Керування рухом транспортного засобу

Основний мотор, який відповідає за рух вперед/назад, підключений через драйвер BTS7960. Для керування напрямком і швидкістю використовується пара PWM-сигналів (RPWM та LPWM), які генеруються основним контролером ESP32.

Основна логіка:

- при надходженні команди вперед — активується RPWM;
- для руху назад — LPWM;
- швидкість змінюється шляхом варіювання скважності сигналу (від 0 до 255).

Рух контролюється залежно від даних сенсорів та заданої траєкторії.

5.4. Керування поворотами

Для керування поворотом передніх коліс,використовується двоканальний драйвер L298N, який дозволяє змінювати напрям та швидкість обертання невеликого електродвигуна, який відповідає за поворот.

Підключення здійснюється так:

- IN1/IN2 — визначають напрямок обертання двигуна (вліво або вправо);
- ENA — вхід для керування швидкістю через PWM-сигнал.

Для здійснення повороту вліво активується одна з комбінацій логічних сигналів (наприклад, IN1 = HIGH, IN2 = LOW), для повороту вправо — інша (IN1 = LOW, IN2 = HIGH). Вхід ENA підключено до одного з PWM-виходів ESP32, що дозволяє контролювати не тільки швидкість повороту, а й кут повороту коліс.

Завдяки такому підходудозволяється більш плавно контролювати рух та уникнути різких поворотів. На відміну від релейної комутації, L298N забезпечує стабільну зміну напряму та регульовану швидкість, яка позитивно впливає на точність руху транспортного засобу.

5.5. Система сенсорів

Ультразвукові датчики

Розташовані в передній частині машинки та з боків. Їхня основна функція — виявлення перешкод. Принцип функціювання ґрунтується на вимірюванні часу між надсиланням ультразвукового сигналу та отриманням його відбиття. У такий спосіб обчислюється відстань до об’єкта. Ці датчики ефективні при виявленні великих або плоских предметів на шляху.

VL53L0X

Цей сенсор використовує лазерне випромінювання (ToF-принцип) для точного вимірювання коротких відстаней. Він корисний у вузьких просторах, де ультразвук може давати похибки. Висока точність і швидка обробка роблять його хорошим доповненням до основних сенсорів.

MPU6050

Модуль поєднує в собі гіроскоп та акселерометр. Дозволяє визначати зміну кута нахилу, різкі прискорення або уповільнення, а також фіксувати удари. Його можна використовувати для детекції перекидання або втрати зчеплення з поверхнею.

QMC5883L

Це цифровий магнітний компас, що визначає орієнтацію відносно магнітного поля Землі. У поєднанні з гіроскопом дає змогу системі орієнтуватися в просторі, навіть якщо GPS тимчасово недоступний.

NEO-6M або GPS зі смартфона

GPS-модуль застосовується для визначення координат у глобальній світовій системі. Коли сигнал слабкий (наприклад, у приміщенні), його може замінити GPS-модуль смартфона, який передає дані на ESP32 через Wi-Fi. Такий спосіб зменшує кількість зовнішніх модулів та суттєво спрощує систему.

5.6. Збірка і підключення

Під час створення моделі дотримувалися таких ключових принципів:

- Єдина шина GND: усі заземлення (ESP32, двигуни, сенсори, живлення) поєднані в одну спільну шину, щоб уникнути проблем зі збоєм сигналів;
- Логічне живлення ESP32 та сенсорів стабілізоване через DC-DC перетворювач на 5 В;
- Силове живлення (для двигунів) — окреме, підключене безпосередньо до акумулятора через драйвер BTS7960 (для основного мотора) та L298N (для кермового механізму);
- Керувальні сигнали (PWM та цифрові) заведено через пін-головки ESP32 з урахуванням довжини проводів та допустимого струму;
- Захист компонентів — використано конденсатори на живлення.

5.7. Висновки до розділу 5

Фізичну модель автономного транспортного засобу було зібрано з використанням доступних та простих у втіленні компонентів. Заміна релейної системи повороту на драйвер L298N дозволила поліпшити точність та плавність керування. Усі датчики були інтегровані без конфліктів, система живлення стабільна, а загальна архітектура залишилася зручною для доповнень та модифікацій. Розроблений пристрій повноцінно виконує завдання автономного керування в режимі реального часу та має непогану автономність.

6. Розробка мобільного додатку для керування автономною системою

6.1. Загальна ідея

Для зручності керування автономною машинкою, її моніторингу та налаштувань, було створено спеціальний мобільний додаток для операційної системи Android. Додаток дозволяє передавати команди до ESP32, переглядати поточний стан системи та, за потреби, надсилати координати маршруту чи власноруч керувати рухом.

Мобільне керування є особливо актуальним, коли потрібно змінити маршрут без перепрошивки мікроконтролера або під час первинного тестування автономного режиму.

6.2. Вибір засобів розробки

Для створення додатку було обрано Android Studio — офіційне середовище розробки для Android-пристроїв. Програмування здійснювалось мовою Kotlin, яка на сьогоднішній день вважається основною для створення Android-застосунків. Це середовище дозволило швидко створити інтерфейс, додати необхідну логіку та протестувати програму без додаткового обладнання.

6.3. Функціональні можливості додатку

Додаток було реалізовано з урахуванням мінімалізму — тільки те, що потрібно для роботи з машинкою:

- Підключення по Wi-Fi — встановлення зв’язку з ESP32 через локальну точку доступу або безпосередньо;
- Передача команд — вперед, назад, ліво, право, стоп;
- Режими роботи — ручне керування або автономний рух (перемикання режимів);
- Поточний статус —індикатор сигналу, стан зв’язку;
- Відправка координат — (у разі використання смартфонного GPS) передача геопозиції на мікроконтролер;

- Кешування карти — можливість завантажити карту, щоб потім використовувати її офлайн.

Додаток не перевантажено зайвими елементами, щоб уникнути складності при використанні.

6.4. Архітектура зв’язку

Передача даних між додатком і ESP32 здійснюється через Wi-Fi. ESP32 працює як точка доступу або підключається до маршрутизатора, після чого очікує на команди від смартфона. Комунікація побудована за принципом:

- Клієнт (додаток) → сервер (ESP32);
- Використовується простий HTTP-запит (GET або POST) або WebSocket (за потреби зворотного зв’язку);
- Дані передаються у вигляді коротких текстових команд: "F", "B", "L", "R", "S" (вперед, назад, ліво, право, стоп).

Це забезпечує мінімальні затримки та простоту реалізації.

6.5. Інтерфейс користувача

Інтерфейс було реалізовано в стилі мінімалізму з врахуванням того, що користувач може користуватися додатком під час руху моделі.

На головному екрані розміщено:

- Підключення до ESP32 (по Wi-Fi чи Bluetooth);
- Статус машинки
- Карта (на цій вкладці можна ввести координати (як натисканням на карту, так і через клавіатуру), надіслати або отримати координати з ESP32)

6.6. Переваги такого підходу

Гнучкість: можна оновлювати функціональність без зміни прошивки на ESP32;

Зручність: користувач керує пристроєм звичним смартфоном;

Швидкість тестування: легко перемикатися між режимами;

Візуалізація: можна додати графіки (наприклад, швидкість, відстань до перешкоди тощо).

6.7. Висновки до розділу 6

Мобільний застосунок став зручним засобом зв’язку між користувачем та автономним транспортним засобом. Простий інтерфейс і стабільне передавання команд через Wi-Fi дали змогу випробувати основні функції системи. Такий підхід також відчиняє можливості для майбутніх доповнень: від віддаленого моніторингу до побудови складних маршрутів з використанням GPS-даних.

7. Інтеграція апаратної та програмної частини системи

7.1. Вступ

Після окремої розробки фізичної моделі транспортного засобу та мобільного додатку, ключовим етапом стало їх об’єднання в одну єдину систему. Інтеграція забезпечує взаємодію між усіма компонентами та дає змогу протестувати повноцінну роботу автономного транспортного засобу в реальному середовищі.

7.2. Структура інтеграції

Загальна система включає два основних рівні:

1. **Апаратна частина (ESP32 + сенсори + двигуни)** — реальний пристрій, який спроектований на базі машинки 1:10, що виконує команди та зчитує інформацію з оточення;
2. **Мобільний додаток (Android)** — слугує як засіб взаємодії між користувачем та системою, а також як джерело GPS та компас-даних (у разі використання смартфона замість окремих модулів).

Усі ці частини взаємодіють між собою через Wi-Fi з’єднання та спільні комунікаційні протоколи.

7.3. Передача команд з додатку на ESP32

Мобільний додаток надсилає короткі команди на ESP32. Наприклад:

- "F" – рух вперед,
- "B" – назад,
- "L" – поворот ліворуч,
- "R" – праворуч,
- "S" – зупинка.

Кожна команда обробляється скетчем на ESP32, який активує відповідні виходи, що керують двигуном і поворотним механізмом. Комунікація

відбувається або через HTTP-запити, або через WebSocket-зв’язок для більш швидкого реагування.

7.4. Отримання даних з сенсорів

ESP32 опитує сенсори за заданим інтервалом (кожні 100 мс):

- Ультразвукові датчики — перевірка відстані до перешкод;
- VL53L0X — визначення відстані до об'єктів на ближній відстані з високою точністю;
- MPU6050 — відстеження нахилу, прискорення, можливих зіткнень;
- QMC5883L / GPS — визначення орієнтації та місцеположення (у разі використання).

Отримані дані можуть або впливати на поведінку транспортного засобу напряму, або надсилатись назад до мобільного додатку чи записуватись для подальшого аналізу.

7.5. Реалізація логіки ухилення від перешкод

У автономному режимі керування машинка не просто реагує на команду "їхати вперед", а приймає рішення залежно від сенсорних даних. Якщо на шляху виявляється об'єкт:

- Вимірюється відстань;
- Якщо вона менша за встановлений поріг (наприклад, 20 см) — зупинка;
- Далі проводиться перевірка бокових напрямків — якщо зліва або справа є вільний простір, виконується поворот;
- Після ухилення машинка повертається на попередній маршрут або вибирає новий.

Це дозволяє імітувати базову автономну поведінку без складних алгоритмів навігації.

7.6. Висновки до розділу 7

Інтеграція програмної й апаратної частини дозволила створити повноцінну автономну систему, здатну працювати в реальному середовищі. Завдяки простій архітектурі та гнучкому підходу, систему легко доповнювати — наприклад, додати навігацію за GPS, складнішу логіку ухилення або машинне навчання. Це відкриває широкі перспективи для навчання, досліджень і подальших розробок.

8. Аналіз результатів роботи автономної системи

8.1. Цілі тестування

Основною метою випробувань було перевірити, як автономна машинка поводиться у реальних умовах — на різних типах поверхонь, при наявності перешкод, при ручному та автоматичному керуванні. Особливу увагу приділено точності виконання команд, стабільності сенсорних даних та надійності роботи електроніки.

8.2. Умови проведення тестів

Випробування проводились у кімнаті площею близько 15 м² із різними типами покриття (ламінат, плитка та килим), та на відкритому просторі в парку, також з різними типами покриття (штучна трава, асфальт, ґрунт). Додатково були встановлені штучні перешкоди (коробки, пластикові пляшки, стільці), що імітували реальні об'єкти на шляху.

Машинка працювала в автономному режимі та з керуванням через мобільний додаток по Wi-Fi. Живлення здійснювалось від двох акумуляторів 2S.

8.3. Результати автономного руху

Рух уперед / назад — працював стабільно. Машинка рухалась плавно, не ривками. Швидкість залежала від значення PWM-сигналу.

Реакція на перешкоди (ультразвукові сенсори + VL53L0X):

- При виявленні об’єкта на відстані менше 20 см машинка зупинялася;
- У деяких випадках (поглинаючі поверхні або тонкі предмети) сенсори могли не спрацювати, однак використання VL53L0X покращувало точність на ближній дистанції;
- У вузьких проходах машинка обирала доступний напрям для об’їзду (вліво або вправо).

Поворот керма (через драйвер L298N):

- Керування поворотом було стабільним, хоча кут залежав від тривалості сигналу;
- Після повороту машинка поверталася у початкове положення з невеликою похибкою (≈5–10°), що прийнятно для прототипу без датчика положення;
- Час реакції на команду повороту — миттєвий (затримка < 150 мс).

Режим стоп — зупинка виконувалася коректно з обнуленням PWM-сигналів.

Дані з MPU6050 дозволяли виявляти різкі нахили або удари (наприклад, зіткнення з перешкодою або перекидання), що можна використати для аварійної зупинки або статистики руху.

Живлення ESP32 і модулів залишалося стабільним протягом усього тестування. Перешкод у роботі від нестачі живлення не зафіксовано.

8.4. Взаємодія з мобільним додатком

- Підключення до ESP32 по Wi-Fi було стабільним у межах 8–10 метрів;
- Передача команд відбувалась без помітної затримки (≈100–200 мс);
- У випадку втрати зв’язку машинка зупинялася, що є бажаною поведінкою для автономної моделі;
- Перемикання між режимами ("ручний"/"автономний") працювало без збоїв.

8.5. Недоліки, виявлені під час тестів

- Похибка в керуванні поворотом через відсутність зворотного зв’язку від кермового механізму — можна усунути встановленням енкодера або потенціометра;
- Нестабільність ультразвукових сенсорів на чорних або м’яких поверхнях (типова особливість);

- GPS-модуль у приміщенні не працював стабільно — вирішено шляхом передачі координат з мобільного телефону;
- Ручне налаштування тривалості повороту потребує тонкої калібровки залежно від типу двигуна та навантаження.

8.6. Висновки до розділу 8

Випробування реальної автономної машинки показали, що система працює надійно та відповідає заявленим цілям. Основні функції реалізовані — самостійний рух, уникнення перешкод, керування зі смартфона, зупинка та перемикання режимів. Незважаючи на окремі обмеження апаратної бази, результати підтверджують, що навіть із простих і доступних компонентів можливо створити ефективний прототип автономного транспортного засобу для навчання чи досліджень.

9. Оцінка витрат та економічна доцільність

Розробка фізичного прототипу автономного транспортного засобу на базі RC-машинки та мікроконтролера ESP32 проводилася з урахуванням принципу економічності. Однією з головних цілей було створення повноцінної автономної системи з мінімальними фінансовими витратами, що робить проєкт доступним для навчальних закладів, студентських лабораторій та ентузіастів.

9.1. Основні витрати

У процесі розробки було використано такі компоненти:

Таблиця 9.1.

№	Компонент	К-сть	Орієнтовна ціна	Сума, грн	Примітка
1	RC-машинка	1	2500	2500	Базове шасі з двигунами
2	Wi-Fi модуль ESP32 LuaNode32 Type-C (38-pin)	1	239	239	Головний мікроконтроллер
3	Ультразвуковий датчик відстані US-025	4	44	176	Виявлення перешкод
4	GPS-модуль NEO-6M	1	217	217	Визначення координат
5	Лазерний сенсор відстані VL53L0X-V2	1	122	122	Виявлення перешкод більш точно
6	MPU-6050 (акселерометр і гіроскоп)	1	115	115	Стабілізація руху
7	Цифровий компас QMC5883L	1	88	88	Оріжнтування по курсу
8	Контролер двигуна BTS7960	1	229	229	Керування основним двигуном
9	Драйвер TB6612FNG	1	94	94	Керування поворотами
10	Перетворювач XL4015 (5A) з регулюванням струму/напруги	1	89	89	Пониження до 5В
11	Перетворювач XL4016E1 (8A)	1	133	133	Пониження до 7,4В
12	Конденсатори	6	5	30	Фільтрація живлення
13	Клемники, дроти, конектори	-	-	300	Орієнтовно
14	Припій, термоусадки	-	-	100	Орієнтовно

Загальна вартість всіх компонентів, склала близько 4500 грн.

Усі компоненти є доступними на ринку, що дозволяє швидко відтворити або масштабувати проєкт.

9.2. Трудові витрати

Загальний час, витрачений на реалізацію проєкту, склав близько 80–100 годин, які включають:

- моделювання симулятора (20–25 год),
- підбір та монтаж апаратних компонентів (20–30 год),
- програмування ESP32 та сенсорів (15–20 год),
- розробку мобільного додатку (10–15 год),
- тестування та налагодження (10–15 год).

9.3. Економічна доцільність

Проєкт доводить, що створення автономного транспортного засобу не потребує значних інвестицій у промислове обладнання. Модель, побудована на базі доступних електронних модулів, може слугувати ефективним інструментом для:

- навчання основ робототехніки та програмування;
- тестування алгоритмів автономної навігації;
- підготовки до більш масштабних промислових рішень.

Враховуючи низьку вартість, відкритість апаратної частини та гнучкість програмного забезпечення, система є економічно доцільною для освітнього, дослідницького та демонстраційного використання.

10. Висновки та перспективи подальшого розвитку

10.1. Основні підсумки роботи

У процесі виконання цієї роботи було розроблено повноцінний прототип системи автономного керування транспортним засобом. Система включає:

- Програмний симулятор у середовищі Unity, що дає змогу перевіряти алгоритми руху в безпечному віртуальному середовищі;
- Апаратну модель на базі ESP32 з підключеними сенсорами, двигунами та системою керування поворотом;
- Мобільний додаток, за допомогою якого можна або вручну керувати моделлю, або перемикатись у автономний режим вказуючи лише кінцеві координати.

Розроблена система була протестована у різних умовах реального середовища. Виявлено, що функціональність прототипу відповідає поставленим вимогам: машинка успішно виконує команди, уникає перешкод та стабільно взаємодіє з мобільним додатком.

10.2. Досягнуті результати

Поставлену мету — створення інтегрованої системи автономного керування з можливістю симуляції, — реалізовано на всі 100%. Серед основних досягнень можна виділити:

- Побудову керованого середовища в Unity, яке дає змогу візуально тестувати рух, обробку перешкод та сенсорну модель;
- Створення автономної машинки з базовими функціями самостійного руху (до вказаних координат) й ухилення від перешкод;
- Підключення мобільного додатку для гнучкого керування та передачі координат;
- Проведення аналізу ефективності системи у віртуальних та реальних умовах;

- Оптимізацію схеми живлення, управління й зчитування сенсорної інформації.

10.3. Практичне значення

Розроблений прототип є реальним прикладом, як навіть з обмеженим бюджетом можна створити робочу автономну систему. Цей проєкт може бути використаний у навчальних закладах для:

- Вивчення принципів автономного керування;
- Проведення лабораторних робіт;
- Підготовки до олімпіад і змагань з робототехніки;
- Розширення наукових досліджень у сфері інтелектуального транспорту.

10.4. Практичне значення

Незважаючи на досягнуті результати, проєкт має великий потенціал для подальшого вдосконалення. Серед можливих напрямів:

1. Вдосконалення алгоритму ухилення від перешкод

Хоча поточна система ефективно обробляє перешкоди завдяки ультразвуковим сенсорам, для покращення точності та стабільності поведінки рекомендується впровадити гібридну сенсорну модель. Комбінація даних з ультразвукових, інфрачервоних, оптичних та гіроскопічних сенсорів дозволить точніше ідентифікувати тип об'єкта, його розміри та відстань, а також мінімізувати хибні спрацьовування. Зокрема, доцільно реалізувати базову фільтрацію помилкових сигналів та використовувати часову кореляцію відгуків.

2. Реалізація адаптивної побудови маршруту

На поточному етапі рух реалізується за прямолінійною траєкторією з об’їздом перешкод без повторного коригування кінцевого вектора. У подальших дослідженнях варто реалізувати алгоритм адаптивного

відновлення маршруту, який дозволить машинці перебудовувати траєкторію в реальному часі на основі актуальних координат після кожного об’їзду. Це дасть змогу значно підвищити точність прибуття у кінцеву точку та зробить навігацію більш гнучкою.

3. Інтеграція алгоритмів машинного навчання

З метою розширення інтелектуальних можливостей системи рекомендується застосувати методи машинного навчання, зокрема нейронні мережі. Це дозволить моделі самонавчатися на основі історичних даних про маршрути, ухилення та помилки. Впровадження нейронних обчислень дозволить ефективніше розпізнавати контекст дорожньої ситуації, типи перешкод (рухомі або статичні) та передбачати оптимальні рішення для маневрування.

4. Оптимізація енергоспоживання та живлення

У подальших роботах варто дослідити варіанти зменшення енергоспоживання системи. Це включає оптимізацію режимів живлення ESP32, переведення сенсорів у режим очікування при неактивності, а також заміну драйверів або регуляторів напруги на більш енергоефективні аналоги. Для тривалого використання на відкритих територіях можна також інтегрувати автономне живлення на основі сонячних панелей.

5. Покращення мобільного додатку

На базі розробленого Android-додатку можна реалізувати додаткові функції: збереження історії маршрутів, дистанційне оновлення прошивки ESP32, налаштування поведінки (режим об’їзду, швидкість, чутливість сенсорів) та моніторинг у реальному часі з використанням карт Google Maps або OpenStreetMap. Також перспективним є впровадження голосового управління або інтеграції з хмарними сервісами.

6. Масштабування та модульність системи

Система має потенціал для масштабування на більш складні апаратні платформи, зокрема роботизовані візки або прототипи електромобілів. Для цього доцільно реалізувати модульну архітектуру, де сенсори, двигуни, блок живлення та керуюча логіка можуть легко замінюватися або вдосконалюватися. Також можна реалізувати підтримку розподіленої обробки, де окремі задачі виконуються на спеціалізованих мікроконтролерах або одноплатних комп’ютерах (наприклад, Raspberry Pi).

7. Поглиблене тестування в динамічних середовищах

Для підвищення реалістичності роботи системи слід організувати тестування в більш складних умовах — на відкритих майданчиках з реальними або випадковими перешкодами, змінним освітленням, вітром тощо. Також доцільно протестувати взаємодію з іншими рухомими об'єктами, наприклад, імітацію пішоходів або інших машин.

10.5. Загальний висновок

Запропонована система — це приклад ефективного поєднання програмного моделювання, апаратної реалізації та мобільного інтерфейсу. Проєкт доводить, автономний транспортний засіб — це не тільки складна інженерна задача, але й цілком реальна ціль для студентських і дослідницьких команд. Такий підхід розширює можливості вивчення новітніх технологій і створює базу для подальших інновацій.

11. Список використаної літератури

- Dosovitskiy A., Ros G., Codevilla F., Lopez A., Koltun V. CARLA: An Open Urban Driving Simulator // Proceedings of the 1st Annual Conference on Robot Learning (CoRL). 2017. P. 1–16. Посилання: https://arxiv.org/abs/1711.03938
- Microsoft AirSim. A simulator for drones, cars and more — built on Unreal Engine. Посилання: https://github.com/microsoft/AirSim
- LGSVL Autonomous Driving Simulator. 2020. Посилання: https://www.svlsimulator.com/
- Krajzewicz D. et al. SUMO – Simulation of Urban MObility: An overview // Proceedings of SIMUL 2012, The Fourth International Conference on Advances in System Simulation, 2012. P. 63–68. Посилання: https://sumo.dlr.de
- Unity Technologies. Unity Real-Time Development Platform. Посилання: https://unity.com
- Espressif Systems. ESP32 Technical Reference Manual. 2020. Посилання: https://www.espressif.com
- Arduino.cc. Official Arduino Documentation. Посилання: https://www.arduino.cc/en/Guide/HomePage
- VL53L0X Time-of-Flight Sensor. Datasheet. Посилання: https://www.st.com/resource/en/datasheet/vl53l0x.pdf
- HC-SR04 Ultrasonic Sensor Technical Data. Посилання: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- MPU6050 Gyroscope + Accelerometer Module Datasheet. Посилання: https://invensense.tdk.com
- TB6612FNG Dual Motor Driver Carrier. Pololu documentation. Посилання: https://www.pololu.com/product/713
- BTS7960 High Power Motor Driver Module. Datasheet. Посилання: https://www.electropeak.com

- NEO-6M GPS Module Datasheet. Посилання: https://www.u-blox.com
- QMC5883L Compass Sensor Datasheet. Посилання: https://wiki.dfrobot.com/QMC5883L
- OpenCV: Open Source Computer Vision Library. Посилання: https://opencv.org
- Android Developers. Official Android Documentation. Посилання: https://developer.android.com
- Документація по ESP32 Arduino Core Посилання: https://docs.espressif.com/projects/arduino-esp32/en/latest/

12. Додатки та коди C#:

12.1. Приклад прошивки ESP32 (Arduino IDE)

```
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_VL53L0X.h>

// Wi-Fi
const char* ssid = "ESP32-Car";
const char* password = "12345678";
WiFiServer server(80);

// Двигун (BTS7960)
#define RPWM 25
#define LPWM 26

// Поворот (L298N)
#define ENA 27
#define IN1 32
#define IN2 33

// Ультразвукові сенсори
#define TRIG_FL 4
#define ECHO_FL 5
```

```
#define TRIG_FR 14
#define ECHO_FR 12
#define TRIG_BL 18
#define ECHO_BL 19
#define TRIG_BR 21
#define ECHO_BR 22

// Лазерний сенсор (опційно)
Adafruit_VL53L0X vl53 = Adafruit_VL53L0X();

// PWM
const int PWM_FREQ = 1000;
const int PWM_RES = 8;

// Автономний режим увімкнено?
bool autonomousMode = false;

// Час об'їзду
unsigned long lastAvoidTime = 0;
const unsigned long avoidDelay = 3000;

void setupPins() {
```

```
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(TRIG_FL, OUTPUT); pinMode(ECHO_FL, INPUT);
  pinMode(TRIG_FR, OUTPUT); pinMode(ECHO_FR, INPUT);
  pinMode(TRIG_BL, OUTPUT); pinMode(ECHO_BL, INPUT);
  pinMode(TRIG_BR, OUTPUT); pinMode(ECHO_BR, INPUT);
}

void setupPWM() {
  ledcSetup(0, PWM_FREQ, PWM_RES); ledcAttachPin(RPWM, 0);
  ledcSetup(1, PWM_FREQ, PWM_RES); ledcAttachPin(LPWM, 1);
  ledcSetup(2, PWM_FREQ, PWM_RES); ledcAttachPin(ENA, 2);
}

void setup() {
  Serial.begin(115200);
  setupPWM();
  setupPins();
  Wire.begin();

  if (!vl53.begin()) {
    Serial.println("VL53L0X не знайдено. Продовжую без нього.");
```

```
  }

  WiFi.softAP(ssid, password);
  server.begin();
  Serial.println("Wi-Fi точка активна: ESP32-Car");
}

long readDistance(int trig, int echo) {
  digitalWrite(trig, LOW); delayMicroseconds(2);
  digitalWrite(trig, HIGH); delayMicroseconds(10);
  digitalWrite(trig, LOW);
  long d = pulseIn(echo, HIGH, 30000);
  return (d > 0) ? d * 0.034 / 2 : -1;
}

// ===== Керування мотором =====
void stopMotors() {
  ledcWrite(0, 0); ledcWrite(1, 0);
}
void forward() {
  ledcWrite(0, 200); ledcWrite(1, 0);
}
```

```
void backward() {
  ledcWrite(0, 0); ledcWrite(1, 200);
}

// ===== Повороти =====
void turnLeft(int duration = 400) {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  ledcWrite(2, 150); delay(duration); ledcWrite(2, 0);
}
void turnRight(int duration = 400) {
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  ledcWrite(2, 150); delay(duration); ledcWrite(2, 0);
}

// ===== Автономна логіка =====
void handleAutonomous() {
  long dFL = readDistance(TRIG_FL, ECHO_FL);
  long dFR = readDistance(TRIG_FR, ECHO_FR);

  Serial.print("Front L: "); Serial.print(dFL); Serial.print(" R: "); Serial.println(dFR);

  if ((dFL > 0 && dFL < 25) || (dFR > 0 && dFR < 25)) {
```

```
    stopMotors();
    delay(300);

    // Невеликий задній хід
    backward(); delay(500); stopMotors(); delay(200);

    // Перевірка боків
    if (dFL > dFR) {
      turnRight();
    } else {
      turnLeft();
    }
    lastAvoidTime = millis();
  } else {
    forward();
  }
}

void loop() {
  WiFiClient client = server.available();
  if (client) {
    String req = client.readStringUntil('\r'); req.trim(); client.flush();
```

```
Serial.println("Команда: " + req);


if (req == "F") { forward(); autonomousMode = false; }
else if (req == "B") { backward(); autonomousMode = false; }
else if (req == "L") { turnLeft(); autonomousMode = false; }
else if (req == "R") { turnRight(); autonomousMode = false; }
else if (req == "S") { stopMotors(); autonomousMode = false; }
else if (req == "AUTO") {
  autonomousMode = true;
  Serial.println("Автономний режим УВІМКНЕНО");
}
else if (req == "MANUAL") {
  autonomousMode = false;
  stopMotors();
  Serial.println("Режим ручний");
}


// Виведення дистанцій
long dFL = readDistance(TRIG_FL, ECHO_FL);
long dFR = readDistance(TRIG_FR, ECHO_FR);
long dBL = readDistance(TRIG_BL, ECHO_BL);
long dBR = readDistance(TRIG_BR, ECHO_BR);
```

```
    VL53L0X_RangingMeasurementData_t measure;
    vl53.rangingTest(&measure, false);

    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/plain");
    client.println();
    client.println("US-FL: " + String(dFL) + " cm");
    client.println("US-FR: " + String(dFR) + " cm");
    client.println("US-BL: " + String(dBL) + " cm");
    client.println("US-BR: " + String(dBR) + " cm");
    client.println("VL53L0X: " + String(measure.RangeMilliMeter) + " mm");
  }

  // Автономний режим (цикл)
  if (autonomousMode) {
    handleAutonomous();
  }
}}
}
```

12.2. Приклад файлу MapActivity.kt, для Android

```
package com.example.cartracker
```

```
import android.Manifest
import android.content.pm.PackageManager
import android.location.Location
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.annotation.RequiresPermission
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationServices
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.MarkerOptions
import com.google.android.gms.maps.model.PolylineOptions
import java.net.HttpURLConnection
import java.net.URL

// Основна активність, яка відповідає за відображення карти і взаємодію з ESP32
class MapActivity : AppCompatActivity(), OnMapReadyCallback {

    // Змінні для карти, кнопок і координат
    private lateinit var mMap: GoogleMap
    private lateinit var coordinateInput: EditText
```

```
    private lateinit var sendToEspButton: Button
    private lateinit var getEspLocationButton: Button
    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private lateinit var setStartPointButton: Button

    // Стартова і кінцева точки маршруту
    private var startPoint: LatLng? = null
    private var destinationLatLng: LatLng? = null
    private var selectingStartPoint = false

    // Код запиту на дозвіл локації
    companion object {
        private const val LOCATION_PERMISSION_REQUEST_CODE = 1001
    }

    // Ініціалізація активності
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_map)

        // Прив'язуємо елементи інтерфейсу до змінних
        coordinateInput = findViewById(R.id.coordinateInput)
        sendToEspButton = findViewById(R.id.sendToEspButton)
        setStartPointButton = findViewById(R.id.setStartPointButton)
        getEspLocationButton = findViewById(R.id.getEspLocationButton)

        // Ініціалізуємо клієнт для отримання локації
        fusedLocationClient                                    =
LocationServices.getFusedLocationProviderClient(this)
```

```
        // Підготовка карти
        val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as
SupportMapFragment
        mapFragment.getMapAsync(this)


        // Обробник кнопки введення координат вручну
        findViewById<Button>(R.id.setCoordinatesButton).setOnClickListener {
            val coordinates = coordinateInput.text.toString()
            if (coordinates.isNotEmpty()) {
                val coords = coordinates.split(",")
                if (coords.size == 2) {
                    try {
                        val lat = coords[0].toDouble()
                        val lng = coords[1].toDouble()
                        destinationLatLng = LatLng(lat, lng)
                        drawRoute()
                    } catch (e: Exception) {
                        Toast.makeText(this, "Некоректні координати",
Toast.LENGTH_SHORT).show()
                    }
                } else {
                    Toast.makeText(this, "Введіть коректні координати",
Toast.LENGTH_SHORT).show()
                }
            } else {
                Toast.makeText(this, "Будь ласка, введіть координати",
Toast.LENGTH_SHORT).show()
            }
        }


        // Надсилання координат на ESP32
```

```
        sendToEspButton.setOnClickListener {
            destinationLatLng?.let {
                sendCoordinatesToEsp32(it)
            } ?: Toast.makeText(this, "Кінцева точка не встановлена", Toast.LENGTH_SHORT).show()
        }

        // Симуляція отримання координат з ESP32
        getEspLocationButton.setOnClickListener {
            Toast.makeText(this, "Запит координат з ESP32...", Toast.LENGTH_SHORT).show()

            // TODO: Зробити HTTP-запит до ESP32 для отримання координат
            // Поки що використовуємо фіксовані координати
            val lat = 48.3794
            val lng = 31.1656
            startPoint = LatLng(lat, lng)
            drawRoute()
        }

        // Вибір стартової точки натисканням на карту
        setStartPointButton.setOnClickListener {
            Toast.makeText(this, "Тепер натисніть на карту для вибору стартової точки", Toast.LENGTH_SHORT).show()
            selectingStartPoint = true
        }
    }

    // Коли карта готова до використання
    override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap
```

```
        // Перевірка дозволів на доступ до локації
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                LOCATION_PERMISSION_REQUEST_CODE
            )
        } else {
            enableUserLocation()
        }

        // Обробка натискання на карту: або задаємо стартову точку, або фінішну
        mMap.setOnMapClickListener { latLng ->
            if (selectingStartPoint) {
                startPoint = latLng
                selectingStartPoint = false
                Toast.makeText(this, "Стартова точка встановлена",
Toast.LENGTH_SHORT).show()
                drawRoute()
            } else {
                destinationLatLng = latLng
                drawRoute()
            }
        }
    }

    // Отримання поточної локації користувача і встановлення її як стартової
точки
```

```
    @RequiresPermission(allOf = [Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION])
    private fun enableUserLocation() {
        fusedLocationClient.lastLocation.addOnSuccessListener { location: Location?
->
            location?.let {
                startPoint = LatLng(it.latitude, it.longitude)
                mMap.clear()
                mMap.addMarker(MarkerOptions().position(startPoint!!).title("Моя
позиція"))

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(startPoint!!, 15f))
            } ?: Toast.makeText(this, "Не вдалося отримати місцезнаходження",
Toast.LENGTH_SHORT).show()
        }
    }


    // Малюємо маршрут між стартом і фінішем
    private fun drawRoute() {
        val start = startPoint
        val dest = destinationLatLng
        if (start != null && dest != null) {
            mMap.clear()
            mMap.addMarker(MarkerOptions().position(start).title("Старт"))
            mMap.addMarker(MarkerOptions().position(dest).title("Фініш"))

            val pathPoints = calculateStraightPath(start, dest)

            val polylineOptions = PolylineOptions()
                .addAll(pathPoints)
                .width(5f)
```

```
            .color(0xFF0000FF.toInt())

        mMap.addPolyline(polylineOptions)
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(dest, 15f))
    }
}

// Розрахунок прямої лінії між двома точками маршруту (10 кроків)
private fun calculateStraightPath(start: LatLng, end: LatLng): List<LatLng> {
    val path = mutableListOf<LatLng>()
    val steps = 10
    for (i in 0..steps) {
        val lat = start.latitude + (end.latitude - start.latitude) * i / steps
        val lng = start.longitude + (end.longitude - start.longitude) * i / steps
        path.add(LatLng(lat, lng))
    }
    return path
}

// Відправлення координат на ESP32 через HTTP GET-запит
private fun sendCoordinatesToEsp32(latLng: LatLng) {
    val ip = "192.168.4.1"  // IP ESP32
    val url =
"http://$ip/set_coords?lat=${latLng.latitude}&lng=${latLng.longitude}"

    Thread {
        try {
            val connection = URL(url).openConnection() as HttpURLConnection
            connection.requestMethod = "GET"
            connection.connectTimeout = 3000
```

```
            connection.readTimeout = 3000

            val responseCode = connection.responseCode
            runOnUiThread {
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    Toast.makeText(this, "Координати надіслано", Toast.LENGTH_SHORT).show()
                } else {
                    Toast.makeText(this, "Помилка: $responseCode", Toast.LENGTH_SHORT).show()
                }
            }
            connection.disconnect()
        } catch (e: Exception) {
            e.printStackTrace()
            runOnUiThread {
                Toast.makeText(this, "Не вдалося надіслати: ${e.message}", Toast.LENGTH_SHORT).show()
            }
        }
    }.start()
}

// Обробка результату запиту на дозвіл доступу до локації
@RequiresPermission(allOf = [Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_COARSE_LOCATION])
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
```

```
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {
            if ((grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)) {
                enableUserLocation()
            } else {
                Toast.makeText(this, "Доступ до локації заборонено",
Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```