Невлюдов І.Ш., Євсєєв В.В., Гурін Д.В.

Технічне та програмне забезпечення розробки малогоборитного мобільного робота

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Кафедра комп’ютерно-інтегрованих технологій, автоматизації та робототехніки (КІТАР)

.

І.Ш. Невлюдов, В.В. Євсєєв, Д.В. Гурін

ТЕХНІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РОЗРОБКИ МАЛОГАБАРИТНОГО МОБІЛЬНОГО РОБОТА

[Монографія]

Харків
2025

УДК 681.51:[004.415.2:007.52]
Н40

Рекомендовано Науково-технічною радою Харківського національного університету радіоелектроніки (протокол № 4 від 22.05.2025 року)

Рецензенти:
доктор технічних наук, професор, завідувач кафедри «Біомедичної інженерії», Харківського національного університету радіоелектроніки
Аврунін Олег Григорович
кандидат технічних наук, доцент, директор Державного підприємства «Південний державний проектно-конструкторський інститут авіаційної промисловості»
Артюх Роман Володимирович
доктор технічних наук, професор, професор кафедри «Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки», Харківського національного університету радіоелектроніки
Цимбал Олександр Михайлович

Н40 **Технічне та програмне забезпечення розробки малогабаритного мобільного робота:** монографія / І.Ш. Невлюдов, В.В. Євсєєв, Д.В. Гурін. Кривий Ріг: Криворізький фаховий коледж Державного некомерційного підприємства «Державний університет «Київський авіаційний інститут», 2025. – 355 с.

Монографія "Технічне та програмне забезпечення розробки малогабаритного мобільного робота" є комплексним дослідженням, що вивчає різні напрямки розробки технічного та програмного забезпечення для малогабаритних мобільних роботів. Монографія охоплює такі теми, як класифікація мобільних роботів, проектування концепції та структурної схеми робота, вибір апаратних модулів, розрахунок витрат енергії для керування рухом робота, розробка системи керування з вибором мови програмування, розробка алгоритмів керування та програмної реалізації, а також методи і алгоритми обробки зображень та побудови маршрутів руху. Монографія містить важливі теоретичні роздуми та практичні висновки, що робить її цінним джерелом знань для вчених, інженерів та здобувачів, що цікавляться робототехнікою.

У першому розділі розглядається розробка самого робота, включаючи класифікацію мобільних роботів, проектування концепції, структурну схему, вибір апаратних модулів, розрахунок витрат енергії та розробку 3D моделей.

Другий розділ присвячений розробці системи керування роботом, включаючи вибір мови програмування, розробку алгоритмів керування, програмну реалізацію, налаштування та прошивку плати керування та створення HMI інтерфейсу.

Третій розділ концентрується на методах та алгоритмах обробки зображень для виявлення та відстеження об'єктів у реальному часі. Він описує методи аналізу зображень, алгоритми Canny, активних контурів, оптичного потоку, алгоритм Грейгема, розпізнавання та відстеження об'єктів, алгоритм Sobel та метод Лукаса-Канаде.

У четвертому розділі розглядається побудова маршрутів руху робота, включаючи алгоритми находження короткого маршруту A*, BRRT та A*, побудову оптимального маршруту з урахуванням перешкод, побудову маршруту на базі волнового алгоритму, побудову 3D маршруту на базі алгоритму PRM та алгоритм D*.

Монографія може бути корисна науковцям, аспірантам, здобувачам, які займаються питаннями розробки інтелектуальних робототехнічних систем та фахівцям в галузі знань 17 - Електроніка, автоматизація та електронні комунікації за спеціальністю 174 - Автоматизація, комп'ютерно-інтегровані технології та робототехніка та освітньо-професійних програм: «Робототехніка та кіберфізичні системи», «Автоматизація та комп'ютерно-інтегровані технології».

ISBN:978-617-8332-74-7.
DOI: 10.30837/978-617-8332-74-7.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

МР - мобільний робот

СКМР - система керування мобільним роботом

АР - апаратний модуль

СІ - схема підключення

ДВЕ - додаткові витрати енергії

МКМР - макет мобільного робота

ВР - вибір среди розробки

СКМР - система керування мобільним роботом

АКМР - алгоритм керування мобільним роботом

ПРСК - програмна реалізація системи керування

НППК - налаштування та прошивка плати керування

HMI - human-machine interface (людино-машинний інтерфейс)

ОО - об'єктовий орієнтований

РДР - рабоча зона мобільного робота

РР - реалізація методів і алгоритмів построєння маршрутів

ЛК - Лукаса-Канаде

АК - активний контур

ОП - оптичний потік

АС - алгоритм Canny

АС - алгоритм Sobel

ОП - оптимальний маршрут

ВА - волновий алгоритм

PRM - probabilistic roadmap method (імовірнісний метод карт маршрутів)

BRRT - bidirectional rapidly exploring random tree (двонапрямлений швидко досліджуючий випадковий дерево)

D* - D-star (D-зірка)

ММ - малогабаритний мобільний

ТПЗ - технічне та програмне забезпечення

ВСТУП

Актуальність досліджень у галузі технічного та програмного забезпечення малогабаритних мобільних роботів є надзвичайно високою у сучасному технологічному світі. Це обумовлено зростаючим попитом на такі роботи у різних сферах, включаючи промисловість, медицину, побут та оборону. Розробка ефективних технічних та програмних рішень для малогабаритних мобільних роботів може значно поліпшити їх функціональність та відкрити нові можливості для використання.

Однією з ключових проблем у цій галузі є розробка ефективних алгоритмів керування та навігації для малогабаритних мобільних роботів. Ці алгоритми повинні бути надійними, швидкими та енергоефективними для забезпечення оптимальної роботи робота у різних умовах.

Дослідження у цій області також мають велике значення для розвитку автономних систем. Розробка програмного забезпечення, яке дозволить малогабаритним мобільним роботам працювати без прямого втручання людини, може значно підвищити їх ефективність та універсальність в різних сферах застосування. Крім того, важливою аспектом є розробка технічного забезпечення для малогабаритних мобільних роботів. Це включає в себе розробку механічних частин, сенсорів, актуаторів та інших компонентів, які дозволять роботам ефективно взаємодіяти з навколишнім середовищем.

Одним із напрямків досліджень є розробка програмних інтерфейсів для взаємодії з мобільними роботами. Це може включати в себе створення інтерфейсів для керування роботами, візуалізації даних, збору і аналізу даних та інших функцій, що підвищать зручність користування роботами.

Розробка малогабаритних мобільних роботів є актуальною також і з соціального погляду, оскільки вона може призвести до автоматизації багатьох видів робіт та покращення умов праці.

Таким чином, дослідження у галузі технічного та програмного забезпечення малогабаритних мобільних роботів мають великий потенціал для подальшого розвитку технологій автономних систем та покращення умов життя людей.

Монографія присвячена дослідженню технічного та програмного забезпечення малогабаритного мобільного робота. У роботі детально розглядаються основні аспекти розробки таких роботів, включаючи класифікацію мобільних роботів, розробку концепції та структурної схеми мобільного робота. Окремий акцент робиться на виборі апаратних модулів та розрахунку витрат енергії для керування рухом мобільного робота.

У другому розділі монографії розглядається розробка системи керування малогабаритним мобільним роботом, включаючи вибір середовища розробки, розробку алгоритму керування, програмну реалізацію системи керування, налаштування та прошивку плати керування та розробку HMI інтерфейсу системи керування мобільним роботом.

Третій розділ монографії присвячено реалізації методів і алгоритмів оконтурювання об'єктів в зоні робочого простору мобільного робота в реальному часі. Автор досліджує такі методи, як аналіз зображень, алгоритм Canny, метод активних контурів, метод оптичного потоку та алгоритм Грейгема, метод розпізнавання та відстеження об'єкта, алгоритм Sobel та метод Лукаса-Канаде.

У завершальному четвертому розділі монографії розглядається реалізація методів і алгоритмів побудови маршрутів руху мобільного робота, включаючи алгоритм находження короткого маршруту A*, алгоритм BRRT та A*, побудову оптимального маршруту з урахуванням перешкод на карті, побудову маршруту на базі хвильового алгоритму в динамічному просторі, побудову 3D маршруту на базі алгоритму PRM та алгоритм D*.

Монографія містить важливі теоретичні та практичні результати у галузі розробки технічного та програмного забезпечення малогабаритних

мобільних роботів, що може бути корисним для науковців, інженерів та здобувачів, що цікавляться даною тематикою.

1. РОЗРОБКА МАЛГОБОРИТНОГО МОБІЛЬНОГО РОБОТА ДЛЯ ДОСЛІДЖЕННЯ ЗРУЙНОВАНИХ БУДІВЕЛЬ

1.1 Класифікація мобільних роботів

Дослідження класифікації мобільних роботів перед розробкою малогабаритного мобільного робота для дослідження зруйнованих будівель є критично важливим кроком для забезпечення ефективного та успішного розвитку проекту. Перш за все, таке дослідження дозволить зрозуміти різноманітні типи мобільних роботів, їх особливості та можливості, що є ключовим для вибору найбільш відповідного типу для конкретної задачі дослідження зруйнованих будівель. Класифікація роботів також допоможе з'ясувати, які саме характеристики та функції потрібні для ефективного дослідження зруйнованих будівель та які роботи вже існують на ринку. Дослідження допоможе визначити потрібні технології та компоненти для розробки мобільного робота, такі як сенсори, актюатори, навігаційні системи тощо. Також важливо врахувати рівень автономності та комунікаційні можливості робота для забезпечення ефективного виконання завдань у складних умовах зруйнованих будівель. Класифікація роботів може визначити потреби в програмному забезпеченні та алгоритмах, необхідних для роботи в умовах обмеженого доступу та обмеженого простору [1]. Дослідження також дозволить виявити потреби в безпеці та надійності роботів для роботи в небезпечних умовах, що є важливим для захисту операторів та навколишнього середовища. Крім того, класифікація мобільних роботів дозволить виявити потенційні перешкоди та виклики, що можуть виникнути при розробці малогабаритного мобільного робота для дослідження зруйнованих будівель.

Загальну класифікацію мобільних роботів можна представити у наступному вигляді [2]:

За типом ходу:

- колісні роботи;
- гусеничні роботи;
- ножні (лапові) роботи.

Середовища (аеродромні, плаваючі, підводні, космічні)

За місцем роботи:

- наземні роботи;
- аеродромні роботи;
- підводні роботи;
- космічні роботи.

За цілями використання:

- дослідницькі роботи;
- військові роботи;
- промислові роботи;
- побутові роботи.

За принципом керування:

- дистанційне керування;
- автономне керування;
- напівавтономне керування.

За конструкцією:

- статичні (непересувні);
- динамічні (змінної форми, деформовані роботи).

За основним завданням:

- пошуково-рятувальні;
- транспортні;
- медичні;
- військові.

За принципом енергопостачання:

- електричні;
- гібридні;

- внутрішнього згоряння;

- сонячні.

За характером використання:

- промислові;

- побутові;

- науково-дослідницькій.

Це лише загальна класифікація, в кожній групі можуть бути ще деякі підгрупи, залежно від конкретних властивостей та характеристик роботів.

Малогабаритний мобільний робот для дослідження зруйнованих будівель може бути класифікований за такими критеріями:

- розмір і маса. Робот має невеликі розміри і низьку масу для легкості переміщення в ускладнених умовах;

- маневреність. Робот повинен мати високу маневреність для руху в зруйнованих просторах, можливість об'їзду перешкод та проникнення в узкі прогалини;

- стабільність. Важливо, щоб робот був стійким на нерівних поверхнях та під час перетину завалів, щоб уникнути падіння та ушкоджень;

- датчики. Робот повинен мати набір датчиків (наприклад, камери, сенсори відстані, акселерометри), які дозволяють отримувати інформацію про навколишнє середовище;

- навігація. Робот може мати систему навігації, яка дозволяє йому прокладати оптимальний шлях в зруйнованому середовищі;

- інфраструктура комунікації. Робот може бути обладнаний засобами зв'язку для передачі даних та команд оператору або іншим системам;

захист від шкідливих впливів. У зруйнованих будівлях можуть бути шкідливі речовини або гази, тому робот може мати систему фільтрації повітря або інші засоби захисту;

- можливість носити вантаж. Робот може бути обладнаний спеціальною конструкцією для перенесення невеликих вантажів або датчиків;

- спроможність до автономності. Робот може мати можливість працювати в автономному режимі без постійного контролю оператора;

- взаємодія з іншими роботами: У складних умовах досліджень можуть бути задіяні кілька роботів, які повинні вміти координуватися та співпрацювати;

- віддалений доступ до даних. Оператор може мати можливість отримувати дані з робота в реальному часі через спеціальні системи зв'язку;

- аналітичні можливості. Робот може мати можливість обробляти отримані дані та надавати аналітичну інформацію про стан зруйнованого об'єкта;

- система енергопостачання. Робот може мати систему енергопостачання, яка дозволяє йому працювати в умовах, коли доступ до джерел енергії обмежений.

Малогабаритний мобільний робот для дослідження зруйнованих будівель є важливим інструментом у сучасних технологіях рятувальних операцій [3,4]. Цей робот має ряд характеристик, які роблять його ефективним і універсальним у таких умовах. Він має невеликі розміри і низьку масу, що дозволяє легко переміщатися в ускладнених умовах зруйнованих будівель. Висока маневреність робота дозволяє йому обходити перешкоди та проникати у вузькі прогалини, щоб досліджувати недоступні місця. Стійкість на нерівних поверхнях та під час перетину завалів гарантує безпеку робота та уникнення можливих ушкоджень [5].

Основні функції робота забезпечуються за допомогою датчиків, які дозволяють отримувати інформацію про навколишнє середовище [6]. Система навігації дозволяє роботу прокладати оптимальний шлях у зруйнованому середовищі, а інфраструктура комунікації забезпечує зв'язок з оператором та іншими системами. Робот може бути обладнаний захисними системами від шкідливих впливів, такими як фільтрація повітря, що дозволяє йому працювати в небезпечних умовах [7]. Можливість носити вантаж та спроможність до автономності роблять робота більш універсальним у

використанні. Взаємодія з іншими роботами та аналітичні можливості дозволяють роботу ефективно виконувати завдання у команді та надавати аналіз інформації [8]. Система енергопостачання забезпечує роботу в умовах обмеженого доступу до джерел енергії.

1.2 Розробка концепту мобільного робота

У сучасних умовах руйнівних техногенних кадастрів і військових подій, які можуть охоплювати різні регіони світу, питання безпеки та ефективності відновлення стають дедалі актуальнішими. Одним із важливих акцентів під час дослідження пошкоджених будівель побудованих за принципом залізобетонних панельних конструкцій, є його крихкість, непередбачуваність поведінки, а також мінімальні робочі простори всередині або між плитами. Що ускладнює пошуково-рятувальні роботи та проведення конструкційного аналізу міцності пошкодженої будівлі [9]. Приклад таких зруйнованих будівель у м. Харкові, Україна внаслідок військової агресії Російської Федерації наведено на рисунку 1.1.

Рисунок 1.1 – Приклад зруйнованих панельних будинків м. Харкова внаслідок воєнних дій

З огляду на особливості конструкційного руйнування залізобетонних панельних конструкцій під час влучання снаряда чи ракети їх можливо класифікувати наступним чином [10]:

- пробивання та проникнення. Снаряд або ракета можуть спричинити пробивання через поверхневі шари залізобетону. Це може створити отвори, спричинити деформацію панелі та призвести до обвалення конструкцій у вигляді складання панелей або часткової руйнації;

- утворення тріщин і осипання. Попадання снаряда може спричинити утворення тріщин у структурі бетону. Це може призвести до осипання поверхневих шарів і, можливо, до подальшого руйнування структури;

- стиснення і деформація. Ударні впливи можуть викликати стиснення і деформацію залізобетонних панелей. Це може призвести до втрати міцності та зміни форми конструкції, часткового обвалення елементів конструкцій будівлі;

- загальне руйнування й обвалення. Після влучання снаряда може статися загальне руйнування і навіть обвалення частин або всієї будівлі, особливо якщо структурні елементи істотно пошкоджені.

Виходячи з цього можна бачити, що застосування класичних мобільних роботів для проведення досліджень пошкоджених будівель, в умовах мінімально обмеженого робочого простору, що виникає під час обстеження панельних конструкцій будівлі, є не доцільним, через великі габарити, що збільшує вірогідність пошкодження мобільного робота з можливістю повної втрати, а також великою вартістю самого робота, що може сягати ~5.000-15.000$ і більше [11]. Приклади таких мобільних роботів наведено на рисунку 1.2

а) б) в)

а) NiFTi [12]; б) Semi – Autonomous Rescure Robot [13]; с) Rescue Robots [14]

Рисунок 1.2 – Класичні мобільні роботи для проведення досліджень пошкоджених будівель

Отже, в цьому контексті розробка малогабаритних роботів з використанням адитивних технологій 3D друку, здатних оперативно та ефективно взаємодіяти всередині зруйнованих залізобетонних панельних конструкцій, в умовах обмеженого робочого простору, є важливим кроком у забезпеченні безпеки та швидкого дослідження місць руйнування для прийняття відповідних рішень щодо стану будівлі.

Роботи в зоні техногенних катастроф, особливо в областях з обмеженим робочим простором, повинні бути розроблені з урахуванням специфіки їх експлуатації [15]. У рамках цих досліджень введемо такі вимоги до розроблюваної конструкції мобільного робота:

- компактні розміри і маневреність. Роботи повинні бути компактними, щоб легко проникати в обмежені і важкодоступні місця між залізобетонними конструкціями тощо;

- інтелектуальна система навігації. Розробка розумних алгоритмів навігації, що дають змогу роботу уникати перешкод і правильно будувати маршрут в складних умовах;

- камери і датчики. Використання камер, лазерних сенсорів та інших датчиків для навігації та пошуку проблемних ділянок у конструкціях;

- стійкість і керування за відстанню. Роботи повинні бути стійкими і здатними витримувати нерівності поверхні, а також бути здатними до керування за відстанню, щоб уникнути ризиків для оператора;

- захист від впливу зовнішніх факторів. Конструкція має бути захищена від пилу, щоб забезпечити надійну роботу в умовах забрудненого середовища.

- безпека. Використання безпечних матеріалів і додаткових заходів безпеки, щоб уникнути аварійних ситуацій, особливо в умовах обвалень або небезпечних зон.

- дальність і тривалість роботи. Забезпечення достатньої дальності та тривалості роботи за один цикл для ефективного виконання завдань в умовах обмеженого доступу.

Виходячи з вимог, які вказані вище, автори розробили концепт конструкцій малогабаритного мобільного робота, ескіз якого представлений на рисунку 1.3.

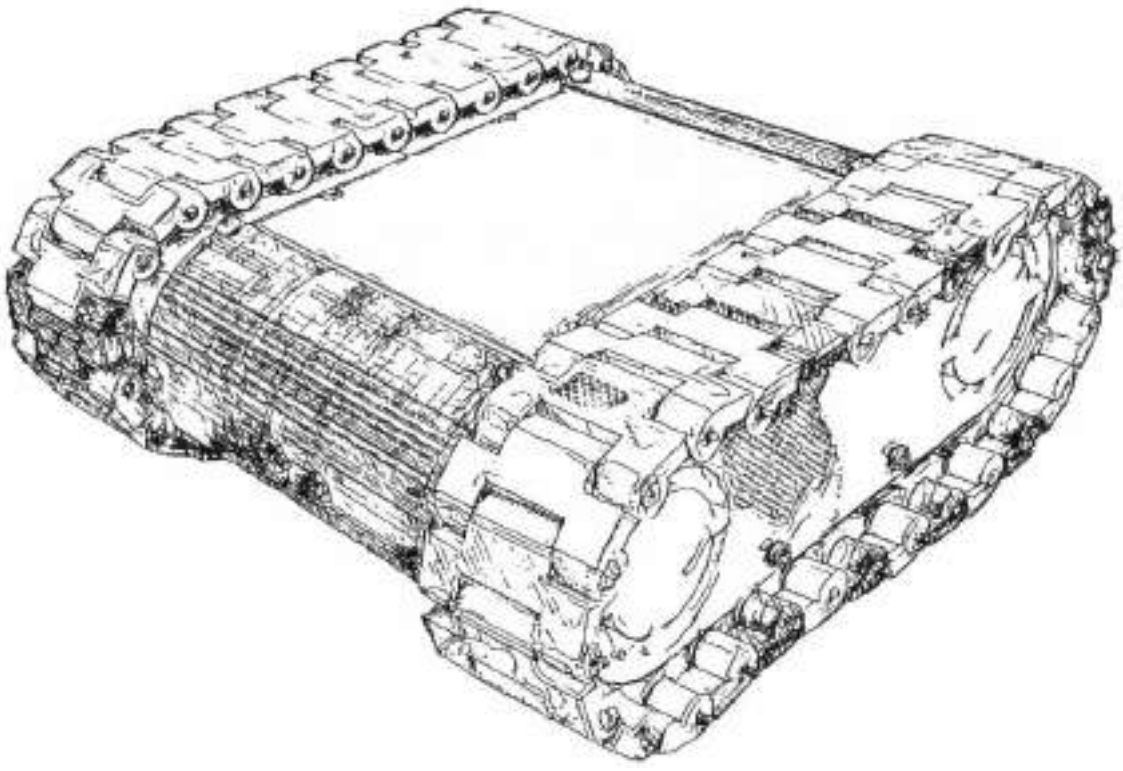


Рисунок 1.3 – Ескіз концепту малогабаритного робота

Розробка ескізу конструкції малогабаритного мобільного робота після створення ескізу концепту має важливе значення для детальнішого проєктування та визначення ключових аспектів майбутнього пристрою [16]. Цей етап дає змогу уточнити технічні рішення, визначити розміри та масу робота, аналізувати технічну здійсненність проєкту, а також оцінити його вартість і терміни реалізації. Розробка ескізу конструкції є невід'ємною частиною підготовки до детальнішого проєктування і створення прототипу робота, що робить цей етап вкрай важливим для успішної реалізації проєкту. Ескіз конструкцій малогабаритного мобільного робота з представленням основних елементів представлений на рисунку 1.4

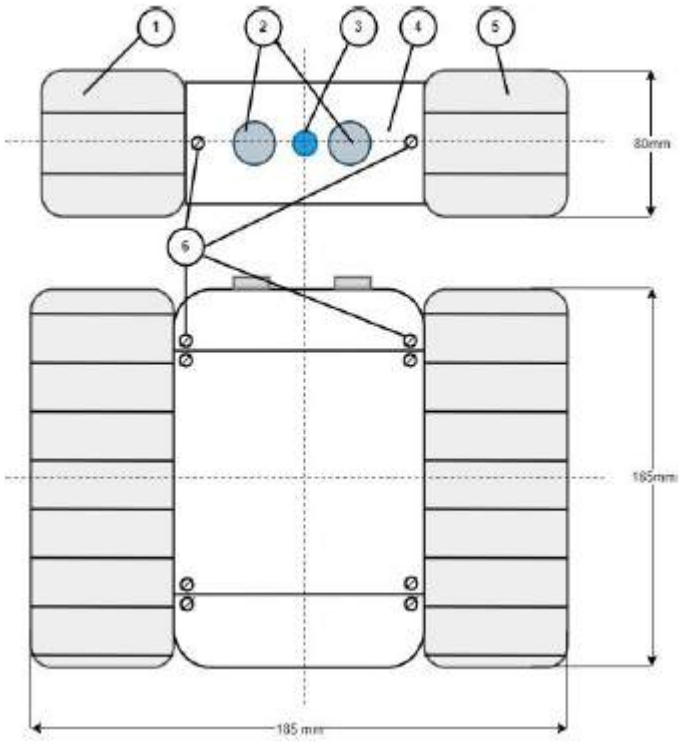


Рисунок 1.4 – Ескіз конструкцій малогабаритного мобільного робота з позначенням габаритних розмірів

Опишемо позначення основних елементів, що вказані на ескізі конструкцій малогабаритного мобільного робота (рис.1.4). 1,5 - гусениці; 2 - датчики відстані; 3 - камера для візуалізацій довкілля оператору; 4 - корпус

мобільного робота, отриманий за допомогою 3D-друку; 6 - металовироби для кріплення елементів корпусу.

1.3 Розроблення структурної схеми мобільного робота

Розробка структурної схеми для мобільного робота є важливим етапом у його створенні та експлуатації. Цей процес дає змогу чітко визначити функціональність робота, організувати її в логічну послідовність і визначити необхідні компоненти та модулі [17,18]. Структурна схема допомагає керувати складністю робота, розбиваючи її на менші підсистеми, і визначити взаємозв'язки між компонентами. Вона також сприяє підвищенню надійності робота, оскільки дає змогу виявляти можливі проблеми на етапі проєктування. Крім того, структурна схема уможливлює легке розширення та модифікацію функціональності робота в майбутньому і спрощує його підтримку та ремонт. Таким чином, розробка структурної схеми є важливим кроком у забезпеченні ефективної та надійної роботи мобільного робота. Виходячи з проведеного аналізу та розробленого концепту мобільного робота (рис. 1.3, 1.4), пропонується наступна структурна схема керування мобільного робота, яку представлено на рисунку 1.5.

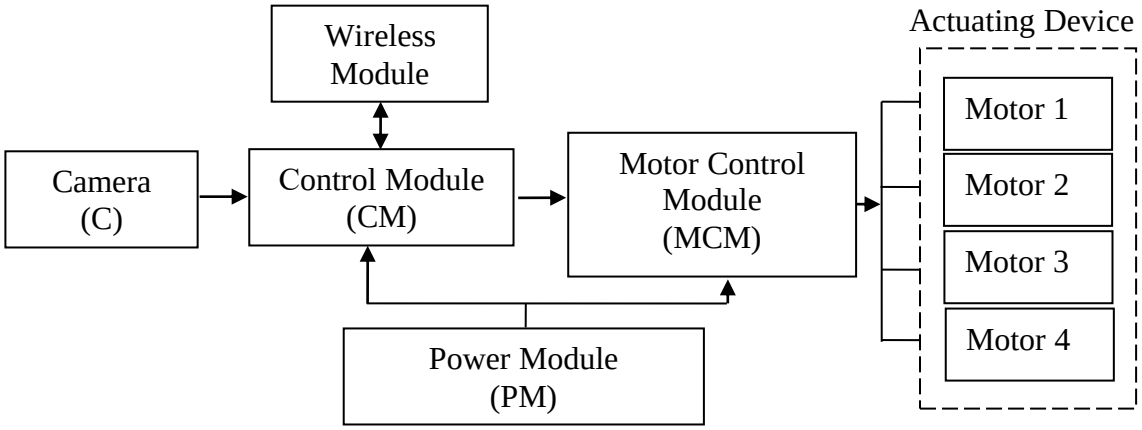


Рисунок 1.5 - Структурна схема керування мобільного робота

Camera (C) – необхідний елемент для забезпечення можливості візуального сприйняття оточуючого середовища. Цей модуль дозволяє роботу отримувати зображення з оточуючого середовища і використовувати їх для прийняття рішень, таких як навігація, виявлення об'єктів або розпізнавання обличь.

Control Module (CM) - відповідає за керування рухом і поведінкою робота. Цей модуль отримує вхідні дані від сенсорів та інших джерел, обробляє їх і видає відповідні команди для керування моторами, сервоприводами та іншими актюаторами. Крім того, модуль керування може включати в себе алгоритми навігації, планування маршруту та керування взаємодією з оточуючим середовищем.

Wireless Module - дозволяє роботу комунікувати з іншими пристроями, наприклад, з комп'ютером, смартфоном або іншими роботами. Бездротовий зв'язок використовується для передачі даних, команд керування, отримання відео потоку з камери робота, взаємодії з користувачем через мобільний додаток або веб-інтерфейс, а також для реалізації бездротових мереж для спільної роботи кількох роботів.

Power Module (PM) – модуль живлення , в мобільному роботі відіграє важливу роль у забезпеченні живлення всіх компонентів робота. Він відповідає за перетворення напруги з джерела живлення на потрібні рівні для живлення моторів, контролерів, сенсорів та інших електронних пристроїв. Модуль живлення також може включати захисні функції, які запобігають пошкодженню робота від перенапруги або перевантаження.

Motor Control Module (MCM) - модуль керування двигуном в мобільному роботі відповідає за керування рухом і поведінкою моторів робота. Цей модуль приймає команди від модуля керування (Control Module) і видає відповідні сигнали для керування швидкістю, напрямком та іншими параметрами руху моторів. Модуль керування двигуном може включати в себе додаткові функції, такі як контроль струму, захист від перегріву або вбудований ПІД-регулятор для точного керування рухом.

Motor1,…,Motor4 - використовуються для приводів, які забезпечують рух робота. Кожен мотор відповідає за привід певної частини робота і керується за допомогою модуля керування двигуном (Motor Control Module, MCM). Разом ці мотори дозволяють роботу рухатися вперед, назад, повертати та виконувати різні маневри в залежності від програми керування.

1.4 Аналіз та вибір апаратних модулів

Відповідно до розробленої структурної схеми керування мобільного робота яка представлена на рисунку 1.5, наступним кроком необхідно провести аналіз та обрати апаратні модулі щоб реалізувати систему керування. При виборі апаратного модуля для реалізації системи керування необхідно враховувати кілька ключових аспектів [19,20]:

- сумісність з мікроконтролером. Модуль повинен бути сумісним з мікроконтролером, який використовується в системі керування;
- модуль Wi-Fi або Bluetooth. Якщо система потребує бездротового зв'язку, необхідно вибрати модуль з підтримкою Wi-Fi або Bluetooth;
- інтерфейси вводу-виводу (GPIO). Модуль повинен мати достатню кількість GPIO для підключення до потрібних пристроїв та сенсорів;
- ємність пам'яті. Для зберігання програмного забезпечення та даних модуль повинен мати достатню пам'ять;
- енергоефективність. Модуль повинен бути енергоефективним, особливо якщо він працює від акумулятора;
- вартість. Вартість модуля також є важливим фактором при виборі.
- розмір і форм-фактор. Розмір модуля повинен бути прийнятним для вбудованої системи;
- можливість розширення. Модуль повинен мати можливість розширення функціональності за необхідності;

- наявність вбудованої підтримки мережі. Деякі модулі можуть мати вбудовану підтримку мережевих протоколів, що полегшує їх інтеграцію в систему керування.

Виходячи з вище перерахованих ключових аспектів вибору модуля системи керування, розглянуто наступні модулі, які представлені на рисунку 1.6, а порівняння технічних характеристик приведено в таблиці 1.1.

а) б) в)

а) ESP32-Cam[21]; б) pyAI-K210 [22]; в) Raspberry Pi Zero 2 W [23]

Рисунок 1.6 –Загальний вигдля апаратних модулів

Таблиця 1.1 – Порівняння технічних характеристик апаратних модулів ESP32-Cam, pyAI-K210 та Raspberry Pi Zero 2 W.

Параметри	Апаратні модулі		
	Raspberry Pi Zero 2 W [23]	pyAI-K210 [22]	ESP32-Cam [21]
1	2	3	4
Мікроконтролер	Broadcom BCM2710A1, Cortex-A53 (ARMv8-A) 64-bit SoC	ESP32 (240 МГц)	ESP32 (240 МГц)
Wireless Communication Module	802.11 b/g/n/ac Wi-Fi, Bluetooth 5.0	+ (Wi-Fi)	+ (Wi-Fi)

Продовження таблиці 1.1

1	2	3	4
Максимальна роздільна здатність	1920 x 1080 (Full HD)	UXGA 1600x1200 (2 мегапікселя)	UXGA 1600x1200 (2 мегапікселя)
Підтримка форматів	H.264, H.265 (HEVC)	UXGA,SXG, SVGA, QVGA, CIF, QCIF	UXGA,SXG, SVGA, QVGA, CIF, QCIF
Швидкість передачі відеопотоку	До 30 кадрів за секунду	UXGA/SXGA 15-30 fps., SVGA 30 fps., CIF - 60 fps.	UXGA/SXGA 15-30 fps., SVGA 30 fps., CIF - 60 fps.
Доступні кодування кольору:	YUV, RGB	YUV(422/ 420)/YCnCr422, RGB565/555, 8-бітні стислі дані	YUV(422/ 420)/YCnCr422, RGB565/555, 8-бітні стислі дані
Напруга	5 Вольт	від 3.6 до 6 В.	5В
Розміри плати	65 x 30 мм	38x42mm	27*39 мм
Вага	9г	18г	10г
Ціна	~20-25$	~50$	~8-10$

Обґрунтування вибору модуля ESP32-Cam для системи керування мобільним роботом може бути наступним [24]:

- інтегрована камера. ESP32-Cam має вбудовану камеру, що дозволяє спростити процес інтеграції та зменшити загальну вартість системи;

- низька вартість. ESP32-Cam є більш доступним в порівнянні з Raspberry Pi Zero 2 W та ruAI-K210, що робить його привабливим варіантом для бюджетних проектів;

- енергоефективність. ESP32-Cam відомий своєю енергоефективністю, що може бути важливим для мобільних роботів, особливо якщо вони працюють від акумулятора;

- достатня продуктивність. ESP32-Cam має достатню продуктивність для багатьох завдань керування мобільним роботом та обробки відео;

- легкість використання. ESP32-Cam може бути легко програмований за допомогою Arduino IDE та має підтримку багатьох бібліотек;

- наявність необхідних інтерфейсів. ESP32-Cam має необхідні інтерфейси для підключення до додаткових пристроїв та сенсорів, що дозволяє розширити можливості системи керування;

- компактний розмір. ESP32-Cam має компактний розмір, що дозволяє легко інтегрувати його в мобільний робот без зайвого обсягу.

Інтерфейс GPIO (General Purpose Input/Output) в модулі ESP32-Cam - це набір пінів, які можна використовувати для зчитування введених даних (вхідні піни) або для виведення сигналів (вихідні піни) [25]. GPIO піни можуть використовуватися для підключення до зовнішніх пристроїв, таких як сенсори, вивідні пристрої, реле, тощо. У модулі ESP32-Cam є певна кількість GPIO пінів, які можуть бути налаштовані для вхідного або вихідного режиму за допомогою програмного забезпечення. Це дозволяє вам взаємодіяти з різними пристроями та компонентами, що розширює можливості використання модуля ESP32-Cam в різних проектах. Загальний вид та призначення пінив шини GPIO на модулі ESP32-Cam представлено на рисунку 1.7

Рисунок 1.7 - Загальний вид та призначення пінив шини GPIO на модулі ESP32-Cam [26]

Наступним кроком необхідно обрати тип камери, яку можна використати для реалізації потокового модуля, і яка сумісна з ESP32-CAM.

Аналіз відкритих торговельних майданчиків показав, що існують такі модулі камер, які представлені на рисунку 1.8, а їхні базові характеристики подано в таблиці 1.2.

a) b) c)

a) Модуль камери OV2640 2Mp/FOV120 (модель HDF3M-811) [27];
b) Модуль камери OV2640 2Mp/FOV70 (модель HW-297) [28]
c) Модуль камери OV5640-AF (модель CV5021-G2) [29]
Рисунок 1.8 - Модулі камер сумісні з ESP32-CAM

Таблиця 1.2 – Порівняння базових характеристик модулів камер сумісні з ESP32-CAM

Параметри	Модулі камер сумісні з ESP32-CAM		
	OV2640 2Mp/FOV120 [27]	OV2640 2Mp/FOV70 [28]	OV5640-AF [29]
Тип матриці	OV2640 2MP	OV2640 2MP	OV5640
Кут огляду	120 градусів	66 градусів	68 градусів
Довжина шлейфу камери (mm)	20	20	20
Максимальна роздільна здатність матриці (відео)	UXGA 1600*1200 (15 fps) / SVGA 800*600 (30 fps)	UXGA 1600*1200 (15 fps) / SVGA 800*600 (30 fps)	QSXGA 2592*1944 (15 fps) / SVGA 800*600 (120 fps)
Підтримка форматів відеозахоплення	8/10-bit Raw RGB, YUV(422/420), RGB565/555	8/10-bit Raw RGB, YUV(422/420), RGB565/555	RawRGB, RGB (RGB565/RGB555 /RGB444), CCIR656, YUV (422/420)
Розміри (mm)	12*12*10	12*12*10	8.5*8.5*10
Ціна ($)	2-4	2-3	3-6

Обґрунтування вибору модуля камери OV2640 2Mp/FOV120 для ESP32-CAM може були такими:

- широкий кут огляду. Модель OV2640 2Mp/FOV120 має широкий кут огляду в 120 градусів, що дозволяє захоплювати більше області інтересу і полегшує візуальне спостереження навколишнього середовища;

- підтримка високої роздільної здатності. Модель має роздільну здатність 2 мегапікселя, що забезпечує деталізовані зображення для більш точного аналізу і розпізнавання об'єктів;

-більш гнучкий для додатків з вимогами до широкого поля зору. Камера з FOV120 може бути більш корисною для додатків, де важлива широка зона покриття, наприклад, для великих приміщень або вуличних систем спостереження;

- покращена можливість виявлення об'єктів. Більший кут огляду дозволяє краще виявляти об'єкти та події в широкому просторі, що корисно для багатьох додатків візуального спостереження та безпеки;

- сумісність з ESP32-CAM. Модуль OV2640 2Mp/FOV120 підтримується ESP32-CAM, що забезпечує легку інтеграцію і використання в проектах, побудованих на цій платформі;

- ефективне використання ресурсів. Хоча модуль може вимагати більше ресурсів для обробки великого обсягу даних, його використання ефективно з точки зору обсягу зображення, яке він може захопити за один кадр;

- збалансована якість і продуктивність. Модель забезпечує збалансовану якість зображення та продуктивність для багатьох застосувань, де важлива широка зона огляду.

Наступним апаратним модулем який необхідно обрати це Motor Control Module (MCM), відповідно до рисунка 1.5 призначення даного модуля це керування двигунами. Виходячи з того що в якості двигунів будуть використовуватися двигуни з редуктором 1:48 DC 3V-6V з наступними технічними характеристиками: номінальний струм – 250мА; робочий струм – 0.35мА, то для керування таким двигунами підходять наступні драйвери

двигунів загальний вид яких представлено на рисунку 1.9, а порівняння їх технічних характеристик в таблиці 1.2

а) б) в)

а) модуль драйвера двигуна L298N [30];

б) модуль драйвера двигуна L9110N [31];

в) модуль драйвера двигуна MX1508 [32].

Рисунок 1.9 - Загальний від апаратних модулів драйверів двигунів

Таблиця 1.2 – Порівняння технічних характеристик драйверів двигунів: L298N, L9110N та MX1508

Характеристика	L298N [30]	L9110S [31]	MX1508 [32]
1	2	3	4
Кількість каналів	2 (підтримка двох двигунів	1 (підтримка одного двигуна)	1 (підтримка одного двигуна)
Напруга живлення	5V - 35V	2.5V - 12V	2.5V - 10V
Максимальний струм	2A	0.8A	1.5A
Метод керування	H-міст (для керування в двох напрямках)	H-міст (для керування в двох напрямках)	H-міст (для керування в двох напрямках)
Мікроконтролери, що підтримуються	Arduino, Raspberry Pi, інші	Arduino, Raspberry Pi, інші	Arduino, Raspberry Pi, інші

Продовження таблиці 1.2

1	2	3	4
Додаткові функції	Регулятор обертального моменту, захист від перевантаження	Відсутні	Відсутні
Розмири	43,5 x 43,2 x 29,4 мм	30 x 24 x 15 мм	24.7*21*5 мм
Ціна	~75-100грн.	~200грн.	~30-50грн.

Як можна бачить з таблиці 1.2, модуль драйвера двигунів L298N є більш потужним і функціональним порівняно з модулем L9110S. L298N підтримує два двигуни одночасно і може керувати їх у двох напрямках (вперед і назад) з допомогою PWM сигналу. Він також має вбудований регулятор обертального моменту і захист від перевантаження, що дозволяє ефективно керувати потужними двигунами. Модуль MX1508 також може керувати двигунами у двох напрямках, але він має меншу потужність і можливості порівняно з L298N. Він підтримує тільки один двигун і не має вбудованого регулятора обертального моменту або захисту від перевантаження, що робить його менш універсальним для потужних застосувань. У порівнянні з L9110S, L298N теж має більше можливостей, таких як керування в двох напрямках і підтримка двох двигунів одночасно. L9110S, незважаючи на свою простоту, може бути корисним для менших проектів або там, де вимоги до потужності не такі великі. Отже, виходячи з умов технічного завдання нам потрібен більш потужний і універсальний драйвер для керування потужними двигунами, L298N буде кращим вибором.

Наступнім кроком необхідно обрати апаратні модулі для реалізацій Power Module (PM) відповідно до розробленої структурної схеми (рис.1.2). Для цього пропонується використовувати наступні апаратні модулі BMS контролер зарядки для акумуляторів 18650 серій. BMS (Battery Management System) контролер - це пристрій, який використовується для керування та контролю літій-іонними або літій-полімерними акумуляторами. Основна

функція BMS - це забезпечення безпеки та надійності під час зарядки та розрядки акумулятора. Проведемо аналіз та вибір модуля BMS, загальній вид обраних модулів представлені на рисунку 1.10, а порівняння їх технічних характеристик представлено в таблиці 1.3

а) б) в)

а) Підвищуючий модуль заряду Li-on 18650 (2S 1A/2A/4A 8.4B) Type-C[33];

б) Модуль зарядки Type-C для Li-ion акумуляторів TP4056[34];

в) Контролер заряду акумуляторів на IP2312, 5B/3A, Type-C[35].

Рисунок 1.10 - Загальній від апаратні модулів BMS контролерів

Таблиця 1.3 - Порівняння технічних характеристик модулів BMS контролерів

Характеристика	Апаратні модулів BMS контролерів		
	Підвищуючий модуль заряду Li-on 18650 (2S) Type-C [33]	Модуль зарядки Type-C для Li-ion акумуляторів TP4056[34]	Контролер заряду акумуляторів на IP2312, 5B/3A, Type-C [35]
1	2	3	4
Максимальна потужність (Вт)	8.4B, 1A/2A/4A	4.2B, 1A	5B, 3A
Кількість клітин (S)	2	1	1
Тип інтерфейсу зарядки	Type-C	Type-C	Type-C
Захист від перевантаження	Так	Так	Так
Захист від недозаряду	Так	Так	Так

Продовження таблиці 1.3

1	2	3	4
Балансування клітин	Ні	Ні	Ні
Розмири	39x18x6.3 мм.	17 x 26 мм.	23 x 11 мм
Ціна	~67 грн.	~18 грн.	~40 грн

Як можна побачити з таблиці 1.3, модуль BMS контролера заряду Li-on 18650 (2S) Type-C може бути кращим вибором порівняно з модулем модуль зарядки Type-C для Li-ion акумуляторів TP4056 або Контролер заряду акумуляторів на IP2312, 5В/3А, Type-C з кількох причин. По-перше, підвищуючий модуль заряду Li-on 18650 (2S) Type-C підтримує вищу максимальну потужність (8.4В, 1А/2А/4А), що може бути корисним для швидшого заряджання або для використання з потужнішими акумуляторами. Крім того, якщо потрібно заряджати більше однієї клітини акумулятора, підвищуючий модуль заряду Li-on 18650 (2S) Type-C підтримує 2 клітини, що може бути важливим для певних застосувань. Також він може мати більше функцій захисту, таких як захист від перевантаження, недозаряду, короткого замикання тощо, що забезпечує більшу безпеку для акумуляторів та пристроїв. Нарешті, модуль підвищуючий модуль заряду Li-on 18650 (2S) Type-C може бути більш універсальним застосуванням, оскільки підтримує більше клітин і вищу потужність, що робить його більш гнучким у використанні.

1.5 Розрахунок витрат енергії для керування рухом мобільного робота

Для розрахунку витрат енергії спочатку визначимо загальну потужність, що споживається системою [36-38]:

- витрата енергії для керування рухом (Витрата енергії на одному моторі (P_{motor}))

$$P_{motor} = U \cdot I_{nom} \quad (1.1)$$

де: U - напруга живлення (6В)

I_{nom} - номінальний струм (250мА)

Витрата енергії на всі 4 мотори (P_{all_motor}):

$$P_{all\_motor} = 4 \cdot P_{motor} \quad (1.2)$$

Витрата енергії (P_{camera}) для роботи камери OVE2560

$$P_{camera} = U \cdot I_{work} \quad (1.3)$$

де: I_{work} - робочий струм (0.35мА)

U - напруга живлення (5В)

Витрата енергії для драйвера двигуна L298N ($P_{draiver}$):

$$P_{draiver} = U \cdot I_{draiver} \quad (1.4)$$

де: $I_{draiver}$ - робочий струм драйвера двигуна L298N (2А);

U - напруга живлення (від 5В – 12В)

Внаслідок чого загальна витрата енергії буде дорівнювати сумі виразів 1.1 - 1.4, та розраховуватися по наступній формулі:

$$P_{total} = P_{motors} + P_{camera} + P_{draiver} \quad (1.5)$$

Робочий час системи на одному заряді:

$$T_{work} = \frac{E_{total}}{P_{total} \cdot N_{cells} \cdot V_{cell}} \quad (1.6)$$

де: E_{total} - енергія одного заряду;

N_{cells} - кількість акумуляторів (2*18650)

V_{cell} - напруга одного акумулятора (3.7В)

Підставимо значення в розрахункові формули 1.1 – 1.6 та отримуємо наступні результати розрахунку загальна витрата енергії та робочий час системи на одному заряді

$$P_{total} = P_{motors} + P_{camera} + P_{draiver} = 6Вт + 2.1мВт + 12Вт = 18.1Вт \quad (1.7)$$

$$E_{total} = N_{cells} \cdot V_{cells} = 2 \cdot 3.7В = 7.4Вт \cdot год \quad (1.8)$$

$$T_{work} = \frac{E_{total}}{P_{total}} = \frac{7.4Вт \cdot год}{18.1Вт} \approx 0.409год \approx 24.56хв \quad (1.9)$$

Як можна бачить з 1.9 розроблена система керування живленням мобільного робота на базі ESP32-Cam на базі 2 акумуляторів 19650 серій може працювати в автономному режимі приблизно 24.56 хвилин.

1.6 Розробка схеми підключення апаратних модулів

Розробка схеми підключення при проектуванні мобільного робота на ESP32-Cam є дуже важливою, оскільки вона дозволяє правильно планувати та виконувати підключення всіх компонентів і модулів до мікроконтролера [39]. Перш за все, на схемі будуть показані всі необхідні підключення мікроконтролера ESP32-Cam до живлення, а також до інших електронних компонентів, таких як датчики, мотори, модулі зв'язку тощо. Крім того, схема

може містити підключення датчиків відстані для виявлення перешкод, датчиків керування рухом, а також підключення до модулів Wi-Fi або Bluetooth для бездротового керування. Схема підключення також може містити інші важливі елементи, такі як регулятори напруги для живлення різних компонентів, конденсатори для стабілізації напруги, а також підключення для програмування і налагодження мікроконтролера. Всі ці елементи дозволяють забезпечити ефективну роботу мобільного робота на ESP32-Cam і допомагають уникнути можливих проблем з підключенням і виконанням програми.

Для розробки схеми підключення малогабаритного мобільного робота було запропоновано обрати наступні середовище розробки: Fritzing [40] або Tinkercad [41]. Для вибору середовища розробки було проведено порівнянний аналіз особливостей середовищ розробки Fritzing та Tinkercad, результат якого представлено в таблиці 1.4.

Таблиця 1.4 - Порівнянний аналіз особливостей середовищ розробки Fritzing та Tinkercad

Особливість	Fritzing [40]	Tinkercad [41]
1	2	3
Графічний інтерфейс	Простий у використанні, інтуїтивно зрозумілий	Також простий у використанні з інтуїтивним інтерфейсом
Моделі компонентів	Має обмежений набір моделей компонентів, але дозволяє створювати свої	Має велику бібліотеку готових моделей компонентів
Моделювання схем	Підтримує моделювання електричних схем та PCB	Підтримує моделювання електричних схем та симуляцію поведінки
PCB дизайн	Має інструменти для проектування печатних плат	Має інструменти для проектування печатних плат

Продовження таблиці 1.4

1	2	3
3D-моделювання	Має обмежені можливості 3D-моделювання	Має більш розвинуті можливості 3D-моделювання
Доступність онлайн	Може використовуватися в онлайн-режимі	Повністю онлайн-сервіс
Ціна	Безкоштовний для базової версії, є платні плани для розширених можливостей	Безкоштовний для базової версії, є платні плани для розширених можливостей

Обґрунтування вибору середовища Fritzing для розробки схеми збірки мобільного робота може бути наступним: Fritzing має інтуїтивно зрозумілий і простий у використанні інтерфейс, що робить його ідеальним для початківців та швидкої розробки прототипів. Також важливою перевагою є можливість створення власних компонентів, що дозволяє адаптувати середовище до потреб конкретного проекту. Крім того, Fritzing підтримує проектування печатних плат, що робить його цікавим для проектів, які потребують печатних плат. Наявність активної спільноти користувачів і форуму для отримання підтримки та порад є ще однією перевагою цього середовища. Важливим фактором є також безкоштовність основної версії Fritzing, що робить його доступним для широкого кола користувачів. Також важливою перевагою є можливість використання Fritzing як в онлайн-режимі, так і офлайн, що робить його зручним для роботи в різних умовах. Для невеликих проектів, таких як мобільний робот, Fritzing може бути більш практичним вибором, оскільки він не навантажує зайвою функціональністю. Також важливою перевагою є можливість експорту схеми у зручному для подальшого використання форматі. Хоча Fritzing має простий інтерфейс, він все ж має деякі розширювані можливості для складних проектів. Нарешті, важливою перевагою є можливість співпраці з Tinkercad для розробки більш складних проектів з апаратно-програмною частиною.

Для підключення апаратних модулів ESP32-Cam, модуля L298N, чотирьох двигунів DC 3-6V, BMS модуля заряду Li-on 18650 (2S) Type-C та двох акумуляторів 18650 можна використати таку схему підключення. Спочатку ESP32-Cam підключається до живлення і землі модуля L298N. Піни керування двигунами ESP32-Cam підключаються до відповідних пінів IN1-IN4 модуля L298N. Живлення модуля L298N підключається до живлення і землі BMS модуля. Двигуни підключаються до виходів OUT1-OUT4 модуля L298N. Вихід акумуляторів 18650 підключається до входу живлення BMS модуля, а виходи BMS модуля - до входів живлення модуля L298N. На цьому етапі схема підключення готова до використання. Ця схема дозволяє керувати чотирма двигунами за допомогою ESP32-Cam, використовуючи модуль L298N для керування потужністю, та забезпечує безпеку та оптимальне заряджання акумуляторів за допомогою BMS модуля. Розроблена схема підключення апаратних модулів мобільного робота на ESP32-Cam представлена на рисунку 1.11.

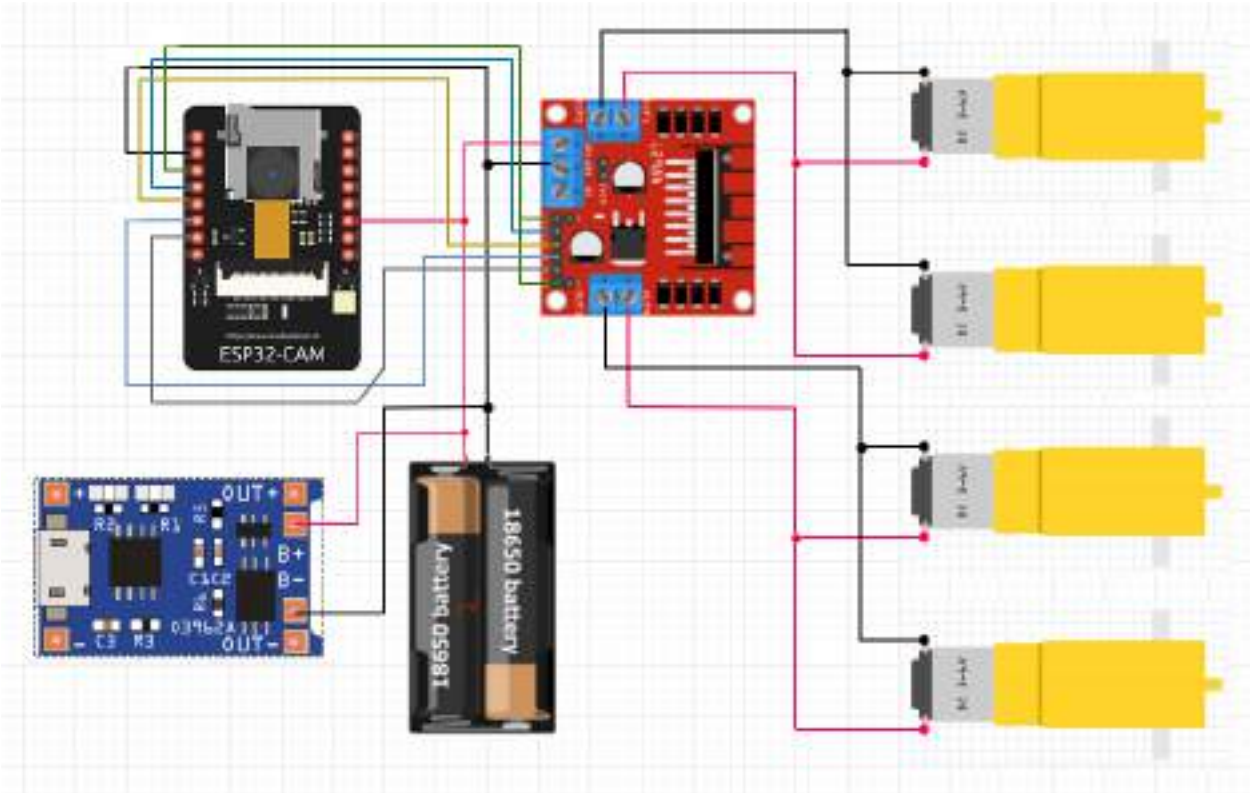


Рисунок 1.11 – Схема підключення апаратних модулів мобільного робота

Розроблена схема підключення має декілька переваг. По-перше, вона дозволяє ефективно керувати чотирма двигунами за допомогою ESP32-Cam, що відкриває широкі можливості для реалізації різних рухомих або маніпуляційних функцій у мобільному роботі. Крім того, використання модуля L298N для керування потужністю дозволяє забезпечити стабільну роботу двигунів та захист від перенапруги чи короткого замикання. Також, використання BMS модуля для заряду акумуляторів 18650 забезпечує безпеку під час заряджання та розряджання акумуляторів, що є важливим для підтримки їхньої тривалої роботи. Крім того, така схема підключення досить компактна та ефективна, що дозволяє зберігати простір у корпусі мобільного робота для інших компонентів або модулів. В цілому, ця схема підключення допоможе забезпечити стабільну та ефективну роботу мобільного робота на базі ESP32-Cam з чотирма двигунами та акумуляторами 18650.

1.7 Розробка 3D моделей конструкцій мобільного робота

Обґрунтування вибору AutoDesk Tinkercad для розробки 3D моделей конструкцій мобільного робота може бути наступним [42]. По-перше, Tinkercad є безкоштовним та відомим середовищем для моделювання 3D, що робить його доступним для широкого кола користувачів, включаючи початківців та студентів. Він має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє легко створювати складні 3D моделі без необхідності великого досвіду у дизайні. Крім того, Tinkercad має широкий вибір інструментів для створення різноманітних деталей та конструкцій, що дозволяє реалізувати різні ідеї та концепції для мобільного робота. Додатково, в Tinkercad можна імпортувати готові 3D моделі або експортувати свої проекти для подальшого використання у програмах для 3D друку або розробки. Також важливою перевагою є можливість спільної роботи над проектами, що дозволяє командам спільно працювати над

розробкою конструкцій мобільного робота. В цілому, AutoDesk Tinkercad є потужним та зручним інструментом для розробки 3D моделей конструкцій мобільного робота, який може задовольнити потреби як початківців, так і досвідчених користувачів.

Використовуючи вбудовані функцій AutoDesk Tinkercad 3D були побудовані всі конструкторські елементи відповідно до розробленого концепт мобільного робота (рис.1.3). Приклад розробки 3D моделі кріплення відомого котка мобільного робота в середовищі 3D проектування AutoDesk Tinkercad представлене на рисунку 1.12

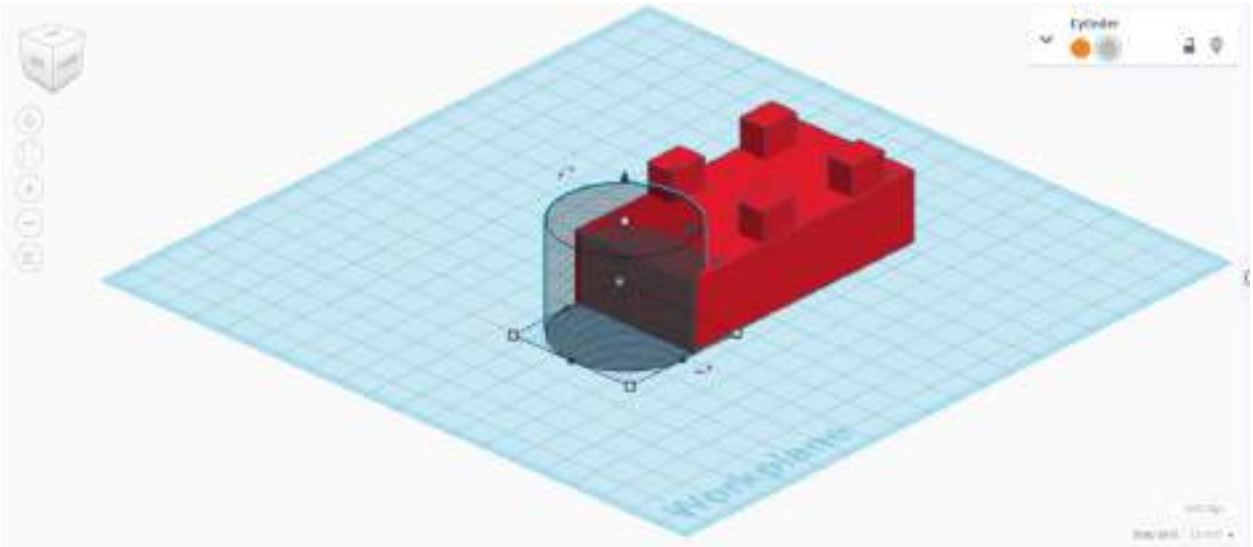
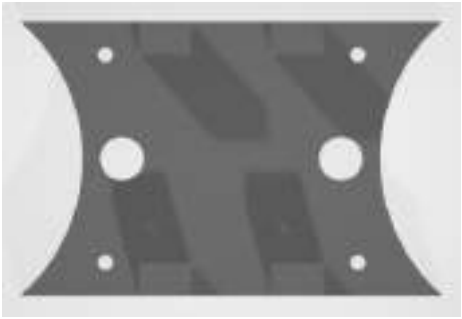
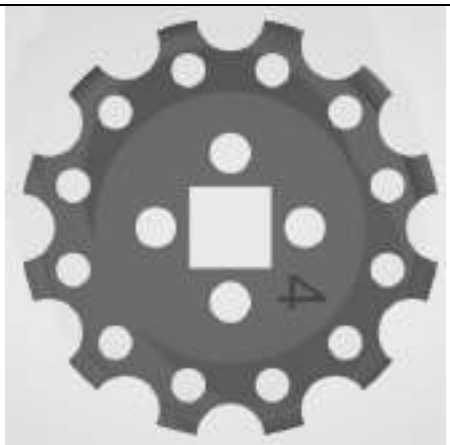
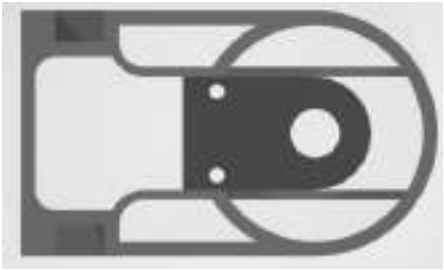


Рисунок 1.12 - Приклад розробки 3D моделі кріплення відомого котка мобільного робота в середовище 3D проектування AutoDesk Tinkercad

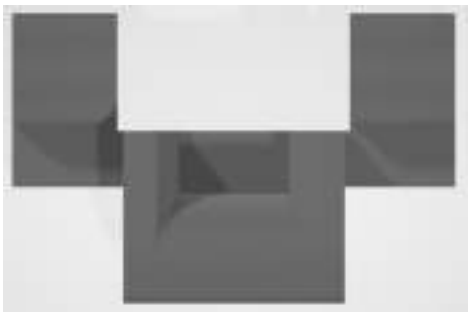
Як можливо бачить з рисунку 1.12 для проектування деталі кріплення відомого котка мобільного робота можна скористатися такими методами. Почніть зі створення базової форми, яка відповідатиме розмірам та формі кріплення. Додайте необхідні додаткові елементи, такі як кільця або проточки, для забезпечення правильного кріплення до іншої деталі. Використовуйте можливості об'єднання та віднімання форм для створення потрібної форми кріплення. Редагуйте форму, додавайте деталі та коригуйте розміри, щоб добитися точності та функціональності кріплення. Використовуйте функцію "Лінійка" для створення точних вимірів та

розміщення отворів для кріплення. Використовуючи данні функцій були розроблені наступні деталі мобільного робота, які представлені в таблиці 1.5

Таблиця 1.5 - Загальний вид та опис призначення розроблених 3D моделей деталей конструкцій мобільного робота

Загальний від 3D моделі деталі	Призначення
1	2
	Кріплення відомого котка мобільного робота використовується для монтажу відомого або ведучого котка. Також воно може застосовуватися для захисту внутрішніх компонентів робота та покращення його зовнішнього вигляду.
	Ведучий коток в гусеничному мобільному роботі - це елемент конструкції, який відповідає за керування рухом робота. Він може бути здійснений у вигляді спеціальної гусеничної системи з моторами та іншими компонентами, що дозволяють роботу переміщатися вперед, назад або обертатися на місці.
	Елемент для кріплення DC двигунів для мобільного робота - це конструкційний елемент, який призначений для закріплення DC двигунів на рамі або іншій частині робота. Цей елемент зазвичай має отвори або точки для кріплення двигунів за допомогою болтів чи інших кріпильних засобів. Він забезпечує надійне кріплення двигунів і дозволяє їх встановити у відповідному положенні для правильної роботи механізму передачі руху або іншої механічної системи.

Продовження таблиці 1.5

1	2
	Елемент захисту ведучого колеса мобільного гусеничного робота - це конструкційний елемент, який призначений для захисту ведучого колеса від пошкоджень та зносу під час руху по різноманітним поверхням.
	Ланка гусеничного тракту мобільного робота - це один із елементів конструкції гусеничної системи, який складається з гусениці та кріпильних елементів, що з'єднують гусеницю з рамою робота. Ланка гусеничного тракту відповідає за передачу руху від приводу (наприклад, мотора) до гусениці, забезпечуючи тим самим рухомість робота по різних поверхнях.
	Дозволяє приводити в рух ведуче колесо за допомогою обертання осі двигуна
	Фронтальна деталь мобільного робота, з отворами під камеру та датчиків.

1.8 Розробка 3D моделей складання мобільного робота

Наступним етапом є розробка деталізованої 3D моделі складання мобільного робота на базі ESP32-Cam. Вибір CAD-системи SolidWorks для розроблення 3D-моделі мобільного робота обґрунтовано її широкими можливостями в галузі механічного проєктування та багаторічними успіхами в інженерній сфері. SolidWorks надає інтуїтивний інтерфейс і потужні інструменти, даючи змогу створювати складні механічні конструкції з високою точністю [43-45]. Її інтеграція з системами CAM забезпечує ефективну підготовку до 3D-друку, роблячи SolidWorks оптимальним вибором для розроблення інноваційних і технічно складних проєктів, таких як мобільні роботи. На базі CAD-системи SolidWorks було спроектовано деталізовану збірку малогабаритного мобільного робота, "Explosion" модель представлено на рисунок 1.13

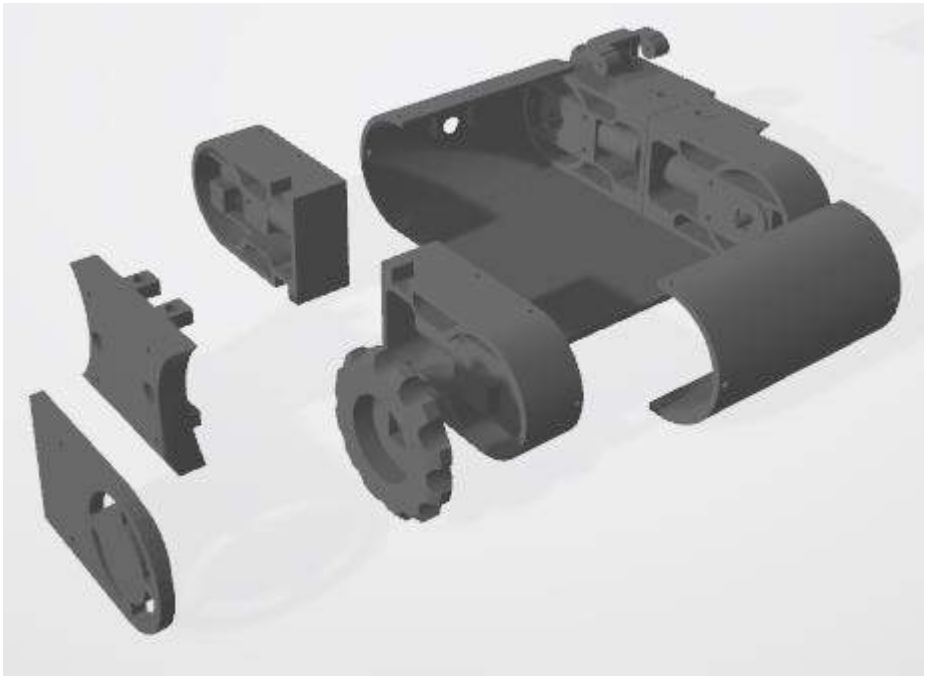


Рисунок 1.13 - "Explosion " 3D модель складання мобільного робота

Як можна побачити з рисунку 1.13 "Explosion" 3D модель складання мобільного робота показує всі складові частини та їх взаємозв'язок в розірваному вигляді. Це дозволяє побачити, як різні компоненти взаємодіють

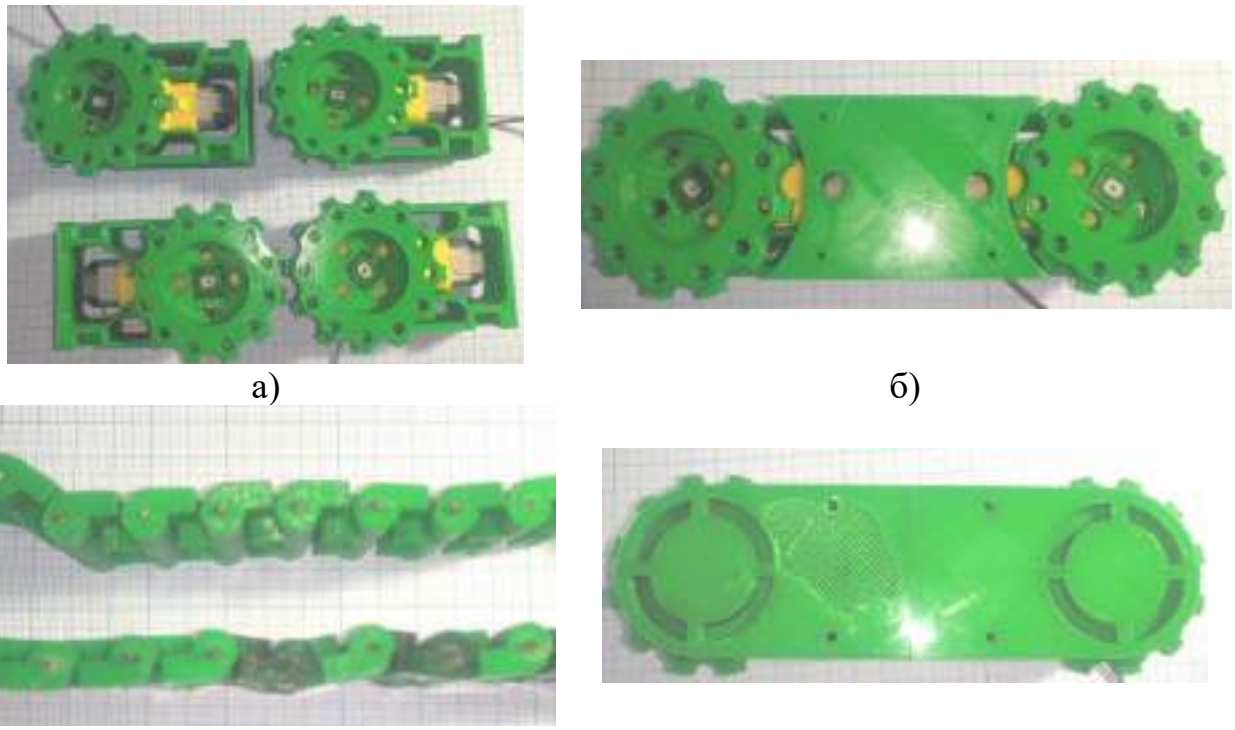
та як вони розташовані один відносно одного всередині робота. Така модель може бути корисною для розуміння принципу дії робота, для виявлення можливих проблем або для навчання складання та обслуговування робота.

1.9 Друк деталей мобільного робот за допомогою 3D принтера

Для отримання фізичних конструкцій спроектованого мобільного робота для складання макета, скористаємося методом адитивних технологій-3D друку [46]. Розроблені моделі деталей за допомогою програми UltiMaker Cura 5.2.2 підготуємо до друку [47]. Для цього необхідно встановити наступні налаштування 3D друку, які представлені на рисунку 1.14. При налаштуванні 3D друку в програмі Ultimaker Cura 5.2.2 для друку PLA пластиком потрібно врахувати кілька важливих аспектів. Перш за все, важливо правильно вибрати налаштування для матеріалу та типу друкарської головки. Для PLA зазвичай використовують температуру друку в діапазоні 190-220 градусів Цельсія, але в нашому відпадку температура складає 245 градусів Цельсія [48]. Далі, слід уважно налаштувати швидкість друку, шаруватість та заповнення моделі. Для PLA рекомендується використовувати швидкість друку близько 50-70 мм/с, шаруватість від 0,1 до 0,3 мм та заповнення від 20% до 50%, залежно від потреби у міцності деталі. Також важливо налаштувати підтримку для моделі, якщо вона має виступаючі елементи. Підтримка допомагає уникнути деформацій та допомагає створити детальну модель.

Рисунок 1.14 - Налаштування 3D друку мобільного робота в програмі UltiMaker Cura 5.2.2

Отриманий результат 3D друку розроблених моделей деталей мобільного робота представлені на рисунку 1.15.



а) б)

в) г)

а) Кріплення DC двигунів з катками; б) Передній та задній каток; в) Гусениця мобільного робота у зборі; г) Зібраний елемент рушія мобільного робота.

Рисунок 1.14 - Отриманий результат 3D друку розроблених моделей деталей мобільного робота

1.10 Складання макета мобільного робота

Відповідно до розробленої структурної схеми (рис.1.5) та схеми підключення апаратних модулів мобільного робота (рис.1.11), проведемо складання макету мобільного робота на базі ESP32-Cam. Внутрішня компоновка апаратних модулів в середині мобільного робота представлена на рисунку 1.15.

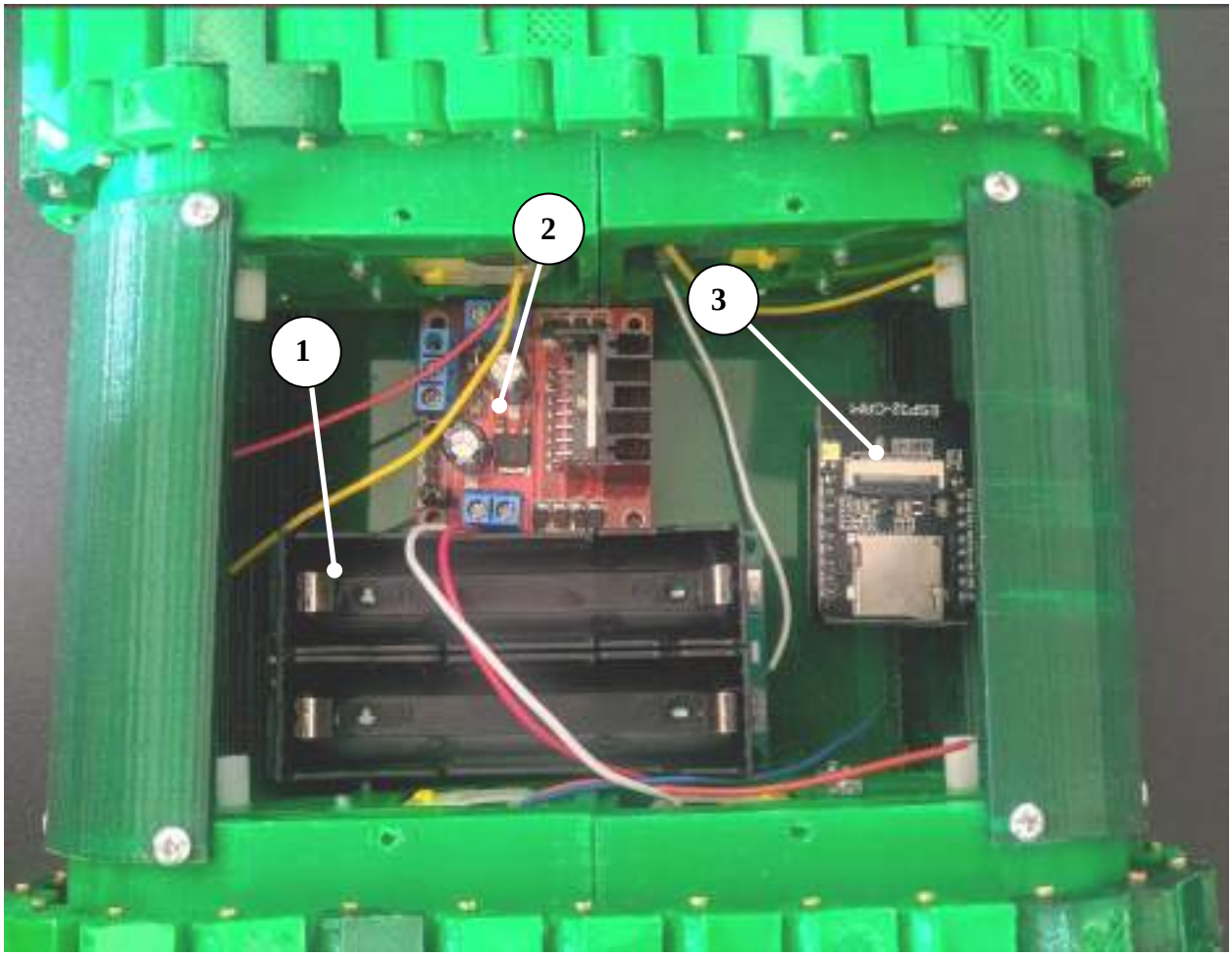
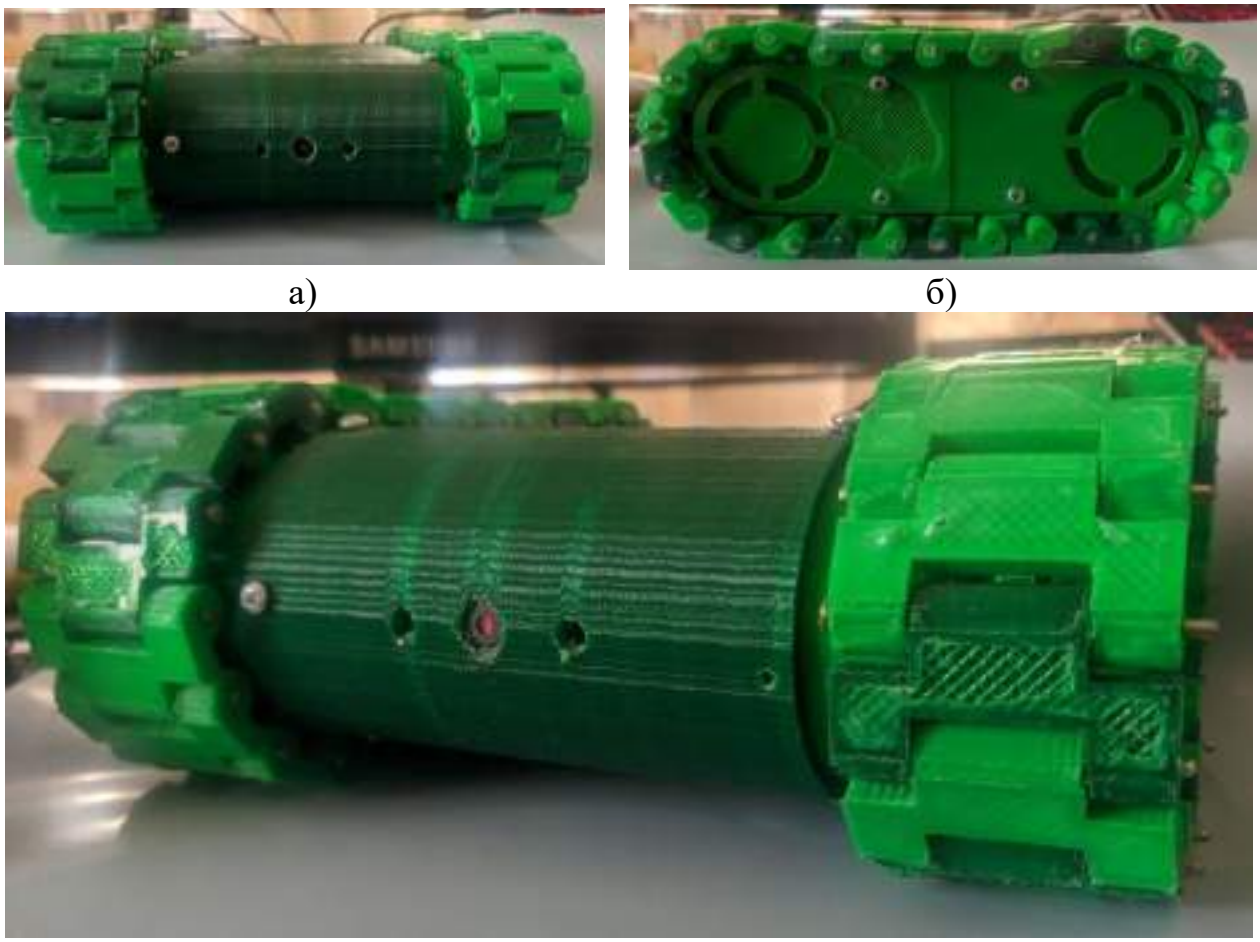


Рисунок 1.15 - Внутрішня компоновка апаратних модулів в середині мобільного робота

1- Модуль живлення мобільним роботом (2*акумулятора 18650), на зворотній стороні знаходиться BMS модуль зарядки Type-C для Li-ion акумуляторів TP4056; 2 – модуль драйвера двигуна L298N; 3 – система керування мобільним роботом на базі ESP32-Cam з підключеною камерою

OVE2650. Загальний від зібраного макету мобільного робота представлені на рисунку 1.16.



а) б)

с)

а) вид спереду ; б) вид з боку; с) вид ізометрія

Рисунок 1.11 - Загальний вид зібраного макету мобільного робота

2 РОЗРОБКА СИСТЕМИ КЕРУВАННЯ МОБІЛЬНИМ РОБОТОМ

2.1 Аналіз та вибір середовища розробки

Для розробки програмного забезпечення для модуля ESP32-Cam можна використовувати різні середовища розробки, які мають свої переваги та недоліки. Розглянемо кілька популярних середовищ та їх характеристики:

- Arduino IDE. Просте та легке у використанні середовище розробки, яке підтримує ESP32-Cam. Має велику спільноту користувачів та багато доступних бібліотек [49];

- PlatformIO. Розширене середовище розробки з великим функціоналом для роботи з ESP32-Cam. Підтримує багато мікроконтролерів та платформ [50];

- Espressif IDF. Офіційне середовище розробки для ESP32 від Espressif. Має великий набір інструментів для розробки різних додатків для ESP32-Cam [51];

- Visual Studio Code з розширенням PlatformIO. Потужне середовище розробки, яке поєднує можливості Visual Studio Code з функціоналом PlatformIO для роботи з ESP32-Cam [52];

- Arduino Web Editor. Веб-середовище розробки, яке дозволяє працювати з Arduino без необхідності встановлення додаткових програм на комп'ютері [53];

- Espressif IoT Development Framework (ESP-IDF). Низькорівневе середовище розробки для ESP32, яке надає прямий доступ до функціоналу мікроконтролера [54];

- MicroPython. Мова програмування та середовище розробки, які дозволяють розробляти програми для ESP32-Cam за допомогою Python [55].

Arduino IDE є привабливим вибором для розробки програмного забезпечення для мобільного робота на базі ESP32-Cam з кількох причин.

По-перше, його простота використання робить його ідеальним для початківців у програмуванні та робототехніці. Arduino IDE має офіційну підтримку для ESP32, що дозволяє легко розробляти програмне забезпечення для цього модуля. Крім того, велика кількість готових бібліотек у спільноті Arduino спрощує розробку різних функцій для мобільного робота.

Додатково, активна спільнота користувачів Arduino може надати допомогу та поради щодо розробки. Arduino IDE підтримує роботу на різних операційних системах і має вбудований текстовий редактор, що робить його зручним для роботи на будь-якому комп'ютері.

Також, Arduino IDE підтримує завантаження програмного забезпечення на ESP32-Cam через USB, що спрощує процес розробки. Інші переваги включають доступність документації та прикладів коду, легкість установки та оновлення, а також широкий вибір платформ та модулів, включаючи ESP32-Cam.

2.2 Розробка алгоритму керування мобільним роботом

Підхід "тонкий клієнт" (Thin client) має на увазі, що обчислення й оброблення даних здійснюються не на самому пристрої (у цьому випадку - на мобільному роботі), а на віддаленому сервері або хмарній платформі [56]. При цьому пристрій слугує тільки для передачі команд і отримання результату обробки.

Під час розроблення алгоритму керування мобільним роботом на базі підходу "тонкий клієнт" слід враховувати такі особливості [57]:

- необхідність стабільного і високошвидкісного інтернет-з'єднання: оскільки обробка даних і обчислення відбуваються на віддаленому сервері, пристрій повинен мати доступ до мережі Інтернет з високою швидкістю і стабільністю;

- низький рівень затримки: для керування мобільним роботом у режимі реального часу необхідно мінімізувати затримки між передачею команд і

отриманням результату обробки. Для цього може знадобитися оптимізація мережевих протоколів і використання спеціалізованих технологій, таких як WebSockets;

- обмеженість ресурсів пристрою: оскільки мобільний робот є тонким клієнтом, йому не потрібне потужне апаратне забезпечення і великий обсяг пам'яті. Це дає змогу використовувати дешевші та компактніші пристрої для керування мобільним роботом;

- надійність і безпека: оскільки вся обробка даних і обчислення відбуваються на віддаленому сервері, необхідні відповідні заходи для захисту інформації та забезпечення безпеки передачі даних;

Необхідність гнучкості та масштабованості: підхід "тонкий клієнт" може бути використаний для керування безліччю мобільних роботів одночасно. Але варто зауважити що, в даному випадку функціональною особливістю роботи з ESP32-Cam як мікроконтролерного пристрою, є можливість створення однієї "точки доступу" для одного оператора. Виходячи з цього пропонується наступний алгоритм керування мобільним роботом, загальний вигляд якого представлений на рисунку 2.1.

Як можна бачити з рис. 2.1, загального алгоритму керування мобільного робота, система керування представлена у вигляді двох незалежних алгоритмів, як мобільного робота так і оператора мобільного робота. Опишемо основні елементи алгоритму:

1. library initialization - ініціалізація бібліотек для роботи з ESP32-Cam (esp_camera.h, WiFi.h, Arduino.h, sp_http_server.h тощо), що дасть можливість реалізувати всі основні функції;

2. LAN Connections - підключення апаратних модулів до локальної бездротової мережі (Wi-Fi);

3. Starting the Web Server - ініціалізація та запуск Web Server на базі ESP32-Cam;

4. Camera Initialization - ініціалізація та запуск передачі потокового відео з камери ESP32-Cam;

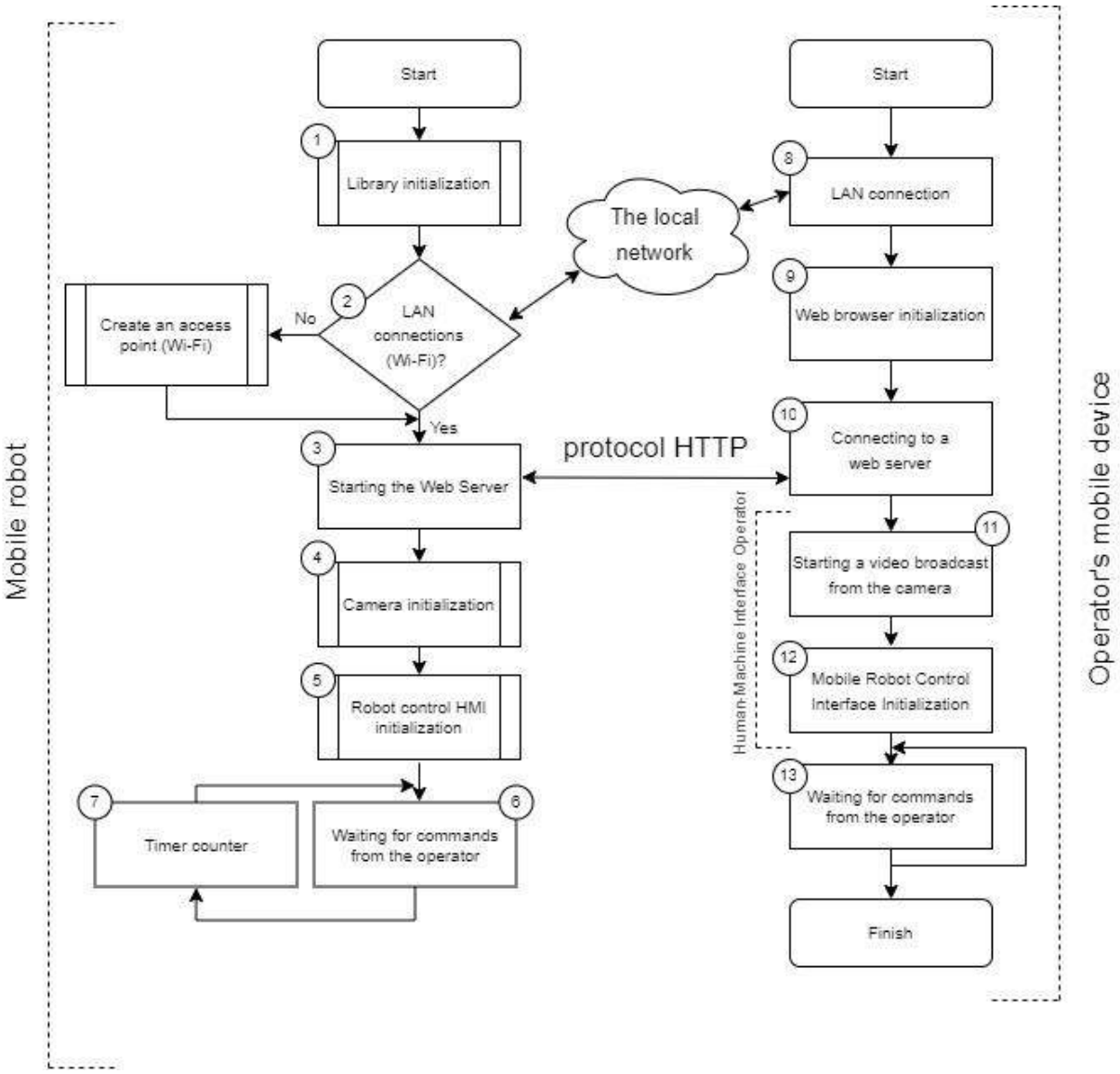


Рисунок 2.1 – Загальний алгоритм керування мобільним роботом

5. Robot Control HMI Initialization - ініціалізація інтерфейсу керування переміщенням мобільної платформи;

6. Timer Counter - лічильник часу, для поновлення очікування команд оператора в часі

7. Waiting for Commands from the Operator - процедура очікування команд від оператора мобільним маніпуляційним роботом.

Основні елементи алгоритму Operator`s Mobile Device:

8. LAN Connections - функцій підключення до локальної бездротової мережі;

9. Web Browser Initialization - ініціалізація Web Browser встановленого на мобільному пристрої оператора

10. Connecting to a Web Server - встановлення під'єднання до Web Server на модулі ESP32-Cam у рамках HMI;

11 Starting a Video Broadcast - запуск функцій відображення відеопотоку в HMI оператора;

12. Mobile Robot Control Interface Initialization - ініціалізація та відображення команд керування мобільним роботом;

13. Waiting for Commands from the Operator - процедура очікування команд від оператора.

Передача даних між мобільним роботом і пристроєм керування оператором відбувається на базі концепцій "тонкого клієнта" клієнт-сервер архітектури, на базі протоколу HTTP. Що дасть змогу не встановлювати на мобільний пристрій оператора додаткове програмне забезпечення.

Недоліками використання системи керування мобільним роботом на базі ESP32-Cam на базі концепцій "тонкого клієнта" можуть бути [58]:

- залежність від мережі. Тонкий клієнт потребує постійного підключення до мережі, що може бути проблематичним у випадку незадовільного покриття мережі або відсутності доступу до неї в деяких місцях.

- безпека даних. Передача даних між тонким клієнтом і сервером може створювати проблеми з безпекою даних, особливо якщо дані недостатньо зашифровані або захищені.

- пропускна здатність мережі. Використання тонкого клієнта може призвести до збільшення навантаження на мережу, особливо якщо багато клієнтів спільно використовують ресурси сервера.

- залежність від сервера. Тонкий клієнт потребує постійного доступу до сервера для виконання більшості операцій, що може стати проблемою у випадку відмови сервера або недоступності мережі.

- складність розгортання. Налаштування і розгортання інфраструктури для підтримки тонких клієнтів може бути складним та вимагати додаткових витрат часу і ресурсів.

- оновлення програмного забезпечення. Оновлення програмного забезпечення тонкого клієнта може бути складним і вимагати втручання користувача, особливо якщо це вимагає встановлення нових версій або патчів.

- відсутність автономності: Тонкий клієнт зазвичай не може працювати в автономному режимі без підключення до сервера або мережі, що обмежує його функціональність у відсутність мережі.

Але всі вище перераховані недоліки, не впливають на результат даного дослідження, в наслідок цього концепт «тонкого клієнта» підходить для вирішення поставленого завдання.

2.3 Програмна реалізація системи керування

Програмна реалізація системи керування мобільним роботом на базі ESP32-Cam має кілька особливостей. ESP32-Cam має вбудований Wi-Fi, що дозволяє керувати роботом через бездротову мережу. Також може використовуватися для отримання відеопотоку з камери, що дозволяє візуалізувати оточуюче середовище робота. ESP32-Cam підтримує роботу з різними датчиками, такими як акселерометр чи гіроскоп, для отримання додаткової інформації про оточуюче середовище. Має низьке споживання енергії, що робить його ідеальним вибором для мобільних роботів. Програмна реалізація може включати алгоритми для автономного керування, використовуючи зібрані датчиками дані. Через наявність вбудованого камери, можливе використання комп'ютерного зору для вирішення завдань розпізнавання об'єктів або навігації. ESP32-Cam має велику спільноту розробників і підтримується широким спектром бібліотек, що полегшує розробку програмного забезпечення для робота. Для забезпечення безпеки

мережі іноді може знадобитися додаткова конфігурація ESP32-Cam. Важливо враховувати обмежені ресурси ESP32-Cam при розробці програмної частини системи керування. Для забезпечення надійності системи можуть використовуватися додаткові алгоритми контролю та відновлення роботи. Проект в середовищі Arduino IDE буде складатись з 2 файлів: файл esp32cam.ino та app_httpd.cpp. Файл esp32cam.ino базовій файл керування мобільним роботом, а файл app_httpd.cpp буде візуалізувати HMI інтерфейс керування для оператора. Почнемо з розробки файлу esp32cam.ino. Для реалізації розробленого алгоритму керування в підрозділі 2.3 в середовищі Arduino IDE в прещу чергу необхідно підключити наступні бібліотеки :

```
#include "esp_wifi.h"
#include "esp_camera.h"
#include <WiFi.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h
```

Даний фрагмент коду містить підключення бібліотек, які необхідні для роботи з Wi-Fi та камерою на платформі ESP32-Cam:

- #include "esp_wifi.h" - це бібліотека для роботи з Wi-Fi модулем ESP32, яка дозволяє встановлювати з'єднання з бездротовою мережею і передавати дані через неї;

- #include "esp_camera.h" - це бібліотека для роботи з камерою ESP32-Cam, яка дозволяє отримувати зображення з камери і використовувати його для візуальної навігації або розпізнавання об'єктів;

- <WiFi.h> - ця бібліотека також використовується для роботи з Wi-Fi, але для платформи ESP32 вона містить деякі додаткові функції та класи для роботи з мережею;

- "soc/soc.h" та "soc/rtc_cntl_reg.h" - ці бібліотеки використовуються для доступу до низькорівневих функцій ESP32, які можуть бути корисними для роботи з Wi-Fi та камерою на рівні апаратного забезпечення.

```
// Setup Access Point Credentials
```

```
const char* ssid1 = "ESP32-CAM Robot";
const char* password1 = "1234567890";
```

Цей фрагмент коду встановлює ім'я та пароль для точки доступу (Access Point, AP) на ESP32-Cam. AP - це бездротова мережа, яку створює ESP32-Cam, і до якої інші пристрої можуть підключатися для комунікації з ним. У цьому випадку ім'я мережі (SSID) встановлено як "ESP32-CAM Robot", а пароль - "1234567890".

```
extern volatile unsigned int  motor_speed;
extern void robot_stop();
extern void robot_setup();
extern uint8_t robo;
extern volatile unsigned long previous_time;
extern volatile unsigned long move_interval;
```

Цей фрагмент коду оголошує декілька змінних і функцій, які використовуються для керування рухом мобільного робота на базі ESP32-Cam [59]:

- extern volatile unsigned int motor_speed; - оголошення змінної motor_speed як беззнакової цілочисельної змінної, використовується для встановлення швидкості руху моторів робота;

- extern void robot_stop(); - оголошення функції robot_stop(), використовується для зупинки руху робота;

- extern void robot_setup(); - оголошення функції robot_setup(), використовується для ініціалізації параметрів та обладнання робота перед початком роботи;

- extern uint8_t robo; - оголошення без знакової цілочисельної змінної robo, використовується для ідентифікації робота або його стану;

- extern volatile unsigned long previous_time; - оголошення змінної previous_time як без знакового довгого цілочисельного типу, , використовується для визначення попереднього часу виконання деякої дії;

- extern volatile unsigned long move_interval; - оголошення змінної move_interval як без знакового довгого цілочисельного типу, яка використовується для визначення інтервалу часу між рухами робота;

```
#define CAMERA_MODEL_AI_THINKER
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
```

Фрагмент коду містить директиви препроцесора #define, які встановлюють значення для різних GPIO (General Purpose Input/Output) пінів на ESP32-Cam для керування камерою. Далі представлено їх пояснення:

- CAMERA_MODEL_AI_THINKER - ця директива вказує на використання камери AI Thinker ESP32-Cam.

- PWDN_GPIO_NUM, RESET_GPIO_NUM, XCLK_GPIO_NUM, SIOD_GPIO_NUM, SIOC_GPIO_NUM, Y9_GPIO_NUM, Y8_GPIO_NUM, Y7_GPIO_NUM, Y6_GPIO_NUM, Y5_GPIO_NUM, Y4_GPIO_NUM, Y3_GPIO_NUM, Y2_GPIO_NUM, VSYNC_GPIO_NUM, HREF_GPIO_NUM, PCLK_GPIO_NUM - ці директиви вказують на номера GPIO пінів ESP32-

Cam, які використовуються для різних функцій камери (наприклад, включення/вимкнення, скидання, клокування, передачі даних тощо).

Ці директиви допомагають у встановленні відповідності між GPIO пінами ESP32-Cam і їх функціями керування камерою, що є важливим для правильної роботи камери на мікроконтролері ESP32.

void startCameraServer();

Ця функція оголошується з прототипом void, тому вона призначена для початку роботи сервера камери. Це може бути частиною бібліотеки або модуля, який надає можливість транслювати відео або зображення з камери ESP32-Cam через мережу (наприклад, через Wi-Fi). Оскільки це лише оголошення функції, код її реалізації буде розташований у відповідному місці програми або бібліотеки. Така функція може включати ініціалізацію камери, підготовку до трансляції, створення веб-сервера для отримання запитів на відео та інше.

```
void setup()
{
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // prevent brownouts by silencing them

  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
```

Ця функція setup() виконує початкове налаштування пристрою ESP32-Cam. Далі представлено пояснення деяких дій, які вона виконує:

- WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); - ця інструкція встановлює регістр контролю реального часу (RTC) для керування броунівськими відключеннями (brownouts) на значення 0, що дозволяє уникнути можливих відключень через недостатню напругу.

- Serial.begin(115200); - ця функція ініціалізує інтерфейс UART для відладки та виводу даних на швидкості 115200 біт за секунду.

- Serial.setDebugOutput(true); - ця функція встановлює вивід відладки через інтерфейс UART, що дозволяє відслідковувати деякі дії або помилки, які виникають в процесі виконання програми.

- Serial.println(); - ця інструкція друкує порожній рядок в вивідному потоці UART, що дозволяє зробити розділення між початковими повідомленнями та даними, які виводяться на вивід.

```
camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
```

Цей фрагмент коду встановлює конфігурацію камери для ESP32-Cam. Далі представлено пояснення кожного поля в структурі camera_config_t:

- ledc_channel, ledc_timer - налаштування каналу та таймера для LED контролера. Вони використовуються для керування LED підсвіткою камери.

- pin_d0 до pin_d7 - GPIO піни для передачі даних від камери до ESP32.

- pin_xclk - GPIO пін для клокового сигналу камери.

- pin_pclk - GPIO пін для сигналу зсуву фази пікселя.

- pin_vsync - GPIO пін для синхронізації вертикальної синхронізації.

- pin_href - GPIO пін для сигналу горизонтальної синхронізації.

- pin_sscb_sda, pin_sscb_scl - GPIO піни для інтерфейсу камери I2C для зчитування конфігурації камери.

- pin_pwdn, pin_reset - GPIO піни для керування живленням та скиданням камери відповідно.

- xclk_freq_hz - частота клокування для камери (20 мегагерц).

- pixel_format - формат пікселів, у якому камера виводить зображення (JPEG).

Ці налаштування встановлюють параметри взаємодії ESP32-Cam з камерою, які необхідні для правильної роботи камери та зчитування зображення з неї.

```
//init with high specs to pre-allocate larger buffers
  if(psramFound()){
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
```

Фрагмент коду встановлює деякі параметри камери, такі як розмір кадру, якість JPEG та кількість буферів кадрів. Далі представлено їх пояснення:

- if(psramFound()) { ... } else { ... } - ця конструкція перевіряє наявність PSRAM (псевдо-статичної оперативної пам'яті) на пристрої. PSRAM може бути використана для зберігання великих обсягів даних, таких як зображення;

Якщо PSRAM знайдено, то встановлюються наступні параметри:

- config.frame_size = FRAMESIZE_QVGA; - розмір кадру QVGA (320x240 пікселів) ;

- config.jpeg_quality = 10; - якість JPEG 10 (низька якість, але менший розмір файлу) ;

- config.fb_count = 2; - кількість буферів кадрів 2 (два буфери для зображень) ;

Якщо PSRAM не знайдено, то встановлюються такі параметри:

- config.frame_size = FRAMESIZE_QVGA; - розмір кадру QVGA (320x240 пікселів) ;

- config.jpeg_quality = 12; - якість JPEG 12 (висока якість, але більший розмір файлу) ;

- config.fb_count = 1; - кількість буферів кадрів 1 (один буфер для зображень);

Ці параметри впливають на якість та розмір зображення, яке зберігається та відображається з камери. Вони можуть бути налаштовані відповідно до ваших потреб щодо якості та обсягу пам'яті.

```
// camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
```

Цей фрагмент коду ініціалізує камеру ESP32-Cam з використанням раніше встановленої конфігурації config. Далі представлено пояснення кожного кроку:

- esp_err_t err = esp_camera_init(&config); - ініціалізує камеру з використанням встановленої конфігурації config. Повертає помилку err, якщо ініціалізація не вдалася.

- if (err != ESP_OK) { ... } - перевіряє, чи ініціалізація камери була успішною. Якщо не вдалося ініціалізувати камеру, виводиться повідомлення про помилку та функція завершується.

Цей код слід викликати один раз на початку програми для ініціалізації камери перед будь-яким використанням. Це дозволяє налаштувати камеру відповідно до ваших потреб і переконатися, що вона готова до використання.

```
//drop down frame size for higher initial frame rate
  sensor_t * s = esp_camera_sensor_get();
  s->set_framesize(s, FRAMESIZE_QVGA);
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
```

Фрагмент коду встановлює розмір кадру камери на QVGA (320x240 пікселів) для підвищення початкової частоти кадрів. Далі представлено пояснення кожного кроку:

- sensor_t *s = esp_camera_sensor_get(); - отримує вказівник на структуру sensor_t, яка представляє сенсор камери ESP32-Cam;

- s->set_framesize(s, FRAMESIZE_QVGA); - встановлює розмір кадру QVGA за допомогою функції set_framesize сенсора камери. Це дозволяє зменшити розмір кадру для отримання більшої частоти кадрів;

- s->set_vflip(s, 1); - встановлює відображення по вертикалі. Це може бути корисним для виправлення зображення, яке перевернуте догори дном;

- s->set_hmirror(s, 1); - встановлює відображення по горизонталі. Це може бути корисним для виправлення зображення, яке відображено відзеркалено;

Ці налаштування дозволяють підвищити початкову частоту кадрів за рахунок зменшення розміру кадру та відображення по вертикалі та горизонталі для покращення зображення з камери.

```
WiFi.softAP(ssid1, password1);
  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
```

Фрагмент коду встановлює ESP32-Cam в режим точки доступу (Access Point, AP) з використанням вказаного імені мережі (SSID) та пароля. Після цього він отримує IP-адресу, яку можна використовувати для з'єднання з цією точкою доступу. Далі представлено пояснення кожного кроку:

- WiFi.softAP(ssid1, password1); - встановлює ESP32-Cam в режим точки доступу з вказаним іменем мережі (SSID) та паролем. Це дозволяє іншим пристроям підключатися до ESP32-Cam як до точки доступу;

- IPAddress myIP = WiFi.softAPIP(); - отримує IP-адресу точки доступу, до якої підключені інші пристрої. Ця адреса може бути використана для доступу до ESP32-Cam через мережу;

- Serial.print("AP IP address: "); - виводить текст "AP IP address: " на порт послідовного інтерфейсу (Serial) ;

- Serial.println(myIP); - виводить IP-адресу точки доступу на порт послідовного інтерфейсу (Serial), що дозволяє переглянути її у серійному моніторі Arduino IDE або іншому терміналі;

Ці кроки дозволяють налаштувати ESP32-Cam як точку доступу з вказаним іменем мережі та паролем, а також дозволяють отримати IP-адресу цієї точки доступу для подальшого використання.

```
startCameraServer();
```

Ця функція викликається для запуску сервера камери, який дозволяє отримувати відеопотік або зображення з камери ESP32-Cam через мережу. Після виклику цієї функції сервер камери розпочне прослуховування запитів на відеопотік або зображення та надсилання їх клієнтам через мережу. Цей виклик є важливим для взаємодії з камерою ESP32-Cam через мережу і дозволяє вам використовувати зображення або відеопотік з камери

```
ledcSetup(7, 5000, 8);
```

```
  ledcAttachPin(4, 7);  //pin4 is LED
  robot_setup();
```

Цей фрагмент коду встановлює параметри для LED керування за допомогою LEDC (LED Controller) на ESP32. Далі представлено пояснення кожного кроку:

- ledcSetup(7, 5000, 8); - ця функція налаштовує канал LEDC з номером 7 на частоту 5000 Гц та бітну роздільну здатність 8 біт. Це визначає, як LED буде відображати різні рівні яскравості на основі PWM (ШИМ) сигналу;

- ledcAttachPin(4, 7); - ця функція призначає пін 4 для використання з каналом 7 LEDC. Це означає, що пін 4 буде використовуватися для виводу PWM сигналу для керування LED;

- robot_setup(); - ця функція виконує початкове налаштування робота. Вона може бути викликана після налаштування LEDC для подальшого налаштування робота або інших функцій програми.

Цей фрагмент коду використовує LEDC для керування яскравістю світлодіода на піні 4 з використанням PWM сигналу.

```
for (int i=0;i<5;i++)
  {
   ledcWrite(7,10);  // flash led
   delay(50);
   ledcWrite(7,0);
   delay(50);
  }
  previous_time = millis();
}
```

Цей фрагмент коду використовується для миготіння світлодіодом з використанням LEDC на ESP32. Далі представлено пояснення кожного кроку циклу:

- for (int i=0;i<5;i++) { ... } - цей цикл виконується 5 разів для миготіння світлодіодом;

- ledcWrite(7,10); - ця функція встановлює яскравість світлодіода, приєднаного до каналу 7 LEDC, на рівень 10 (з 0 до 255). Це включає світлодіод;

- delay(50); - ця функція затримує виконання програми на 50 мілісекунд, що призводить до утримання світлодіода у включеному стані протягом 50 мілісекунд;

- ledcWrite(7,0); - ця функція встановлює яскравість світлодіода на рівень 0, що вимикає світлодіод;

- delay(50); - ця функція знову затримує виконання програми на 50 мілісекунд, що призводить до утримання світлодіода у вимкненому стані протягом 50 мілісекунд;

- previous_time = millis(); - ця інструкція оновлює попередній час, використовуваний у вашій програмі (можливо, для подальшого використання в інших частинах коду).

Цей код використовується для створення ефекту миготіння світлодіода за допомогою PWM сигналу LEDC на ESP32. Кожні 100 мілісекунд світлодіод буде перемикатися між увімкненим та вимкненим станом протягом 5 циклів.

```
void loop() {
  if(robo)
  {
    unsigned long currentMillis = millis();
    if (currentMillis - previous_time >= move_interval) {
      previous_time = currentMillis;
      robot_stop();
      char rsp[32];
      sprintf(rsp,"SPPED: %d",motor_speed);
      Serial.println("Stop");
      robo=0;
    }
```

```
  }
  delay(1);
  yield();
}
```

Фрагмент коду представляє цикл loop, який перевіряє змінну robo для виконання деяких дій у разі, якщо вона має значення відмінне від нуля. Далі представлено пояснення:

- if(robo) { ... } - перевіряє, чи змінна robo не дорівнює нулю. Це означає, що виконуються дії, коли робот рухається (змінна robo дорівнює одиниці);

- unsigned long currentMillis = millis(); - отримує поточний час в мілісекундах від початку виконання програми;

- if (currentMillis - previous_time >= move_interval) { ... } - перевіряє, чи пройшов час, необхідний для перемикання робота на нову позицію. Якщо так, виконуються наступні дії:

- previous_time = currentMillis; - оновлює попередній час для наступної ітерації;

- robot_stop(); - зупиняє рух робота;

- char rsp[32]; sprintf(rsp,"SPPED: %d",motor_speed); - створює рядок з швидкістю руху робота;

- Serial.println("Stop"); - виводить повідомлення про зупинку робота;

- robo=0; - встановлює значення robo на 0, щоб позначити, що робот зупинено;

- delay(1); та yield(); - ці функції допомагають у виконанні інших задач у мікроконтролері ESP32, таких як обробка мережевих запитів або інших переривань, що можуть виникнути у фоновому режимі.

Цей код дозволяє роботу зупинятися після досягнення певного інтервалу часу, що дозволяє реалізувати певну логіку руху або дій у вашій програмі.

Опишемо лістинг файла app_httpd.cpp, якій відповідає за керування та візуалізацію HMI інтерфейсу користувача. Спочатку підключимо бібліотеки:

```
#include "dl_lib_matrix3d.h"
#include <esp32-hal-ledc.h>
#include "esp_http_server.h"
#include "esp_timer.h"
#include "esp_camera.h"
#include "img_converters.h"
#include "Arduino.h"
```

Цей фрагмент коду містить директиви #include, які додають до вашої програми деякі бібліотеки та заголовні файли, необхідні для роботи з ESP32-Cam та іншими компонентами. Далі представлено їх пояснення:

- #include "dl_lib_matrix3d.h" - ця директива додає заголовний файл для роботи з матрицями 3D. Цей файл може використовуватися для обробки та аналізу даних у тривимірному просторі;

- #include <esp32-hal-ledc.h> - ця директива додає заголовний файл для роботи з LED контролером ESP32. Цей файл може використовуватися для керування світлодіодами та іншими пристроями, які підтримують PWM сигнали;

- #include "esp_http_server.h" - ця директива додає заголовний файл для роботи з веб-сервером ESP32. Цей файл може використовуватися для створення веб-сервера на ESP32 для обробки HTTP запитів;

- #include "esp_timer.h" - ця директива додає заголовний файл для роботи з таймерами ESP32. Цей файл може використовуватися для створення таймерів та вимірювання часу у вашій програмі;

- #include "esp_camera.h" - ця директива додає заголовний файл для роботи з камерою ESP32-Cam. Цей файл може використовуватися для керування камерою та отримання зображень з неї;

- #include "img_converters.h" - ця директива додає заголовний файл для конвертації зображень. Цей файл може використовуватися для конвертації форматів зображень;

- #include "Arduino.h" - ця директива додає заголовний файл для роботи з функціями Arduino. Цей файл містить основні функції для програмування мікроконтролерів ESP32 в середовищі Arduino IDE.

// TB6612FNG H-Bridge Connections (both PWM inputs driven by GPIO 12)

```
#define MTR_PWM   12
#define LEFT_M0    15
#define LEFT_M1    14
#define RIGHT_M0   13
#define RIGHT_M1   2
```

Цей фрагмент коду визначає піни ESP32, які використовуються для керування мостом H TB6612FNG. Далі представлено пояснення кожного піна:

- #define MTR_PWM 12 - пін, який використовується для керування швидкістю обертання мотора за допомогою сигналу ШІМ. Вказано GPIO 12;

- #define LEFT_M0 15 - пін, що визначає напрямок обертання лівого мотора, перший вхід (M0) моста H. Вказано GPIO 15;

- #define LEFT_M1 14 - пін, що визначає напрямок обертання лівого мотора, другий вхід (M1) моста H. Вказано GPIO 14;

- #define RIGHT_M0 13 - пін, що визначає напрямок обертання правого мотора, перший вхід (M0) моста H. Вказано GPIO 13;

- #define RIGHT_M1 2 - пін, що визначає напрямок обертання правого мотора, другий вхід (M1) моста H. Вказано GPIO 2;

```
// Define Speed variables
int speed = 255;
int noStop = 0;
```

Цей фрагмент коду визначає змінні для швидкості руху двигуна та стану "noStop" для контролю руху робота. Далі представлено пояснення кожної змінної:

- int speed = 255; - ця змінна визначає швидкість руху двигуна. Значення 255 вказує на максимальну швидкість, оскільки вона виражається в 8-бітному беззнаковому значенні (255 - максимальне значення для 8 біт).

- int noStop = 0; - ця змінна використовується для вказівки на те, що робот не повинен зупинятися. Зазвичай це може бути значенням, що позначає відсутність зупинки або рух без зупинки.

Ці змінні можуть бути використані для керування швидкістю руху робота та визначення стану руху (зупинено або не зупинено) у вашій програмі або проекті.

```
//Setting Motor PWM properties
const int freq = 2000;
const int motorPWMChannnel = 8;
const int lresolution = 8;
volatile unsigned int  motor_speed   = 200;
volatile unsigned long previous_time = 0;
volatile unsigned long move_interval = 250;
```

Фрагмент коду визначає властивості PWM для керування швидкістю обертання двигуна. Далі представлено пояснення кожної змінної:

- const int freq = 2000; - ця змінна визначає частоту PWM у герцах. У цьому випадку частота становить 2000 Гц;

- const int motorPWMChannnel = 8; - ця змінна вказує на номер каналу PWM для керування двигуном. У цьому випадку використовується канал 8;

- const int lresolution = 8; - ця змінна визначає роздільну здатність (кількість біт) для PWM. У цьому випадку використовується 8 біт, що дозволяє встановити швидкість в діапазоні від 0 до 255;

- volatile unsigned int motor_speed = 200; - ця змінна визначає швидкість обертання двигуна. Вона є відмінною від speed і змінюється за допомогою інших частин програми;

- volatile unsigned long previous_time = 0; - ця змінна використовується для зберігання попереднього часу, що дозволяє визначити, коли потрібно змінити швидкість обертання двигуна;

- volatile unsigned long move_interval = 250; - ця змінна визначає інтервал часу між змінами швидкості обертання двигуна. У цьому випадку інтервал становить 250 мілісекунд.

Ці змінні дозволяють налаштовувати параметри PWM для керування швидкістю обертання двигуна у вашій програмі або проекті.

```
// Placeholder for functions
void robot_setup();
void robot_stop();
void robot_fwd();
void robot_back();
void robot_left();
void robot_right();
uint8_t robo = 0;
```

Цей фрагмент коду містить оголошення функцій та змінної для керування роботом. Далі представлено пояснення кожного елементу:

- void robot_setup(); - ця функція встановлює параметри та налаштування для робота. Вона повинна бути визначена додатково для конкретної реалізації;

- void robot_stop(); - ця функція зупиняє рух робота. Також потрібно визначити додатково для конкретної реалізації;

- void robot_fwd();, void robot_back();, void robot_left();, void robot_right(); - ці функції відповідають за рух робота вперед, назад, вліво та вправо відповідно. Їх також потрібно визначити окремо;

- uint8_t robo = 0; - ця змінна використовується для вказівки на поточний стан робота. Наприклад, вона може мати значення 1, якщо робот рухається, або 0, якщо він зупинений.

Ці оголошення використовуються для створення інтерфейсу керування роботом, але фактичні функції повинні бути реалізовані відповідно до потреб вашого проекту.

```
typedef struct {
  httpd_req_t *req;
  size_t len;
} jpg_chunking_t;
```

Фрагмент коду визначає новий тип даних jpg_chunking_t, який є структурою з двома полями:

- httpd_req_t *req; - це вказівник на структуру httpd_req_t, яка представляє HTTP запит. Це дозволяє зберігати посилання на запит, який пов'язаний з даною порцією JPEG даних.

- size_t len; - це поле, яке вказує на довжину (кількість байт) даних, що будуть відправлені в даній порції.

Така структура може бути використана для передачі порцій JPEG даних через HTTP веб-сервер частинами, що може бути корисно для передачі великих файлів по мережі з обмеженими обсягами пам'яті або пропускної здатності.

```
#define PART_BOUNDARY "123456789000000000000987654321"
static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
```

Фрагмент коду визначає деякі константи для побудови потокового відтворення мультимедійного вмісту через HTTP, зокрема, для передачі зображень у форматі JPEG. Далі представлено їх пояснення:

- #define PART_BOUNDARY "123456789000000000000987654321" - визначає рядок, який використовується як границя між частинами даних в потоковому відтворенні. Це допомагає відокремити різні частини даних;

- static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY; - визначає тип контенту для потокового відтворення (multipart/x-mixed-replace) з вказанням границі між частинами даних;

- static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n"; - визначає рядок, який вказує на початок нової частини даних у потоці. Він складається з послідовності переносу рядка \r\n, знаку мінуса -- та границі частини PART_BOUNDARY;

- static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n"; - визначає формат частини даних у потоці. В цьому випадку вказується, що дані будуть у форматі JPEG (Content-Type: image/jpeg), а також передбачається вказання довжини частини даних у полі Content-Length.

Ці константи використовуються для побудови потокового відтворення даних у форматі JPEG через HTTP, де зображення передаються як окремі частини з вказанням їхньої довжини та типу.

httpd_handle_t stream_httpd = NULL;

httpd_handle_t camera_httpd = NULL;

Цей фрагмент коду оголошує дві змінні типу httpd_handle_t для керування HTTP серверами:

- httpd_handle_t stream_httpd = NULL; - ця змінна використовується для зберігання дескриптора (handle) HTTP сервера, який обробляє потокове відтворення даних (наприклад, відео або зображення).

- httpd_handle_t camera_httpd = NULL; - ця змінна використовується для зберігання дескриптора HTTP сервера, який обробляє запити для камери (наприклад, налаштування камери або отримання зображень).

Обидва сервери (stream_httpd і camera_httpd) можуть бути створені та налаштовані окремо для різних потреб вашого додатку. Наприклад, stream_httpd може бути використаний для відтворення відео з камери, а camera_httpd - для керування налаштуваннями камери або отримання зображень у форматі JPEG.

```
static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len) {
  jpg_chunking_t *j = (jpg_chunking_t *)arg;
  if (!index) {
    j->len = 0;
  }
  if (httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK) {
    return 0;
  }
  j->len += len;
  return len;
}
```

Функція jpg_encode_stream призначена для потокового кодування та передачі даних у форматі JPEG через HTTP. Далі представлено пояснення кожного рядка функції:

- static size_t jpg_encode_stream(void * arg, size_t index, const void* data, size_t len) { - визначення функції jpg_encode_stream з параметрами arg (вказівник на структуру jpg_chunking_t), index (індекс частини даних), data (вказівник на дані, які треба відправити), та len (довжина даних у байтах).

- jpg_chunking_t *j = (jpg_chunking_t *)arg; - приведення параметра arg до типу jpg_chunking_t * для доступу до структури jpg_chunking_t, яка містить вказівник на HTTP запит та довжину даних.

- if (!index) { j->len = 0; } - якщо це перша частина даних, скинути лічильник довжини даних до нуля.

- if (httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK) { return 0; } - відправлення частини даних через HTTP відповідь. Якщо відправлення не вдалося, функція повертає 0.

- j->len += len; - додавання довжини поточної частини даних до загальної довжини.

- return len; - повернення довжини поточної частини даних, щоб показати успішність відправлення.

Ця функція може бути використана для передачі даних у форматі JPEG через HTTP відповідь в потоковому режимі, дозволяючи побудувати потокове відтворення або передавати великі файли зображень по мережі.

```
static esp_err_t capture_handler(httpd_req_t *req) {
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  int64_t fr_start = esp_timer_get_time();

  fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }
```

Функція capture_handler відповідає за захоплення зображення з камери і відправку його через HTTP запит. Далі представлено пояснення кожного рядка функції:

- static esp_err_t capture_handler(httpd_req_t *req) { - визначення функції capture_handler з параметром req, який є вказівником на структуру HTTP запиту.

- camera_fb_t * fb = NULL; - оголошення вказівника fb на структуру camera_fb_t, яка представляє кадр, отриманий з камери.

- esp_err_t res = ESP_OK; - ініціалізація змінної res значенням ESP_OK, що вказує на успішне виконання операції.

- int64_t fr_start = esp_timer_get_time(); - отримання часу початку захоплення кадру з камери за допомогою функції esp_timer_get_time.

- fb = esp_camera_fb_get(); - захоплення кадру з камери і збереження його в структурі fb.

- if (!fb) { - перевірка на успішність захоплення кадру.

- Serial.println("Camera capture failed"); - виведення повідомлення у випадку невдалого захоплення кадру.

- httpd_resp_send_500(req); - відправлення статусу 500 (Internal Server Error) у відповідь на HTTP запит.

- return ESP_FAIL; - повернення коду помилки ESP_FAIL, що вказує на невдачу операції захоплення кадру.

Ця функція використовується для захоплення кадру з камери і обробки ситуації, коли захоплення кадру не вдалося.

httpd_resp_set_type(req, "image/jpeg");

httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg");

Ці дві функції встановлюють тип контенту та заголовок HTTP відповіді, які будуть використані для передачі зображення у форматі JPEG через HTTP.

- httpd_resp_set_type(req, "image/jpeg"); - встановлює тип контенту відповіді як "image/jpeg", що вказує на те, що дані, що відправляються, є зображенням у форматі JPEG.

- httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg"); - встановлює заголовок "Content-Disposition" зі значенням "inline; filename=capture.jpg", що вказує браузеру відобразити

зображення прямо у вікні (inline) і вказує ім'я файлу "capture.jpg" для збереження на локальному пристрої.

Ці дві функції допомагають коректно відображати та обробляти зображення у форматі JPEG, що відправляється через HTTP.

```
size_t out_len, out_width, out_height;
  uint8_t * out_buf;
  bool s;
  {
    size_t fb_len = 0;
    if (fb->format == PIXFORMAT_JPEG) {
      fb_len = fb->len;
      res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
    } else {
      jpg_chunking_t jchunk = {req, 0};
      res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk) ? ESP_OK : ESP_FAIL;
      httpd_resp_send_chunk(req, NULL, 0);
      fb_len = jchunk.len;
    }
    esp_camera_fb_return(fb);
    int64_t fr_end = esp_timer_get_time();
    Serial.printf("JPG: %uB %ums\n", (uint32_t)(fb_len), (uint32_t)((fr_end - fr_start) / 1000));
    return res;
  }
```

Цей фрагмент коду відповідає за відправку зображення через HTTP відповідь. Далі представлено пояснення кожного рядка:

- size_t out_len, out_width, out_height; - оголошення змінних для зберігання довжини, ширини та висоти вихідного зображення.

- uint8_t * out_buf; - вказівник на буфер для зберігання вихідного зображення.

- bool s; - змінна, що вказує на статус операції.

- {} - блок коду, який обмежує область видимості деяких змінних.

- size_t fb_len = 0; - ініціалізація довжини зображення.

- if (fb->format == PIXFORMAT_JPEG) { ... } else { ... } - перевірка формату зображення: якщо формат JPEG, то зображення відправляється цілком; в іншому випадку воно кодується у формат JPEG та відправляється частинами.

- esp_camera_fb_return(fb); - повернення буфера кадру в камеру після використання.

- Serial.printf("JPG: %uB %ums\n", (uint32_t)(fb_len), (uint32_t)((fr_end - fr_start) / 1000)); - виведення інформації про відправлене зображення: його розмір у байтах та час відправлення у мілісекундах.

- return res; - повернення результату відправлення.

Цей код використовується для передачі зображення через HTTP відповідь, оброблюючи різні формати зображення та відправляючи їх відповідно.

```
dl_matrix3du_t *image_matrix = dl_matrix3du_alloc(1, fb->width, fb->height, 3);
  if (!image_matrix) {
    esp_camera_fb_return(fb);
    Serial.println("dl_matrix3du_alloc failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }
```

Цей фрагмент коду використовується для створення 3D матриці для зображення за допомогою функції dl_matrix3du_alloc. Далі представлено пояснення кожного рядка:

- dl_matrix3du_t *image_matrix = dl_matrix3du_alloc(1, fb->width, fb->height, 3); - створення 3D матриці image_matrix для зображення з одним каналом (1), шириною fb->width, висотою fb->height та 3 каналами кольору (RGB).

- if (!image_matrix) { ... } - перевірка, чи вдалося виділити пам'ять під матрицю. Якщо виділення пам'яті не вдалося, відбувається звільнення буфера кадру, виведення помилки та відправлення статусу 500 у відповідь на HTTP запит.

Цей код важливий для обробки зображення та його подальшої обробки або використання у додатку.

```
out_buf = image_matrix->item;
  out_len = fb->width * fb->height * 3;
  out_width = fb->width;
  out_height = fb->height;
  s = fmt2rgb888(fb->buf, fb->len, fb->format, out_buf);
  esp_camera_fb_return(fb);
  if (!s) {
    dl_matrix3du_free(image_matrix);
    Serial.println("to rgb888 failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
  }
```

У цьому фрагменті коду здійснюється конвертація формату зображення у RGB888 та визначення розмірів зображення для подальшої обробки. Далі представлено пояснення кожного рядка:

- out_buf = image_matrix->item; - отримання вказівника на початок даних в матриці image_matrix.

- out_len = fb->width * fb->height * 3; - визначення загальної довжини даних у байтах для RGB888 формату (ширина * висота * 3 байти на кожен піксель).

- out_width = fb->width; - збереження ширини зображення.

- out_height = fb->height; - збереження висоти зображення.

- s = fmt2rgb888(fb->buf, fb->len, fb->format, out_buf); - конвертація формату зображення у RGB888 за допомогою функції fmt2rgb888. Результат операції зберігається у змінній s.

- esp_camera_fb_return(fb); - звільнення буфера кадру після завершення обробки.

- if (!s) { ... } - перевірка успішності конвертації. У разі невдачі відбувається звільнення пам'яті, виведення помилки та відправлення статусу 500 у відповідь на HTTP запит.

Цей фрагмент коду важливий для підготовки зображення у форматі RGB888 для подальшої обробки або відображення.

```
jpg_chunking_t jchunk = {req, 0};
  s = fmt2jpg_cb(out_buf, out_len, out_width, out_height, PIXFORMAT_RGB888, 90, jpg_encode_stream, &jchunk);
  dl_matrix3du_free(image_matrix);
  if (!s) {
    Serial.println("JPEG compression failed");
    return ESP_FAIL;
  }
  int64_t fr_end = esp_timer_get_time();
  return res;
}
```

У цьому фрагменті коду здійснюється стиснення зображення у формат JPEG за допомогою функції fmt2jpg_cb. Далі представлено пояснення кожного рядка:

- jpg_chunking_t jchunk = {req, 0}; - створення структури jpg_chunking_t з параметрами для подальшого використання у функції jpg_encode_stream.

- s = fmt2jpg_cb(out_buf, out_len, out_width, out_height, PIXFORMAT_RGB888, 90, jpg_encode_stream, &jchunk); - стиснення

зображення у формат JPEG з RGB888 відповідно до вказаних параметрів (якість стиснення 90) та використання функції jpg_encode_stream для передачі даних через HTTP.

- dl_matrix3du_free(image_matrix); - звільнення пам'яті, яку займала матриця зображення.

- if (!s) { ... } - перевірка успішності стиснення. У разі невдачі виводиться повідомлення про помилку та функція повертає ESP_FAIL.

- int64_t fr_end = esp_timer_get_time(); - отримання часу завершення стиснення.

- return res; - повернення результату відправлення (успішно чи ні) для подальшого використання в програмі.

Цей фрагмент коду використовується для стиснення зображення у формат JPEG та передачі його через HTTP відповідь у вигляді потоку даних.

```
static esp_err_t stream_handler(httpd_req_t *req) {
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];
  dl_matrix3du_t *image_matrix = NULL;
  static int64_t last_frame = 0;
  if (!last_frame) {
    last_frame = esp_timer_get_time();
  }
```

Цей фрагмент коду відповідає за обробку HTTP запиту для потокової передачі зображення через HTTP. Далі представлено пояснення кожного рядка:

- camera_fb_t * fb = NULL; - оголошення вказівника на структуру camera_fb_t, яка представляє кадр, отриманий з камери.

- esp_err_t res = ESP_OK; - ініціалізація змінної res значенням ESP_OK, що вказує на успішне виконання операції.

- size_t _jpg_buf_len = 0; - ініціалізація змінної _jpg_buf_len нульовим значенням для зберігання довжини стиснутого зображення у форматі JPEG.

- uint8_t * _jpg_buf = NULL; - оголошення вказівника _jpg_buf для зберігання стиснутого зображення у форматі JPEG.

- char * part_buf[64]; - оголошення масиву символів для зберігання частини HTTP відповіді.

- dl_matrix3du_t *image_matrix = NULL; - оголошення вказівника на 3D матрицю для обробки зображення.

- static int64_t last_frame = 0; - оголошення статичної змінної last_frame, яка зберігає час останнього кадру для обробки.

- if (!last_frame) { ... } - перевірка, чи вже був отриманий перший кадр. Якщо ні, то зберігається час отримання першого кадру.

Цей код використовується для обробки HTTP запиту для потокової передачі зображення через HTTP.

```
res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if (res != ESP_OK) {
    return res;
  }
if (res == ESP_OK) {
      size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
      res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
```

Цей фрагмент коду використовується для відправки частини HTTP відповіді, що містить заголовок для частини зображення у форматі JPEG. Далі представлено пояснення кожного рядка:

- if (res == ESP_OK) { ... } - перевірка, чи попередні операції виконалися успішно.

- size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len); - створення частини HTTP відповіді з заголовком, який містить довжину стиснутого зображення у форматі JPEG.

- res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen); - відправлення частини HTTP відповіді з заголовком до клієнта.

Цей фрагмент коду допомагає у відправці частин зображення у форматі JPEG через HTTP відповідь.

```
// Look at values within URL to determine function
  if (!strcmp(variable, "framesize"))
  {
    Serial.println("framesize");
    if (s->pixformat == PIXFORMAT_JPEG) res = s->set_framesize(s, (framesize_t)val);
  }
  else if (!strcmp(variable, "quality"))
  {
    Serial.println("quality");
    res = s->set_quality(s, val);
  }
  else if (!strcmp(variable, "flash"))
  {
    ledcWrite(7, val);
  }
  else if (!strcmp(variable, "flashoff"))
  {
    ledcWrite(7, val);
  }
  else if (!strcmp(variable, "speed"))
  {
    if      (val > 255) val = 255;
```

```
  else if (val <   0) val = 0;
  speed = val;
  ledcWrite(8, speed);
 }
```

Цей фрагмент коду використовується для обробки параметрів, які передаються через URL, для керування налаштуваннями камери або іншими функціями програми. Далі представлено пояснення кожного рядка:

- if (!strcmp(variable, "framesize")) { ... } - перевіряє, чи параметр в URL є "framesize". Якщо так, то встановлює новий розмір кадру для камери.

- else if (!strcmp(variable, "quality")) { ... } - перевіряє, чи параметр в URL є "quality". Якщо так, то встановлює нову якість стиснення зображення.

- else if (!strcmp(variable, "flash")) { ... } - перевіряє, чи параметр в URL є "flash". Якщо так, то вмикає світлодіодний індикатор з вказаною яскравістю.

- else if (!strcmp(variable, "flashoff")) { ... } - перевіряє, чи параметр в URL є "flashoff". Якщо так, то вимикає світлодіодний індикатор.

- else if (!strcmp(variable, "speed")) { ... } - перевіряє, чи параметр в URL є "speed". Якщо так, то встановлює нову швидкість руху (speed) для рухомих частин.

Цей фрагмент коду дозволяє вам динамічно змінювати налаштування камери та інші параметри через HTTP запити, що дозволяє дистанційно керувати програмою.

```
static esp_err_t status_handler(httpd_req_t *req) {
  static char json_response[1024];

  sensor_t * s = esp_camera_sensor_get();
  char * p = json_response;
  *p++ = '{';

  p += sprintf(p, "\"framesize\":%u,", s->status.framesize);
  p += sprintf(p, "\"quality\":%u,", s->status.quality);
```

```
  *p++ = '}';
  *p++ = 0;
  httpd_resp_set_type(req, "application/json");
  httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
  return httpd_resp_send(req, json_response, strlen(json_response));
}
```

Цей фрагмент коду відповідає за обробку HTTP запиту для отримання статусу камери у форматі JSON. Далі представлено пояснення кожного рядка:

- static char json_response[1024]; - статичний масив для зберігання відповіді у форматі JSON.

- sensor_t * s = esp_camera_sensor_get(); - отримання вказівника на структуру sensor_t, яка містить інформацію про стан камери.

- char * p = json_response; - вказівник p встановлюється на початок масиву json_response.

- *p++ = '{'; - додає відкриваючу фігурну дужку до відповіді у форматі JSON.

- p += sprintf(p, "\"framesize\":%u,", s->status.framesize); - додає рядок у форматі "framesize":%u, до відповіді, де %u - це значення розміру кадру.

- p += sprintf(p, "\"quality\":%u,", s->status.quality); - додає рядок у форматі "quality":%u, до відповіді, де %u - це значення якості стиснення зображення.

- *p++ = '}'; - додає закриваючу фігурну дужку до відповіді у форматі JSON.

- *p++ = 0; - додає завершальний нуль-символ до відповіді у форматі JSON.

- httpd_resp_set_type(req, "application/json"); - встановлює тип відповіді як "application/json".

- httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*"); - встановлює заголовок для дозволу доступу з будь-якого джерела ("*").

- return httpd_resp_send(req, json_response, strlen(json_response)); - відправляє відповідь у форматі JSON до клієнта.

Цей фрагмент коду дозволяє отримувати статус камери у форматі JSON через HTTP запит та передавати цю інформацію на віддалений запит.

```
static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>ESP32 CAM Robot</title>
    <style>
  body{font-family:Arial,Helvetica,sans-serif,…….}
</style>
```

Цей фрагмент коду містить HTML-код для створення сторінки інтерфейсу користувача для керування мобільним роботом на ESP32-CAM через веб-інтерфейс. Він включає в себе метадані сторінки, такі як кодування символів та метатеги для налаштування масштабування, а також основний стиль для тексту.

Повний текст програми керування наведено у додатку Й

2.4 Налаштування та прошивка плати керування

Налаштування та прошивка плати керування ESP32-Cam в середовищі Arduino IDE є ключовим етапом у розробці програмного забезпечення для мобільного робота. Arduino IDE є популярним інтегрованим середовищем розробки (IDE) для мікроконтролерів Arduino, але також підтримує ESP32, що робить його зручним інструментом для роботи з ESP32-Cam. Перед початком роботи з ESP32-Cam у Arduino IDE необхідно налаштувати

середовище для роботи з цим мікроконтролером. Це включає встановлення підтримки ESP32 у Arduino IDE, вибір потрібної плати (наприклад, ESP32 Dev Module), налаштування параметрів комунікації (наприклад, швидкості передачі даних) і вибір правильного порту для підключення ESP32-Cam до комп'ютера.

Після успішного налаштування середовища ви можете розпочати прошивку плати керування ESP32-Cam. Це включає написання програмного коду у Arduino IDE, який керуватиме роботом, включаючи керування рухом, роботу камери, обробку даних та взаємодію з іншими пристроями. Для роботи з камерою ESP32-Cam можна використовувати спеціалізовані бібліотеки, які дозволяють зчитувати зображення та відео з камери, а також виконувати їхню обробку та передачу через мережу або інші канали зв'язку.

Налаштування та прошивка плати керування ESP32-Cam у середовищі Arduino IDE дозволяє створювати потужні програмні рішення для мобільних роботів з використанням функціоналу ESP32 та його камери. У цьому підрозділі ми детально розглянемо процес налаштування та програмування ESP32-Cam у Arduino IDE, щоб забезпечити оптимальне керування вашим мобільним роботом.

Метод налаштування модуля ESP32-Cam має наступну послідовність [60]:

- встановлення Arduino IDE із офіційно сайту: https://www.arduino.cc/en/Main/Software

- потрібно інтегрувати підтримку нашого модуля ESP32. Йдемо до налаштувань (File – Preferences) і в нижній частині вікна налаштувань в Additional Boards Manager URLs додаємо рядок:

https://dl.espressif.com/dl/package_esp32_index.json

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Отриманий результат повинен виглядати як представлено на рисунку 2.2

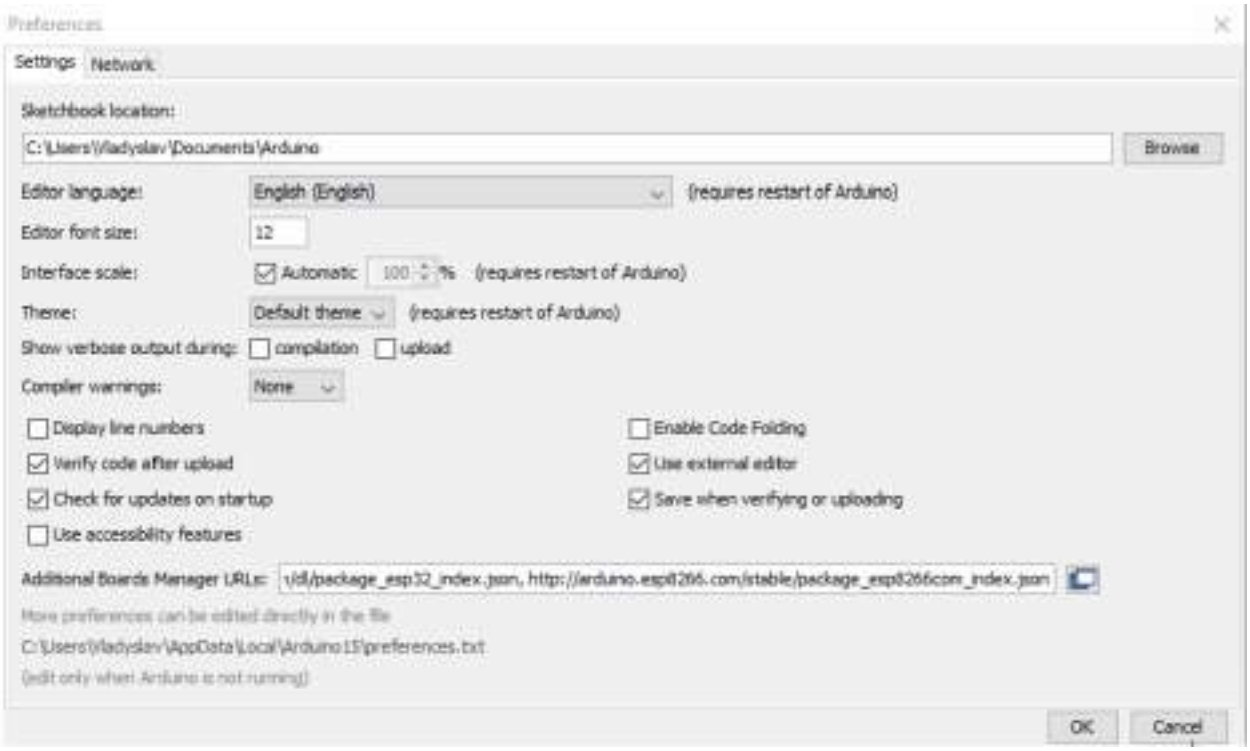


Рисунок 2.2 - Налаштування підтримки модулів ESP32-Cam в середовище Arduino IDE

- переходимо до Boards Manager: Tools – Board – Boards Manager для встановлення пакету ESP32-Cam як показано рисунку 2.3

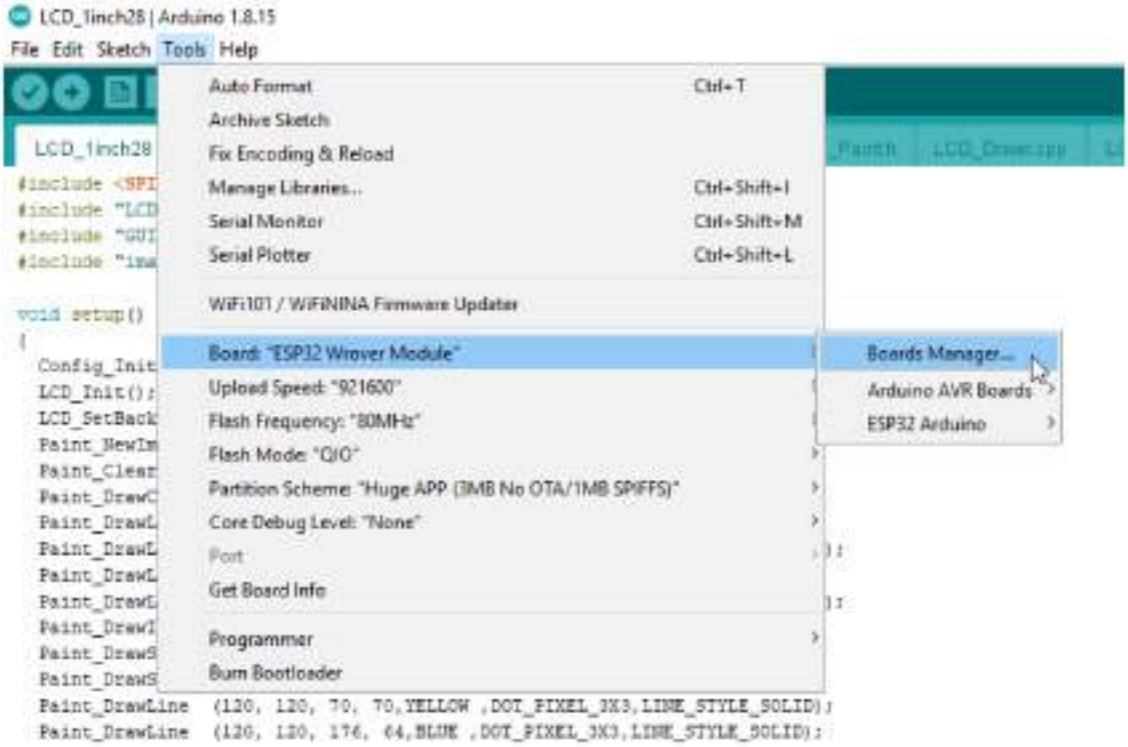


Рисунок 2.3 - Виклик вікна Boards Managerв середовище Arduino IDE

У вікні Boards Manager у рядку пошуку Type введіть "esp32" і проводимо Install пакету ESP32 by Espressif Systems, як показано на рисунку 2.4.

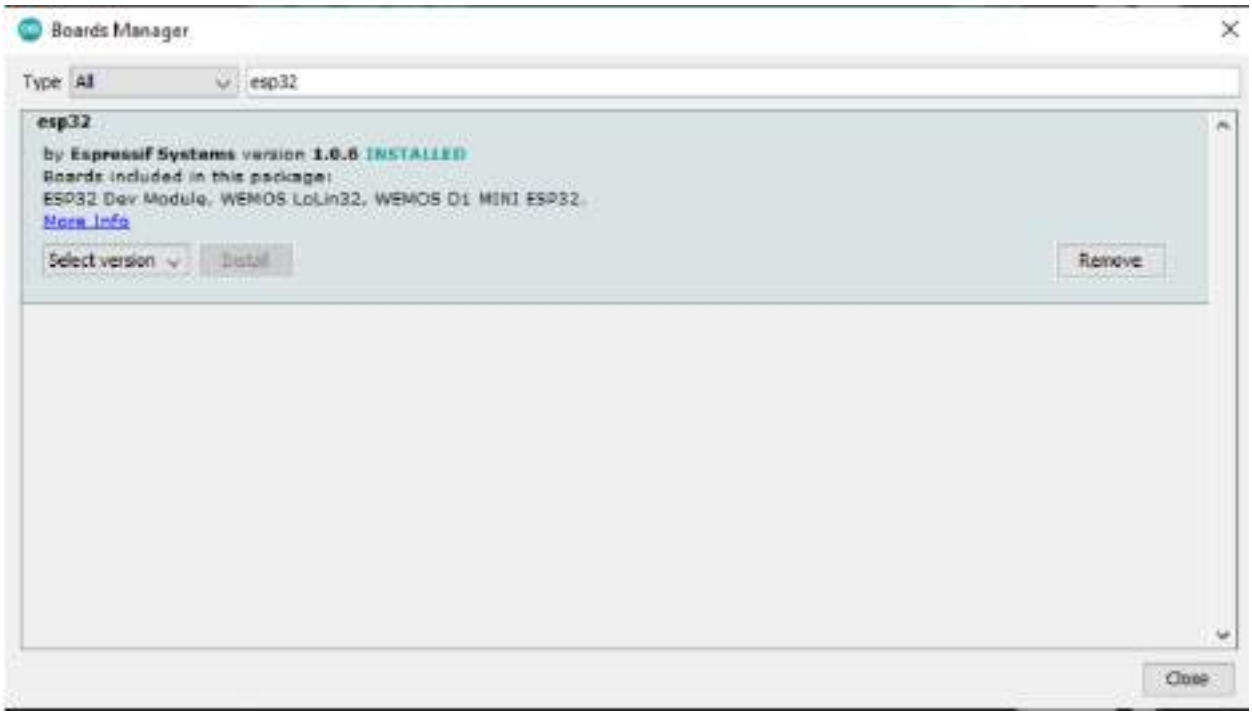


Рисунок 2.4 - Вікно завантаження та встановлення пакету ESP32 by Espressif Systems

Для перевірки працездатності ESP32-CAM та правильності підключення USB-UART необхідно виконати такі кроки.

Крок 1. Відкриваємо приклад Arduino IDE для роботи з камерою. Для цього слідуємо в File - Examples - ESP32 - Camera - CameraWebServer. Приклад шляху представлено рисунку 2.5.

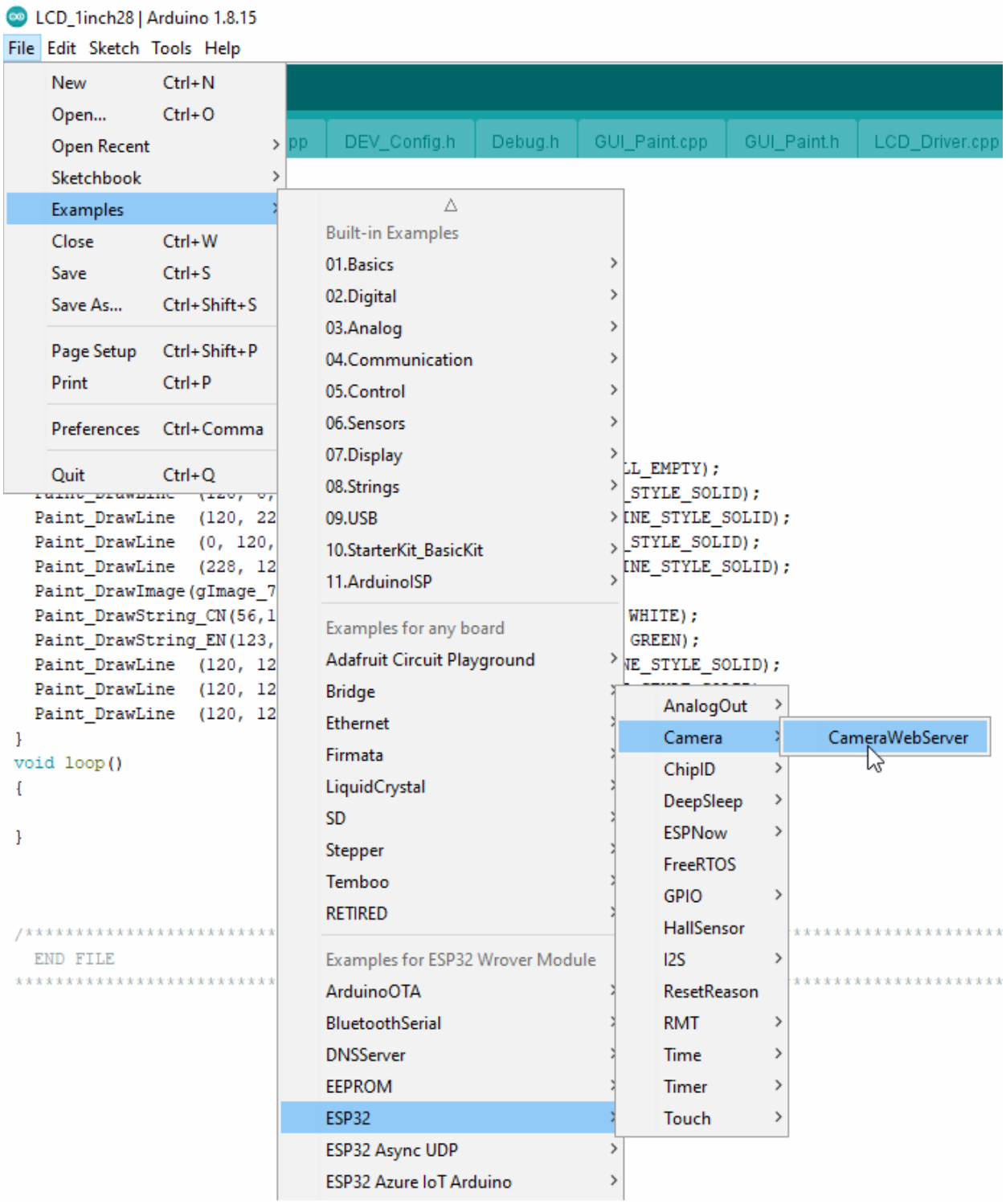


Рисунок 2.5 - Завантаження тестового скетчу для ESP32-CAM

Крок 2. Проводимо налаштування модуля камери для успішної роботи, внаслідок чого необхідно розкоментувати наступний рядок і закоментувати всі інші. У файлі CameraWebServer розкоментувати:

#define CAMERA_MODEL_AI_THINKER // Has PSRAM

Крок 3. Необхідно вказати SSID і password Wi-Fi мережі в якій працюватиме ESP32-CAM, для цього необхідно внести дані в рядках:

const char * ssid = "**********";

const char* password = "**********";

* нагадування SSID – назва мережі; password – пароль підключення.

Крок 4. У меню Tools потрібно задати актуальні параметри для прошивки плати ESP32-CAM. Для цього необхідно вибрати плату, номер COM порту для прошивки. Приклад параметрів підключення наведено рисунку 2.6.

Крок 5. Прошиваємо плату ESP32-CAM, для цього натискаємо кнопку UPLOAD (не забувши попередньо підтягнути пін GPIO0 до землі, якщо використовується для прошивки USB-UART перехідник на базі PL2303). Як показано рисунок 2.6.

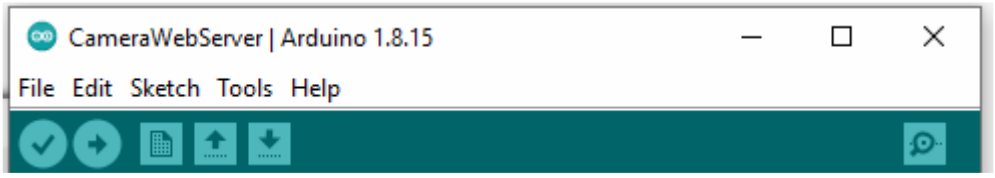


Рисунок 2.6 - Прошивки плати ESP32-CAM

Після завершення прошивки ESP32-CAM у вікні Arduino IDE видається наступний запис:

Leaving...

Hard resetting via RTS pin...

Що означає, необхідно перезавантаження ESP32-CAM і при цьому необхідно відключити підтяжку GPIO0.

Крок 6. Проведемо перевірку правильності прошивки та підключення до Wi-Fi мережі. Для цього заходимо до Tools-Serial monitor, як показано на рисунок 2.7.

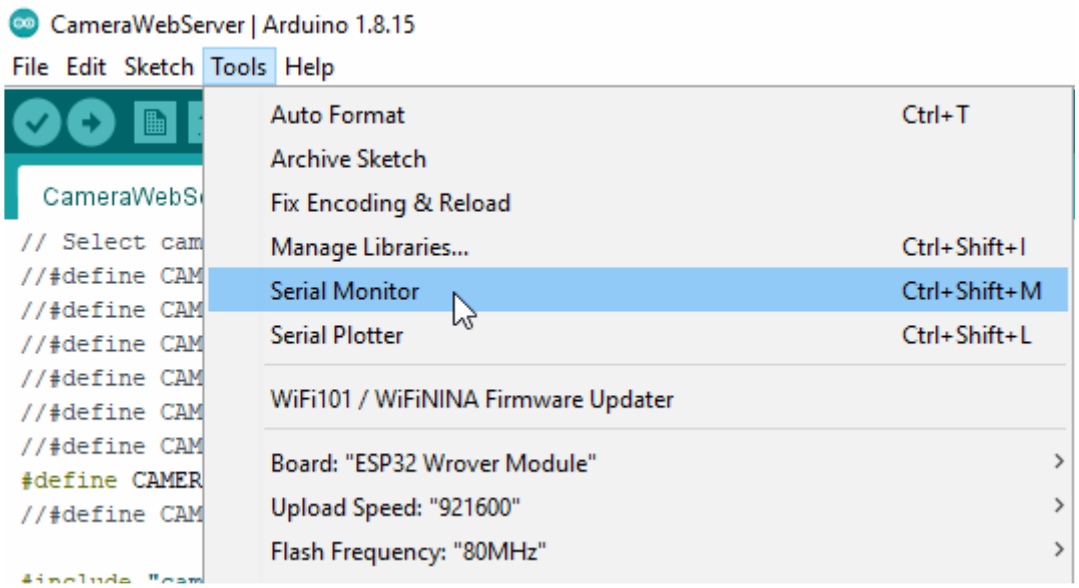


Рисунок 2.7 - Шлях для вибору Serial Monitor

При успішному з'єднанні з поточною Wi-Fi мережею в моніторі порту буде наступний запис, який представлений на рисунку 2.8

```
.....
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.43.1' to connect
```

Рисунок 2.8 - Вікно монітора порту

Для перевірки працездатності ESP32-CAM необхідно зайти на ПК або на мобільному телефоні, підключеному до цієї Wi-Fi мережі за адресою http://192.168.43.1

Крок 7. При переході на адресу присвоєну модулю ESP32-CAM вашою Wi-Fi мережею, стане доступний інтерфейс налаштування та запуску трансляцій, загальний вигляд якого представлений на рисунку 2.9.

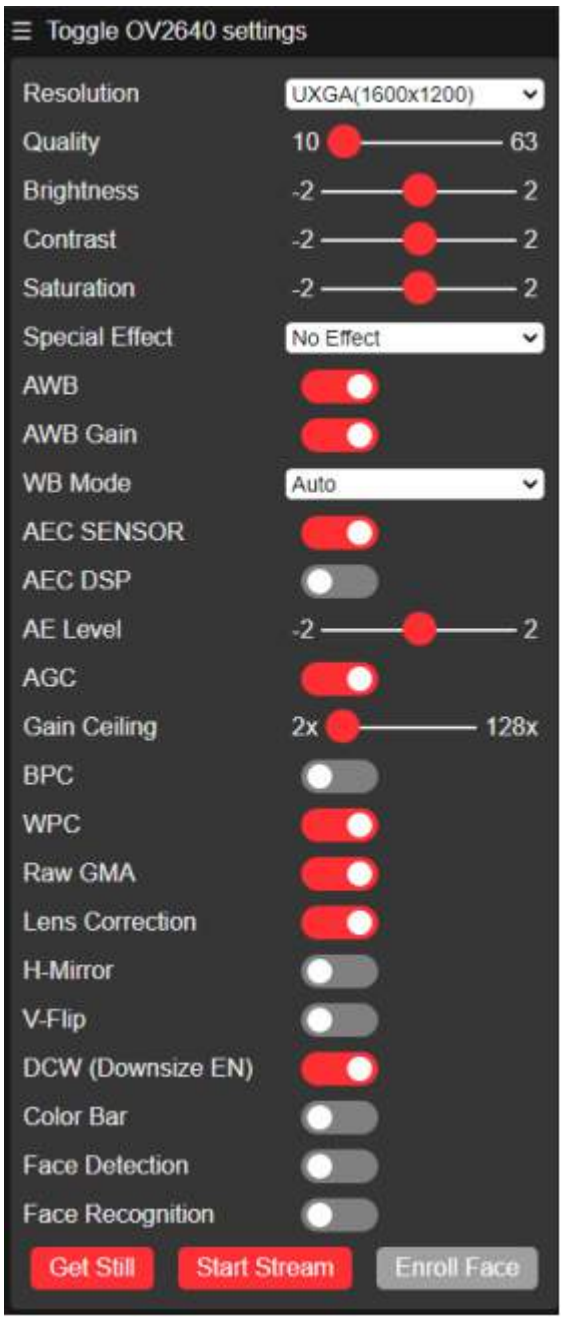


Рисунок 2.9 - Інтерфейс налаштування трансляцій ESP32-CAM

Щоб почати транслювати потокове відео з модуля ESP32-CAM, натисніть Start Stream. Приклад отриманого результату трансляцій відео в режимі UXGA з роздільною здатністю 1600x1200 представлений рисунку 2.10.

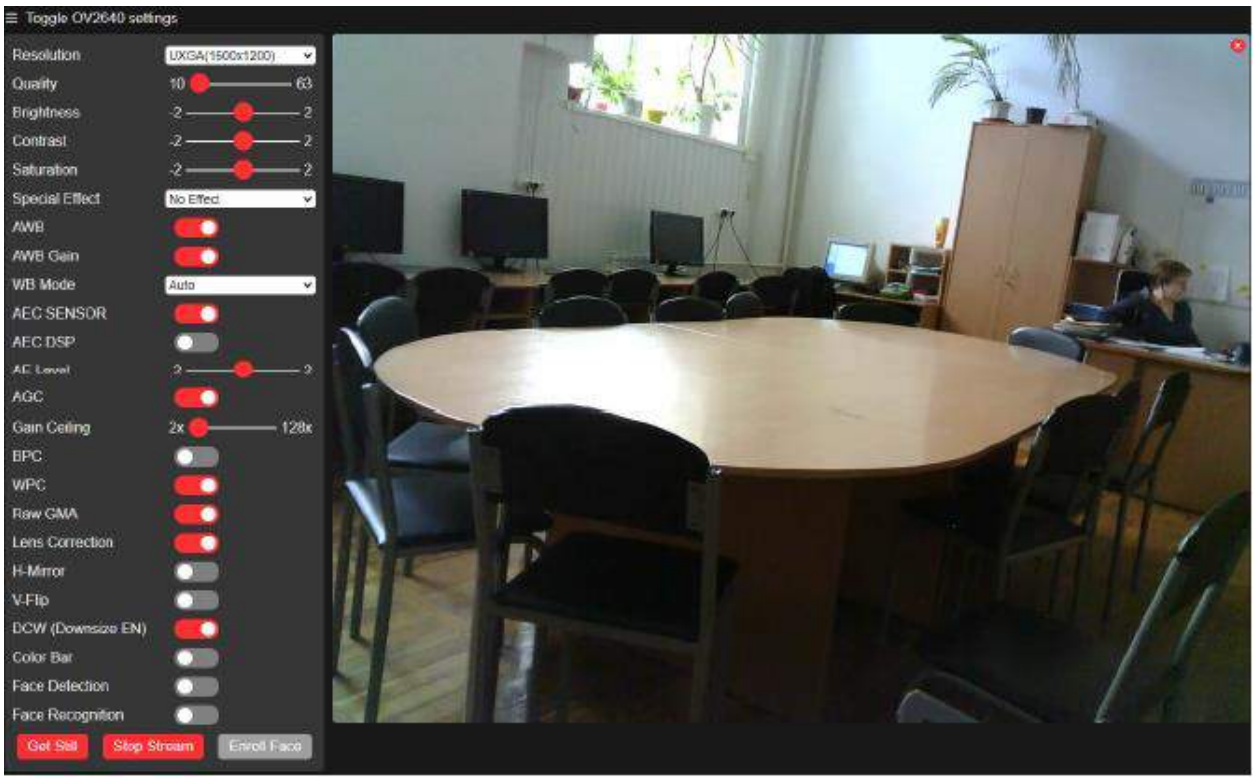


Рисунок 2.10 - Трансляцій відео в режимі UXGA з роздільною здатністю 1600x1200

Послідовність прошивки системи керування мобільним роботом ESP32-Cam, яка описана в підрозділі 2.3 така сама, як прошивка тестового скетчу для ESP32-CAM «CameraWebServer».

2.5 HMI інтерфейс системи керування мобільним роботом

Обрання реалізації системи керування мобільним роботом на базі "тонкого клієнта" HTTP протоколу має ряд переваг [61]. Використання цього підходу дозволяє створити простий інтерфейс для керування роботом через веб-браузер без необхідності встановлення додаткового програмного забезпечення на мобільний пристрій. Крім того, система буде працювати на будь-якому пристрої з можливістю запуску веб-браузера, що дозволяє операторам використовувати різні пристрої для керування роботом. HTTP протокол також дозволяє керувати роботом з великої відстані через Інтернет, що може бути корисним у віддалених або небезпечних обставинах. Реалізація

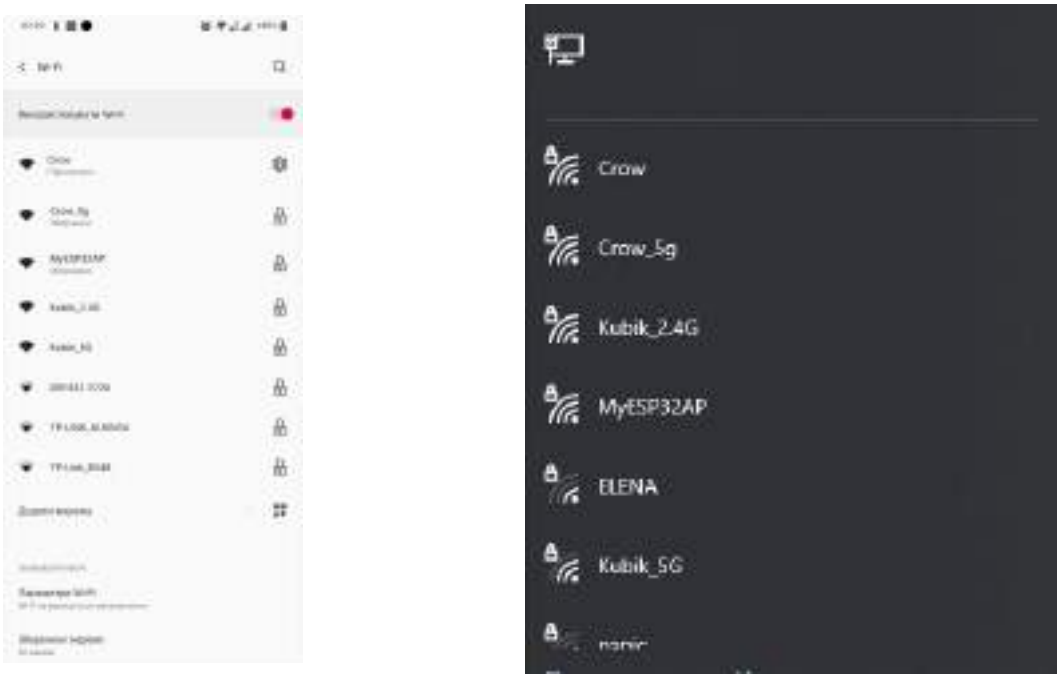
системи на базі HTTP протоколу вимагає мінімальних зусиль і може бути легко реалізована на ESP32-Cam. Використання механізмів автентифікації та авторизації може забезпечити захист від несанкціонованого доступу до системи керування. Також важливою перевагою є можливість легкої розширення функціоналу системи та її інтеграція з іншими системами.

Для реалізацій створення точки доступу на базі модуля ESP32-Cam в середовище розробки Arduino IDE необхідно реалізувати наступній код:

```
#include <WiFi.h>
const char *ssid = "MyESP32AP";
const char *password = "12345678";
void setup() {
  Serial.begin(115200);
  delay(1000);
    Serial.println("Setting up Access Point...");
  WiFi.softAP(ssid, password);
  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);
}
```

У даному коді встановлюється ESP32-Cam як точка доступу з ім'ям " MyESP32AP " та паролем "12345678". Для цього використовується бібліотека WiFi, яка дозволяє працювати з мережею Wi-Fi. Після запуску пристрою викликається функція setup, яка ініціалізує зв'язок через порт UART для взаємодії з ESP32-Cam. Також в цій функції встановлюється з'єднання з точкою доступу, яка створюється за допомогою методу softAP із вказаними ім'ям та паролем. Після успішного налаштування точки доступу отримується її IP-адреса за допомогою методу softAPIP і виводиться на вбудований порт UART для подальшого використання. Функція loop викликається постійно, але в даному випадку вона порожня і не містить додаткових команд. Такий підхід дозволяє створити простий інтерфейс для керування ESP32-Cam через

веб-браузер, що робить його доступним для використання на різних пристроях без необхідності встановлення додаткового програмного забезпечення. Приклад реалізацій підключення до створеної Wi-Fi точки на базі ESP32-Cam представлено на рисунку 2.11



а) б)

а) реалізацій підключення до точки доступу на OS Android;

б) реалізація точки доступу OS Windows.

Рисунок 2.11 - Приклад реалізацій підключення до створеної Wi-Fi точки на базі ESP32-Cam

При розробці інтерфейсу оператора для керування гусеничним мобільним роботом на базі ESP32-Cam з системою комп'ютерного зору рекомендується додати такі елементи:

- відеопотік з камери, включення відеопотоку з камери на інтерфейсі оператора для візуального контролю навколишнього середовища;

- відображення результатів аналізу зображень, показати результати аналізу зображень, такі як виявлення обличчя, об'єктів, ліній, або інших елементів, що можуть бути корисними для оператора при прийнятті рішень;

- елементи керування аналітикою, додати елементи керування для запуску аналізу зображень та відображення результатів на інтерфейсі;

- панель інструментів для обробки зображень, забезпечити доступ до інструментів для обробки зображень, таких як збільшення, обрізання, фільтри тощо, для додаткового аналізу;

- індикатори статусу системи комп'ютерного зору, відображення індикаторів, що показують статус та результати роботи системи комп'ютерного зору (наприклад, активний/неактивний, стан розпізнавання тощо);

-можливість взаємодії з результатами аналізу, додати можливість взаємодії з результатами аналізу зображень, наприклад, вказувати точки інтересу або виправляти помилки розпізнавання;

- системи навігації для відображення мапи або шляху руху робота на інтерфейсі оператора;

- системи автоматичного керування, можливість переключення між ручним та автоматичним режимами керування на основі результатів аналізу зображень.

Додавання цих елементів дозволить створити повноцінний інтерфейс оператора з системою комп'ютерного зору для керування гусеничним мобільним роботом на базі ESP32-Cam, що дозволить оператору більш ефективно керувати роботом та взаємодіяти з ним. Фрагмент коду реалізацій системи керування мобільним роботом на базі ESP32-Cam в середовище Arduino IDE приведені в додатку К, а результат реалізацій системі керування приведено на рисунку 2.12.

Рисунок 2.12 - Результат реалізацій системі керування мобільним роботом

"Photo" в системі керування мобільним роботом на базі ESP32-Cam використовується для зйомки фотографій або захоплення кадрів відео з камери робота. Кнопка "Restart" дозволяє оператору здійснювати швидкий та зручний перезапуск робота в разі потреби, що полегшує керування та обслуговування мобільного робота на базі ESP32-Cam. Кнопка "Stop Stream" в системі керування мобільним роботом на базі ESP32-Cam призначена для припинення потокового відтворення відео з камери робота. Кнопки "Forward", "Stop", "TurnLeft","TurnRight" та "Backward" дозволяє оператору керувати переміщенім мобільного робота в просторі. Налаштування "Flash" дає можливості включення освітлення на модулі ESP32-Cam. "Speed" дозволяє прискорити швидкість обороту двигунами або зменшить. "Quality" та "Resolution" дозволяють налаштувати якість зображення отриманий з камери OVE 2560.

3. РЕАЛІЗАЦІЯ МЕТОДІВ І АЛГОРИТМІВ ОКОНТУРУВАННЯ ОБ'ЄКТІВ У ЗОНІ РОБОЧОЇ ЗОНІ МОБІЛЬНОГО РОБОТА В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

3.1 Метод аналізу зображень побудованого на виявленні змін яскравості на зображенні

У системах комп'ютерного зору оконтурювання – це процес визначення контурів об'єктів на зображенні [62]. Контур – це лінія, яка з'єднує крапки з однаковою яскравістю чи кольором. Окантовка може бути використана для виділення об'єктів у робочій зоні робота. У системах робототехніки оконтурювання може використовуватися для таких цілей [63]:

- визначення положення об'єктів дозволяє визначити положення об'єктів у робочій зоні робота. Це може бути корисним для роботів, які повинні маніпулювати об'єктами або уникати перешкод.

- розпізнавання об'єктів можна використовувати для розпізнавання об'єктів на зображенні. Це може бути корисним для роботів, які повинні виконувати завдання, що потребують ідентифікації об'єктів, таких як сортування або збирання.

- вимір розмірів об'єктів може використовуватися для вимірювання розмірів об'єктів на зображенні. Це може бути корисним для роботів, які повинні виконувати завдання, що вимагають вимірювання розмірів об'єктів, таких як упаковка або виробництво.

В даному випадку пропонується використовувати метод аналіз зображень, побудованого на виявленні змін яскравості на зображенні [64]. Опишемо реалізацію даного методу з використанням математичних виразів. Нехай $I(x,y)$- яскравість зображення в точці з координатами (x,y). Проведемо перетворення вихідного зображення робочої зони робота на двійкове зображення з використанням порогової бінаризації:

$$B(x,y) = \begin{cases} 1, \text{if } I(x,y) > Threshold \\ 0 \end{cases} \quad (3.1)$$

Наступною дією необхідно визначити пов'язані області (компоненти) на бінарному зображенні. Це створює безліч пов'язаних компонентів, кожен є групою пікселів, об'єднаних зв'язком у межах цієї групи. На початковому етапі кожен піксель вважається окремою компонентою, і кожна компонента містить лише один піксель.

$$R = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\} \quad (3.2)$$

де: m - кількість пікселів.

Аналізуючи зображення перевіряємо якщо два суміжні пікселі мають значення (1), то об'єднуємо компоненти, до яких вони належать:

$$Union(R_i, R_j), \text{if pixels } (x_i, y_i) \text{ and } (x_j, y_j) > 1 \quad (3.3)$$

Після об'єднання компонентів знаходимо безліч зв'язаних компонентів, що об'єдналися пікселів.

$$R = \{R_1, R_2, ..., R_k\} \quad (3.4)$$

де: k - кількість пов'язаних компонентів.

Для кожної пов'язаної компоненти витягуємо список пікселів, що належать до цієї компоненти.

$$R_k = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), ..., (x_{in}, y_{in})\} \quad (3.5)$$

Після цього проводимо індексацію, щоб кожна компонента була унікально ідентифікована.

Наступним кроком для кожної компоненти, яка є об'єктом, витягуються контури. Це можна зробити, використовуючи алгоритм проходження кордону об'єкта. Кожен контур представляється списком точок (x_i, y_i):

$$c_i = \{(x_{i1}, y_{i1}), (x_{i2}, y_{i2}), ..., (x_{in}, y_{in})\} \quad (3.6)$$

Після цього будуємо ієрархію контурів. Ієрархія вказує на відносини між контурами, наприклад, які з них внутрішні, зовнішні або знаходяться на одному рівні. Нехай c_i - це i контур, а H_i - відповідна інформація про ієрархію для i контуру.

$$H_i = [Next_i, \Pr ev_i, FirstChild_i, Parent_i] \quad (3.7)$$

де: $Next_i$ - наступний контур на тому самому рівні ієрархії. Якщо це -1, то наступного контуру на рівні немає;

$\Pr ev_i$ - попередній контур на тому самому рівні ієрархії. Якщо це -1, то попереднього контуру на рівні немає;

$FerstChild_i$ - перший контур, який є дочірнім по відношенню до поточного контуру. Якщо це -1, то контур не має дочірніх контурів.

$Parent_i$ - є батьком по відношенню до поточного контуру. Якщо це -1 то контур знаходиться на верхньому рівні ієрархії.

Таким чином, ієрархія контурів є масивом H, де кожен елемент H_i містить інформацію про зв'язки i-го контуру з іншими контурами на тому ж рівні ієрархії, а також про зв'язок з дочірніми та батьківськими контурами.

Щоб перевірити правильність міркувань, ми розробимо програму на Python у середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації математичних виразів, описаних вище.

Підключимо бібліотеки:

Імпорт CV2

імпортувати numpy як np

cv2 — це бібліотека OpenCV для комп'ютерного зору. Він надає різноманітні функції для роботи із зображеннями та відео, включаючи читання, запис, обробку та аналіз.

numpy — це бібліотека для наукових обчислень на Python. Він надає багато зручних функцій для роботи з багатовимірними масивами (такими як матриці), що робить його корисним для обробки зображень та операцій лінійної алгебри.

Створимо функцію def calculate_kidney_contour_size(image_path):, яка бере в якості аргументу шлях до зображення (image_path). Він призначений для аналізу контурного зображення.

Завантаження зображення

img = cv2.imread(image_path, cv2. IMREAD_GRAYSCALE)

Цей фрагмент коду завантажує зображення за вказаним шляхом image_path використання бібліотеки OpenCV (cv2). Прапор cv2. IMREAD_GRAYSCALE вказує на те, що зображення буде завантажуватися у відтінках сірого, тобто в одноканальному режимі, де кожен піксель представлений одним числовим значенням, яке відображає яскравість цього пікселя.

Це корисно у випадку, коли інформація про колір зображення не потрібна, а достатньо лише інформації про яскравість. Це також може прискорити обробку зображень, оскільки багато алгоритмів обробки зображень працюють в одноканальному режимі.

scale_percent = 100 # Відсоток від початкового розміру

```
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
```

Цей фрагмент коду обчислює нові розміри зображення на основі відсоткової шкали відносно його початкових розмірів.

scale_percent – змінна, яка містить відсоток від вихідного розміру, на який буде масштабовано зображення. Наприклад, якщо scale_percent встановлено на 100, то розміри зображення залишаться незмінними.

width і height - це змінні, які обчислюють нову ширину і висоту зображення відповідно. Вони обчислюються шляхом множення вихідних розмірів зображення на scale_percent з подальшим діленням на 100.

dim - кортеж, що містить нову ширину і висоту зображення (width, height).

Ці розрахунки не змінюють фактичний вміст зображення, а лише визначають його нові розміри відповідно до вказаного відсотка масштабування. Нові розміри будуть використовуватися для зміни масштабу зображення за допомогою відповідного методу масштабування, наприклад, у функції cv2.resize().

```
# Зміна розміру зображення
img = cv2.resize(img, dim, interполяція=cv2. INTER_AREA)
```

Цей фрагмент коду змінює розмір зображення за допомогою функції cv2.resize() з бібліотеки OpenCV (cv2).

Параметр img представляє оригінальне зображення, яке буде змінено.

Параметр dim містить нові розміри зображення, які були розраховані на попередньому кроці.

Параметр інтерполяції визначає метод інтерполяції, який буде використовуватися для зміни розміру зображення. У цьому випадку використовується cv2. INTER_AREA, що означає метод інтерполяції, який підходить для зменшення розміру зображення. Цей метод зазвичай дає

хороші результати в зменшенні розміру зображення, збереженні деталей і зниженні шумів.

Як тільки цей код буде виконаний, змінна img міститиме змінене зображення, яке тепер має нові розміри, визначені змінною dim.

Застосування фільтрації та бінаризації для виділення контурів

_, thresh = cv2.threshold(img, 120, 255, cv2. THRESH_BINARY)

Цей фрагмент коду застосовує фільтрацію та бінаризацію до зображення, щоб виділити контури.

Функція cv2.threshold() використовується для бінаризації зображення. Він перетворює зображення в двійкове зображення (тільки чорно-біле), де всі пікселі, значення яскравості яких вище порогового значення, встановлюються на максимальне значення (в даному випадку 255), а решта - на мінімальне значення (в даному випадку - 0). Порогове значення задається другим параметром функції.

Змінна thresh міститиме двійкове зображення.

Перший повернутий параметр (який ігнорується за допомогою _) зазвичай містить порогове значення, яке було обчислено автоматично, якщо поріг не вказано явно.

Цей фрагмент коду готує зображення до подальшого пошуку шляху, перетворюючи його на двійкове зображення, де шляхи більш виражені.

Пошук контурів

contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

Цей фрагмент коду знаходить контури в двійковому зображенні за допомогою функції cv2.findContours() з бібліотеки OpenCV (cv2).

Параметр thresh являє собою двійкове зображення, на якому будуть знайдені шляхи.

Розділ 2. RETR_EXTERNAL визначає метод пошуку контурів. У цьому випадку використовується RETR_EXTERNAL, який знаходить тільки

зовнішні контури і не враховує контури, які знаходяться в межах інших контурів.

cv2.CHAIN_APPROX_SIMPLE Визначає метод апроксимації контурів. У цьому випадку використовується CHAIN_APPROX_SIMPLE, який апроксимує контури шляхом стиснення горизонтальних, вертикальних і діагональних відрізків і залишення тільки їх кінцевих точок.

Після виконання цього рядка коду змінна contours міститиме список контурів, знайдених у двійковому зображенні.

Малювання контурів на новому зображенні

contour_img = np.zeros_like(img)

cv2.drawContours(contour_img, contours, -1, (255), 2)

Фрагмент коду створює нове зображення contour_img, яке намалює шляхи, знайдені на оригінальному зображенні.

np.zeros_like(img) створює масив нулів з тими ж розмірами та типом даних, що й оригінальне зображення img. Цей масив буде використовуватися для створення зображення, на якому будуть намальовані контури.

Функція cv2.drawContours() використовується для малювання контурів на зображенні. Для цього потрібні такі аргументи:

contour_img - зображення, на якому будуть намальовані контури.

contours - список контурів, які будуть намальовані.

-1 - індекс контуру у списку contours. Значення -1 вказує на те, що будуть намальовані всі контури зі списку.

(255) - колір контуру у форматі (B, G, R). У цьому випадку використовується лише один канал для яскравості (відтінку сірого) контурів.

2 - товщина лінії контуру.

Після виконання цього рядка коду змінна contour_img міститиме зображення з намальованими контурами.

Розрахунок розмірів контуру

contour_sizes = [cv2.contourArea(contour) for contour in contours]

фрагмент коду обчислює розміри (площу) кожного контуру у списку contours з використанням функції cv2.contourArea() з бібліотеки OpenCV (cv2).

cv2.contourArea(contour) обчислює площу контуру, переданого як аргумент. Кожен елемент у списку contours є контуром, знайденим на бінаризованому зображенні.

Змінна contour_sizes міститиме список площ контурів. Кожен елемент цього списку відповідатиме площі відповідного контуру зі списку contours.

Цей фрагмент коду дозволяє отримати інформацію про розміри контурів, які можуть бути використані для аналізу або класифікації об'єктів на зображенні.

return contour_img, contour_sizes

Цей фрагмент коду завершує виконання функції calculate_kidney_contour_size. Він повертає два значення:

contour_img: зображення, на якому намальовані контури.

contour_sizes: перелік розмірів (площ) контурів.

Ці значення можуть бути використані в програмі для аналізу контурів точкового зображення. Наприклад, contour_img може бути відображено для візуалізації контурів, а contour_sizes можуть бути використані для подальшого аналізу розмірів контурів і виявлення будь-яких особливостей або аномалій.

Приклад використання

image_path = 'Pic/12.jpg' # Замініть на шлях до вашого зображення

contour_img, contour_sizes = calculate_kidney_contour_size(image_path)

Фрагмент коду показує приклад використання функції calculate_kidney_contour_size. Він встановлює шлях до зображення змінну image_path, а потім викликає функцію calculate_kidney_contour_size з цим шляхом.

Результат виконання функції – зображення з намальованими контурами contour_img та список розмірів (площ) контурів contour_sizes – зберігаються у відповідних змінних.

Після виконання цього фрагмента коду змінні contour_img та contour_sizes будуть містити результати аналізу зображення. Ці результати можуть бути використані для подальшого аналізу або відображення, залежно від вимог програми

```
# Виведення зображення з контурами точок
cv2.imshow('Kidney Contours', contour_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Цей фрагмент коду використовує функцію cv2.imshow() для відображення зображення з намальованими контурами точок.

'Kidney Contours' - це рядок, який задає заголовок вікна, в якому буде відображено зображення з контурами точок.

contour_img – це зображення з намальованими контурами, яке буде відображено.

cv2.waitKey(0) очікує натискання клавіші для закриття вікна із зображенням. Параметр 0 означає нескінченне очікування.

cv2.destroyAllWindows() закриває всі вікна, створені бібліотекою OpenCV.

Після виконання цього коду ви побачите вікно із зображенням, на якому будуть намальовані контури точок.

```
# Виведення розмірів контурів
for i, size in enumerate(contour_sizes):
 print(f"Contour {i + 1} Size: {size}")
```

Цей фрагмент коду виводить розміри (площі) кожного контуру на консоль.

enumerate(contour_sizes) створює перелік, який включає індекси елементів списку contour_sizes разом із самими значеннями.

У циклі for змінні i та size перебираються з цього перерахування.

Форматований рядок "Contour {i + 1} Size: {size}" виводить номер контуру (i + 1, оскільки індексація починається з нуля) і його розмір (size).

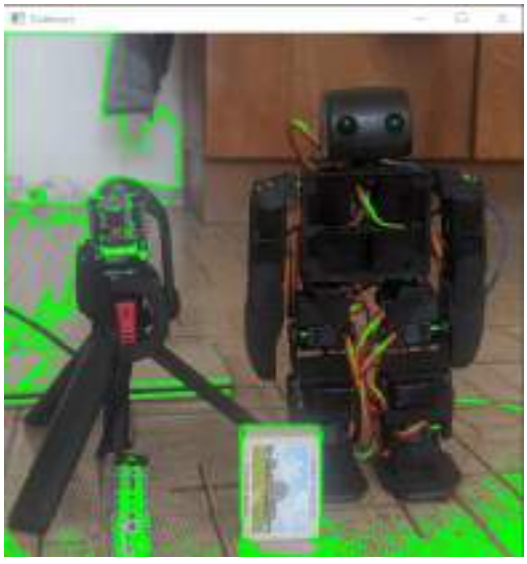
print() виводить цей рядок на консоль.

Таким чином, після виконання цього коду буде виведено розміри (площі) всіх контурів на консоль. Проведемо дослідження впливу порогового значення інтенсивності пікселя на оконтурювання об'єкта у кадрі. Результати дослідження наведено на рисунку 3.1.

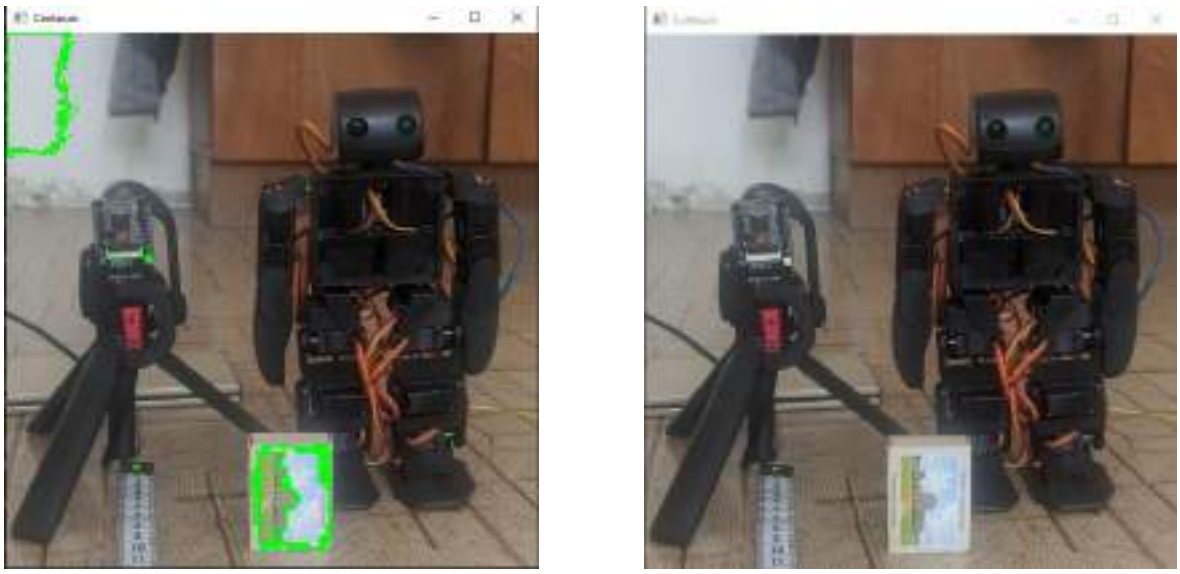
а)

б) (50x50)

в) (100x100)



г) (150x150)



ґ) (200x200) д) (250x250)

а) вихідне зображення; б) граничне значення інтенсивності пікселя 50;
в) граничне значення інтенсивності пікселя 100;
г) граничне значення інтенсивності пікселя 150;
ґ) граничне значення інтенсивності пікселя 200;
д) граничне значення інтенсивності пікселя 250;

Рисунок 3.1 - Вплив порогового значення інтенсивності пікселя на оконтурювання об'єкта

Як можна бачити з проведеного експерименту, в даному випадку для оконтурювання сірникової коробки, яка знаходиться в робочій зоні, рекомендоване граничне значення інтенсивності пікселя приблизно дорівнює 200, для більш точного налаштування необхідно досвідченим шляхом налаштувати $\pm\Delta$ від цього числа. Повний код програми наведено у додатку Л.

Метод аналізу зображень на основі виявлення змін яскравості є ефективним підходом для пошуку контуру об'єкта в робочій зоні мобільного робота. Цей метод ґрунтується на аналізі відмінностей яскравості пікселів між двома або кількома зображеннями, що дозволяє виявляти зміни, що відповідають контурам об'єктів.

Перевагою є його здатність виявляти контури об'єктів без необхідності заздалегідь відомої моделі об'єкта. Це робить його придатним для ситуацій, де об'єкти можуть мати різні форми та розміри.

Для успішної реалізації методу аналізу зображень з урахуванням виявлення змін яскравості рекомендується враховувати кілька аспектів. По-перше, необхідно правильно вибирати граничні значення для визначення змін яскравості, щоб мінімізувати помилкові спрацьовування. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб забезпечити швидку обробку та виявлення контурів об'єктів.

Також рекомендується використовувати оптимізації, такі як алгоритми зменшення шуму та фільтрації зображень, щоб поліпшити якість виявлення контурів об'єктів [65]. Важливо також враховувати зміни у освітленні та умовах довкілля для більш точного виявлення контурів об'єктів.

В цілому, метод аналізу зображень на основі виявлення змін яскравості є ефективним інструментом для пошуку контурів об'єктів у робочій зоні мобільного робота. Його здатність працювати без попередньої моделі об'єкта робить його універсальним та придатним для різних завдань у галузі робототехніки.

3.2 Алгоритм Canny

Алгоритм Canny – це метод виявлення кордонів на зображенні, розроблений Джоном Канні у 1986 році [66]. Він ґрунтується на кількох ключових кроках. Спочатку зображення піддається згладжуванню фільтром Гаусса для зменшення шуму. Потім обчислюються градієнти зображення з використанням операторів Собеля у горизонтальному та вертикальному напрямках [67]. Алгоритм Canny є ефективним засобом виділення меж, що підтримує високу чутливість до контурів та мінімізує вплив шумів на зображення [68]. Це робить його широко використовується в комп'ютерному зорі для завдань, таких як розпізнавання об'єктів, відстеження об'єктів, що рухаються, і виявлення контурів. Загальний вид реалізації алгоритму Canny для отримання контуру об'єкта в режимі реального часу з камери представлений на рисунку 3.2.

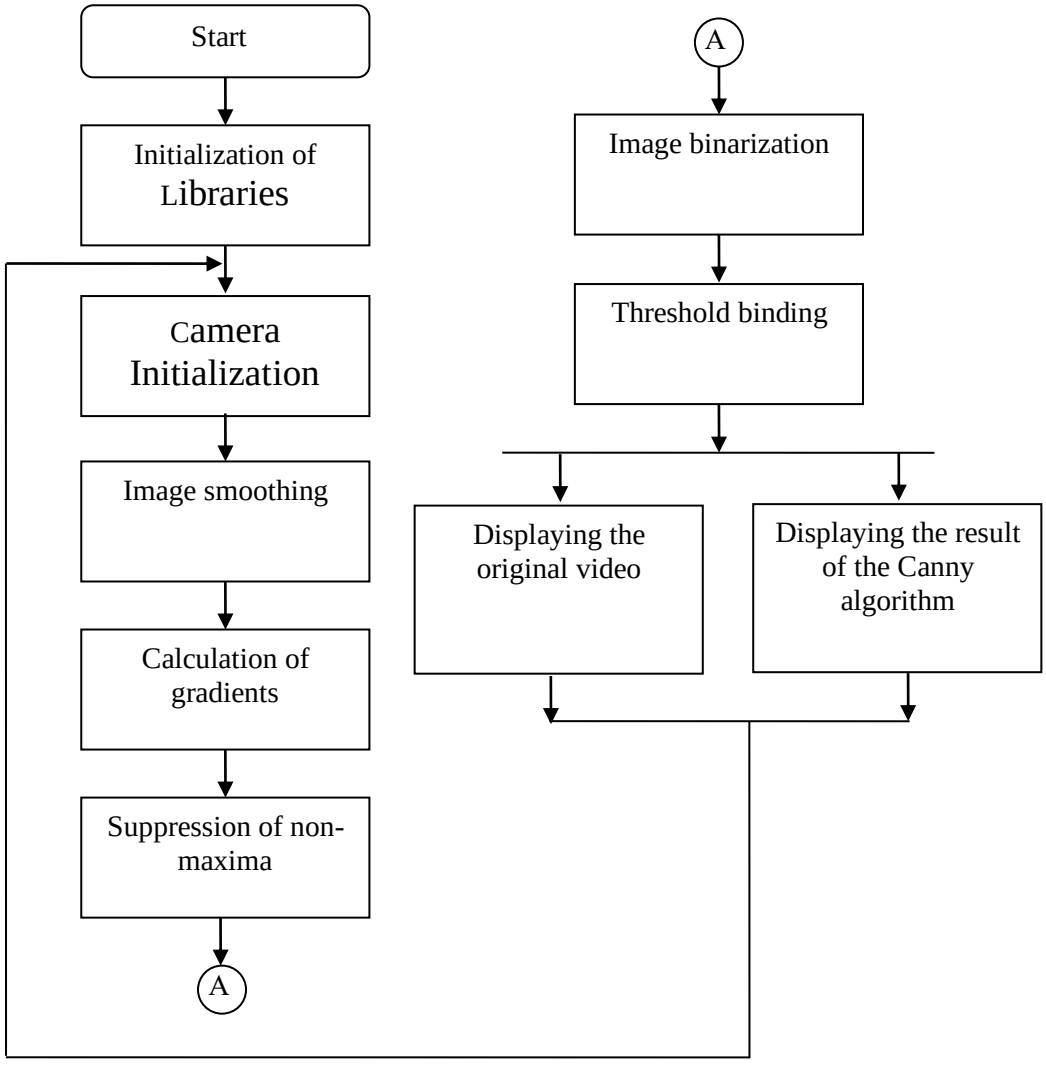


Рисунок 3.2 - Загальний вид реалізації алгоритму Canny для отримання контуру об'єкта в режимі реального часу з камери

Для програмної реалізації алгоритму Canny для отримання контуру об'єкта в режимі реального часу проведемо математичний опис базових кроків алгоритму (рис.3.2). Згладжування зображення, засноване на застосування фільтра гауса для згладжування шумів і підготовки зображення для подальшої обробки, загальний вигляд якого представлений нижче:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} \tag{3.8}$$

Де: $G(x, y)$ - ядро гаусівського фільтра, що визначається двовимірним розподілом Гауса. Це ядро є двовимірною функцією, яка визначає вагу кожного пікселя під час згортки;

x, y - координати пікселя у зображенні;

σ - параметр, який визначає стандартне відхилення (розмиття) функції Гауса. Чим більше σ, тим більше розмите зображення.

$\frac{1}{2\pi\sigma^2}$ - нормалізуючий коефіцієнт, який використовується для нормалізації значень ваги в ядрі. Він забезпечує, щоб сума всіх значень в ядрі дорівнювала 1, що зберігає яскравість зображення;

$e^{-(x^2+y^2)/(2\sigma^2)}$ - експоненційна частина функції Гауса, де e - основа натурального логарифму. Ця частина визначає вагу кожного пікселя в ядрі відповідно до відстані центрального пікселя. Чим далі піксель від центру, тим менша його вага.

Наступним кроком проводимо обчислення градієнта, використовуючи оператор Собеля [69]. Позначимо через G_x - матриця ядра для горизонтального оператора Собеля, а через G_y - матриця ядра для вертикального оператора Собеля, де x, y - координати пікселя у зображенні. Звідси оператор Собеля для цього випадку можна представити так:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.9}$$

Оператори Собеля обчислюють градієнт зображення у напрямках x і y, та їх комбінація дозволяє визначити амплітуду градієнта та напрямок зміни інтенсивності в кожному пікселі. Це забезпечує виділення країв та меж об'єктів на зображенні.

Для визначення локальних максимумів використовують вікно (зазвичай 3x3) навколо кожного пікселя. Якщо амплітуда градієнта у

поточному пікселі ($M(x,y)$) більше, ніж у сусідніх пікселях, розташованих у напрямку градієнта ($\theta(x,y)$), то поточний піксель вважається локальним максимумом.

Амплітуда градієнта (M) у кожному пікселі обчислюється як:

$$M(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{3.10}$$

Де : G_x - матриця ядра для горизонтального оператора Собеля;

G_y - матриця ядра для вертикального оператора Собеля;

x, y - координати пікселя у зображенні.

Напрямок градієнта (θ) у кожному пікселі обчислюється як:

$$\theta(x,y) = \arctan(\frac{G_y(x,y)}{G_x(x,y)}) \tag{3.11}$$

Звідси локальний максимум можна представити як:

$$M(x,y) > M(x \pm 1, y \pm 1) \rightarrow \theta(x,y) \tag{3.12}$$

Процес придушення не-максимумів допомагає зберегти лише локальні максимуми у напрямку градієнта, що дозволяє виділити тонкі межі об'єктів на зображенні.

Наступним кроком необхідно провести бінаризацію отриманого зображення. Бінарізація - це процес перетворення зображення у формат, що представляє кожен піксель як один із двох можливих значень: чорний або білий [70]. У цьому випадку бінаризацію можна подати так:

$$E(x,y) = \begin{cases} 1, if M(x,y) > T_{upper} \\ 0, if M(x,y) < T_{lower} \\ \text{Marked for linking, if } \ T_{lower} < M(x,y) < T_{upper} \end{cases} \tag{3.13}$$

де: $M(x, y)$ - амплітуда градієнта;

T_{upper} - верхній поріг;

T_{lower} - нижній поріг.

Останнім кроком проводитися зв'язування меж, в даному випадку ми будемо використовувати метод зв'язування по гістерезісу, для з'єднання сильних і слабких граничних пікселів та формування кривих контуру.

У контексті мобільних роботів застосування алгоритму Canny у потоковому відео з камери дозволяє роботам ефективно виділяти та аналізувати межі об'єктів у реальному часі, що є важливим компонентом для навігації та безпеки в робочій зоні.

Для перевірки правильності міркувань, розробимо програму мовою Python у середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації вище описаних математичних виразів.

```
import cv2
import time
```

Код імпортує бібліотеки cv2 та time. Бібліотека cv2 є OpenCV, що широко використовується бібліотекою для обробки зображень та комп'ютерного зору в Python. Бібліотека time надає функції для роботи з часом, такі як затримка виконання (time.sleep()).

```
def main():
    # Відкриття відеопотоку з камери (зазвичай 0 для вбудованої камери)
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Помилка при відкритті камери.")
        return
```

функція main() є основною частиною програми. Вона відповідає за відкриття відеопотоку з камери та перевірку успішності цієї операції.

cv2.VideoCapture(0) створює об'єкт відео, який буде використовуватися для отримання кадрів з камери. Аргумент 0 вказує на індекс пристрою, з якого потрібно отримати відеопотік.

cap.isOpened() перевіряє, чи відеопотік успішно відкритий, метод повертає True, і програма продовжує виконання.

У разі виникнення помилки під час відкриття камери програма припиняє виконання за допомогою оператора return.

Створення вікна для відображення вихідного відео

cv2.namedWindow('Original Video', cv2.WINDOW_NORMAL)

Цей фрагмент коду створює вікно під назвою "Original Video" для відображення вихідного відеопотоку.

cv2.namedWindow() створює вікно із заданою назвою.

Параметр cv2.WINDOW_NORMAL вказує, що розмір вікна можна змінювати вручну користувачем.

Після виконання цього фрагмента коду буде створено вікно з назвою "Original Video", яке використовуватиметься для відображення вихідного відеопотоку з камери.

Створення вікна для відображення результату роботи алгоритму Canny

cv2.namedWindow('Canny Edge Detection', cv2.WINDOW_NORMAL)

Цей фрагмент коду створює вікно під назвою "Canny Edge Detection" для відображення результату роботи алгоритму детектування меж Canny.

cv2.namedWindow() створює вікно із заданою назвою.

Параметр cv2.WINDOW_NORMAL вказує, що розмір вікна можна вручну змінювати користувачем.

Після виконання цього фрагмента коду буде створено вікно з назвою "Canny Edge Detection", яке використовуватиметься для відображення результатів детектування кордонів за допомогою алгоритму Canny.

hile True:

    # Захоплення кадру з відеопотоку

```
ret, frame = cap.read()
if not ret:
    print("Не вдалося отримати кадр.")
    break
```

Цей фрагмент коду є нескінченним циклом, який отримує кадри з відеопотоку, поки потік кадрів доступний.

cap.read() захоплює кадр із відеопотоку, повертаючи True у змінну ret, якщо кадр успішно захоплений, і сам кадр у змінну frame.

Якщо захоплення кадру не вдалося (наприклад, через помилку або досягнення кінця відеопотоку), змінна ret буде False, і програма виведе повідомлення про помилку "Не вдалося отримати кадр". Потім програма виходить із циклу за допомогою оператора break.

Цей фрагмент коду забезпечує отримання кадрів з відеопотоку та переривання циклу у разі виникнення помилки або завершення потоку.

```
# Вимірювання часу початку обробки кадру
start_time = time.time()
```

Цей фрагмент коду виконує вимірювання часу початку обробки кожного кадру відеопотоку.

time.time() повертає поточний час у секундах з початку епохи (зазвичай, починаючи з 1 січня 1970 року). Значення цієї функції зберігається у змінній start_time.

Вимірювання часу початку обробки кадру може бути корисним для оцінки часу виконання алгоритмів обробки кадрів або для оцінки продуктивності програми в цілому.

```
# Перетворення зображення на відтінки сірого
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Цей фрагмент коду перетворює кожен захоплений кадр зображення у відтінки сірого.

cv2.cvtColor() використовується для перетворення колірного простору зображення. Він приймає два аргументи: вихідне зображення frame і

цільовий колірний простір cv2.COLOR_BGR2GRAY, який позначає перетворення з простору кольору BGR в відтінки сірого (grayscale).

Результат перетворення зберігається у змінній gray.

Перетворення зображення у відтінки сірого часто використовується при обробці зображень, так як воно спрощує аналіз зображення та зменшує обчислювальне навантаження..

Застосування алгоритму Canny

edges = cv2.Canny(gray, 50, 150)

Цей фрагмент коду застосовує алгоритм детектування меж Canny для зображення у відтінках сірого.

cv2.Canny() приймає три аргументи: вихідне зображення у відтінках сірого (gray), мінімальний поріг для детектування кордонів (50) та максимальний поріг для детектування кордонів (150).

Алгоритм Canny шукає межі в зображенні, використовуючи градієнт яскравості, і створює двійкове зображення, де пікселі на кордонах встановлені в максимальне значення (білий), а решта пікселів встановлені в мінімальне значення (чорний).

Результат застосування алгоритму Canny зберігається у змінній edges.

Відображення вихідного відео

cv2.imshow('Original Video', frame)

Фрагмент коду відображає вихідне відео (кадр у кольоровому форматі) у вікні під назвою "Original Video".

cv2.imshow() використовується для відображення зображення у вікні. Перший аргумент - це назва вікна, в якому буде відображено зображення.

Після виконання цього фрагмента коду вікно з назвою "Original Video" відображатиме поточний кадр із відеопотоку.

Відображення результату роботи алгоритму Canny

cv2.imshow('Canny Edge Detection', edges)

Цей фрагмент коду відображає результат роботи алгоритму детектування меж Canny у вікні під назвою "Canny Edge Detection".

cv2.imshow() використовується для відображення зображення у вікні. Перший аргумент – це назва вікна, в якому буде відображено зображення. Другий аргумент - це зображення, отримане в результаті алгоритму роботи Canny (edges).

Після виконання цього фрагмента коду вікно під назвою "Canny Edge Detection" відображатиме результати детектування кордонів на поточному кадрі з відеопотоку.

```
# Замір часу закінчення обробки кадру
 end_time = time.time()
```

Цей фрагмент коду виконує вимірювання часу закінчення обробки кожного кадру відеопотоку.

time.time() повертає поточний час у секундах з початку епохи (зазвичай, починаючи з 1 січня 1970 року). Значення цієї функції зберігається у змінній end_time.

Вимірювання часу закінчення обробки кадру може бути корисним для оцінки часу, витраченого на обробку кожного кадру, і для оцінки продуктивності алгоритмів обробки кадрів.

```
# Розрахунок швидкості обробки кадрів та побудови контуру
 processing_speed = 1 / (end_time - start_time)
```

Цей фрагмент коду обчислює швидкість обробки кадрів та побудови контуру.

end_time - start_time обчислює час, витрачений обробку кадру і побудова контуру, шляхом віднімання часу початку обробки start_time з часу закінчення обробки end_time.

1/(end_time - start_time) обчислює зворотне значення часу, тобто швидкість обробки кадрів у секундах на кадр. Якщо час обробки кадру близький до нуля, швидкість буде дуже високою.

Розрахунок швидкості обробки кадрів та побудови контуру може бути корисним для оцінки продуктивності алгоритмів обробки кадрів та для визначення, наскільки швидко працює програма.

```
# Виведення результатів у термінал
 print(f"Швидкість обробки: {processing_speed:.2f} кадрів за секунду")
```

Цей фрагмент коду виводить результати обробки кадру у термінал.

f"Швидкість обробки: {processing_speed:.2f} кадрів за секунду" - це рядок форматування, який виводить швидкість обробки кадрів, округлену до двох знаків після коми, із зазначенням одиниці виміру - кадри за секунду.

print() використовується для виведення рядка у термінал.

Цей фрагмент коду виводить швидкість обробки кадрів у термінал для моніторингу продуктивності програми.

```
# Звільнення ресурсів та закриття вікон
 cap.release()
 cv2.destroyAllWindows()
if __name__ == "__main__":
 main()
```

Цей фрагмент коду звільняє ресурси камери та закриває всі відкриті вікна після завершення роботи програми.

cap.release() звільняє ресурси, зайняті об'єктом відеозахоплення cap. Це важливо для коректного звільнення ресурсів камери після завершення програми.

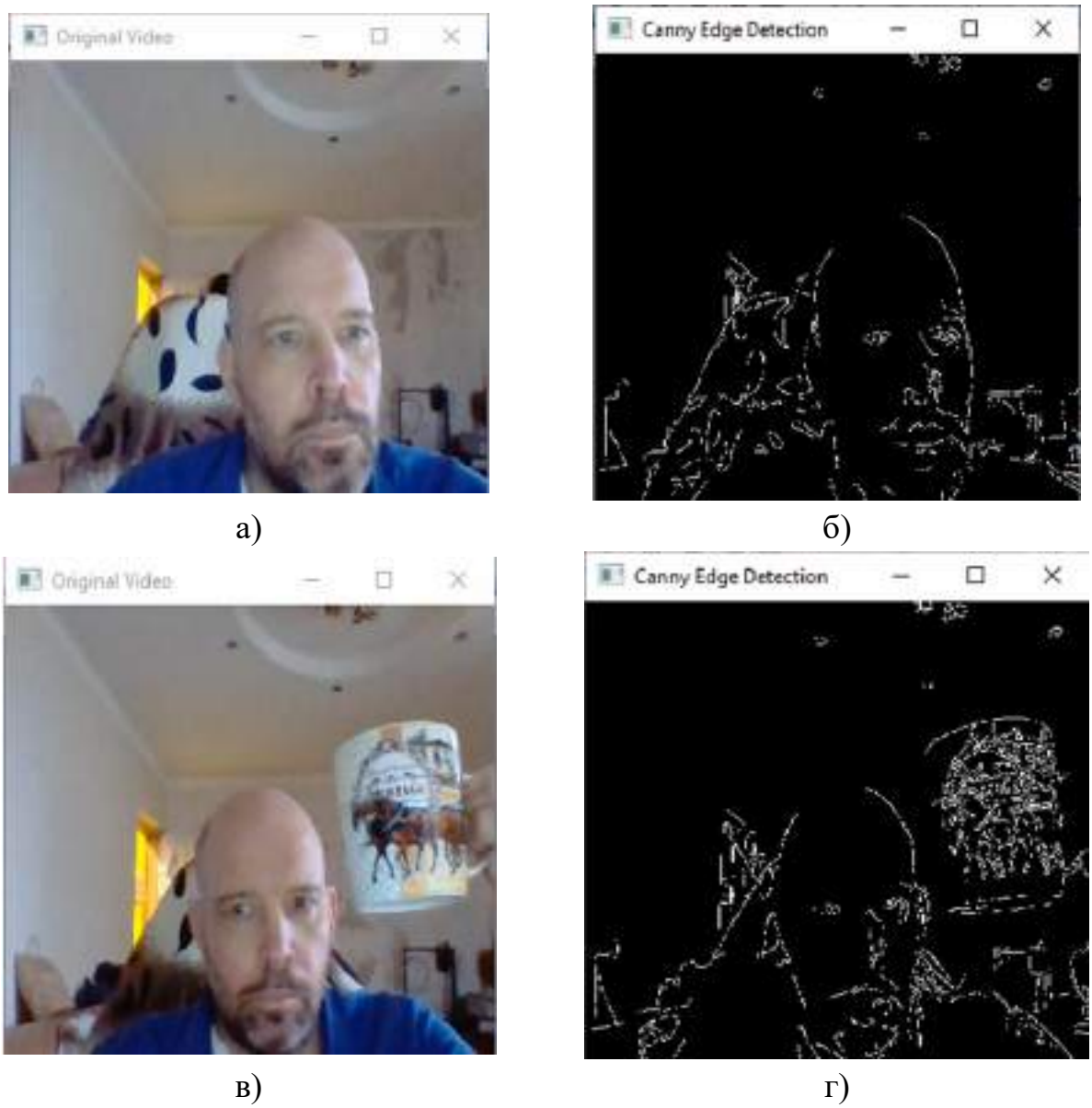
cv2.destroyAllWindows() закриває всі вікна, які були створені функцією cv2.namedWindow().

Код if __name__ == "__main__": використовується для виконання функції main(), якщо скрипт запускається безпосередньо, а не імпортується до іншого модуля.

Цей фрагмент коду забезпечує правильне завершення роботи програми та коректне звільнення ресурсів після використання.

Для проведення досліджень використовувалося наступне апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 2). Програма для отримання контуру об'єкта в режимі

реального часу з камери розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Результати роботи програми представлені рисунку 3.3.



а) б)

в) г)

а),в) – Original Video; б), г) – Canny Edge Detection

Рисунок 3.3. – Результати роботи програми для отримання контуру об'єкта в режимі реального часу

У ході проведення експерименту отримання контуру об'єкта в режимі реального часу з камери так само були виміряні показники швидкості обробки для всіх випадків представлених на рисунку 3.3, в середньому швидкість обробки коливалася в межах 1000.07 - 1002.70 кадрів в секунду. Повний код програми наведено у додатку А.

Метод Canny є одним із найбільш широко використовуваних та ефективних методів виявлення контурів об'єктів у робочій зоні мобільного робота. Він ґрунтується на виділенні меж об'єктів на зображенні шляхом пошуку локальних максимумів градієнта яскравості. Перевагою методу Canny є його здатність виявляти контури об'єктів з високою точністю та мінімальним рівнем шуму.

Для успішної реалізації методу Canny для пошуку контуру об'єкта у робочій зоні мобільного робота рекомендується враховувати декілька аспектів. По-перше, необхідно вірно налаштовувати параметри алгоритму, такі як розмір вікна фільтра та порогові значення для детекції кордонів. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб мінімізувати затримки та забезпечити швидке виконання.

Також рекомендується використовувати оптимізації, такі як паралельні обчислення та використання апаратного прискорення для покращення продуктивності методу на мобільних роботах. Важливо також враховувати умови освітлення та фон зображення для точного виявлення контурів об'єктів.

Загалом метод Canny є ефективним інструментом для виявлення контурів об'єктів у робочій зоні мобільного робота. Його висока точність і низький рівень шуму роблять його придатним для різних завдань у галузі робототехніки, де потрібне точне виявлення контурів об'єктів..

3.3 Метод активних контурів

Метод активних контурів є ефективним підходом у мобільній робототехніці для виділення меж об'єктів на зображеннях [71]. Ця техніка забезпечує мобільним роботам здатність адаптуватися до різноманітних форм об'єктів у навколишньому середовищі, що особливо важливо за умови навігації в різних умовах [72-74]. За допомогою методу активних контурів роботи можуть проводити сегментацію об'єктів, що полегшує сприйняття

навколишнього середовища та прийняття рішень у реальному часі. Крім того, цей метод враховує контекст сцени, що підвищує точність виділення меж у складних сценаріях, де умови освітлення чи наявність шуму можуть бути змінними. Інтеграція методу активних контурів з іншими сенсорами, такими як лідари або радари, забезпечує мобільним роботам більш надійне сприйняття навколишнього простору та покращує загальну навігацію [75]. Результати виділення контурів можуть бути використані для корекції траєкторії руху робота, що робить цей метод важливим інструментом у забезпеченні безпечного та ефективного функціонування мобільних роботів.

Позначимо через $E_{\text{int}ernal}$ - внутрішню енергію контуру, яка зазвичай представляється математичними термінами, такими як пружність або згладжування, і вона описує, наскільки контур "прагне" до певної форми. У методі активних контурів ця енергія мінімізується в процесі пошуку оптимального положення контуру. $E_{external}$ - зовнішню енергію. У контексті методу активних контурів в обробці зображень, зовнішня енергія контуру є енергією, що походить із зовнішніх впливів або даних, наприклад, інтенсивності пікселів на зображенні. Ця енергія сприяє залученню контуру до певних фрагментів зображення, які містять об'єкти, що цікавлять. Отже, загальна енергетичну функцію можна визначити так:

$$E_{total} = E_{\text{int}ernal} + E_{external} \tag{3.14}$$

Тепер опишемо еволюцію контуру шляхом мінімізації загальної енергії. Це досягається зміною положення контуру в напрямку протилежному градієнту загальної енергії по відношенню до контурної кривої. Еволюція контуру методом активних контурів може бути представлена математично у вигляді диференціального рівняння. Основна ідея полягає в тому, щоб рухати контур у напрямку протилежному градієнту загальної енергії по відношенню до контурної кривої. Позначимо контурну

криву як $C(s,t)$, де s - довжина контуру, t - параметр часу. Тоді еволюція контуру може бути представлена рівнянням:

$$\frac{dC}{dt} = -\frac{dE_{total}}{dC} \tag{3.15}$$

Тепер заглибимося в частину з градієнтом загальної енергії по відношенню до контурної кривої:

$$\frac{dE_{total}}{dC} = \frac{dE_{\text{int}ernal}}{dC} + \frac{dE_{external}}{dC} \tag{3.16}$$

Таким чином, ми отримуємо рівняння еволюції контуру:

$$\frac{dC}{dt} = -\frac{dE_{\text{int}ernal}}{dC} - \frac{dE_{external}}{dC} \tag{3.17}$$

Це рівняння визначає, як контур змінюється з часом у напрямку, протилежному градієнту загальної енергії. Вирішення цього рівняння дозволяє контуру еволюціонувати та підлаштовуватися під межі об'єкта на зображенні.

Зауважимо що у методі активних контурів $E_{\text{int}ernal}$ включає два терміни: термін деформації контуру та термін довжини контуру [76]:

- деформація контуру стосується змін форми або положення замкнутої кривої (контуру) на зображенні. У контексті методу активних контурів деформація контуру відбувається у відповідь на вплив зовнішніх сил та енергій, таких як градієнт яскравості або кольору на зображенні. Чим ближче контур до меж об'єктів, тим менше деформації, і навпаки;

- довжина контуру, це фізична або математична характеристика замкнутої кривої, що є межею об'єкта на зображенні. У методі активних контурів довжина контуру розглядається в контексті енергії контуру. Коли контур наближається до меж об'єктів, його довжина зазвичай зменшується,

оскільки метод прагне мінімізувати енергію контуру, вирівнюючи його із зовнішніми факторами, такими як яскравість чи колір.

Звідси $E_{\text{int}ernal}$ набуває наступного вигляду:

$$E_{\text{int}ernal} = \alpha \cdot Length(C) + \beta \cdot \int (|\nabla C|^2) dx \tag{3.18}$$

де: α - параметр, що регулює жорсткість контуру.

β - параметр, що регулює деформацію контуру

$Length(C)$ - довжина контуру C

∇C - градієнт контуру.

Зовнішня енергія $E_{external}$ оцінює, наскільки контур відповідає об'єкту, який хочемо виділити. Ця енергія зазвичай базується на інтенсивності зображення і може включати додаткові фактори, такі як градієнти і текстури. Внаслідок чого $E_{external}$ у загальному вигляді можна описати наступним виразом:

$$E_{external} = \int_C w_1 \cdot I(x,y) + w_2 \cdot \| \nabla I(x,y) \| + w_3 \cdot texture(x,y) ds \tag{3.19}$$

де: w_1, w_2, w_3 - вагові коефіцієнти, що дозволяють налаштовувати внесок кожного фактору;

$\nabla I(x,y)$ - градієнт інтенсивності зображення;

$texture(x,y)$ - функція, що оцінює текстурні характеристики у точці (x,y).

Таким чином, зовнішня енергія є виваженою комбінацією інтенсивності пікселів, градієнтів і текстур вздовж контуру [77]. Підстроювання вагових коефіцієнтів дозволяє настроювати чутливість до різних аспектів зображення при виділенні контуру об'єкта.

Простежимо за еволюцією контуру. Еволюція контуру в контексті методу активних контурів (Active Contour Model або Snake) є процесом зміни

положення контурної кривої на зображенні з часом або ітерацій [78]. Мета цього процесу - підстроювання контуру так, щоб він краще відповідав межам об'єкта, який потрібно виділити на зображенні. В основному контур еволюціонує у напрямку мінімізації загальної енергії.

$$C(t+1) = C(t) - \frac{dE_{total}}{dC} \tag{3.20}$$

де: $\frac{dE_{total}}{dC}$ - градієнт загальної енергії по відношенню до контурної кривої.

Метод активних контурів часто використовується для сегментації об'єктів на зображеннях і може бути адаптований для різних умов та вимог задач.

Для перевірки правильності міркувань розробимо програму мовою Python в середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації вище описаних математичних виразів.

import cv2

import numpy as np

импортировали библиотеки cv2 (OpenCV) і numpy. Ці бібліотеки широко використовуються для обробки зображень, комп'ютерного зору та наукових обчислень у Python.

def active_contour(image_path, alpha=0.015, beta=10, gamma=0.001):

функція active_contour виглядає як початок реалізації алгоритму активних контурів (active contours), також відомого як алгоритм змії (snake algorithm). Цей алгоритм використовується для сегментації об'єктів на зображеннях.

Далі представлено опис аргументів функції:

image_path: Цей аргумент представляє шлях до зображення, на якому застосовуватиметься алгоритм активних контурів..

alpha, beta, gamma: Ці аргументи є параметрами алгоритму. Вони керують поведінкою контуру у процесі сегментації. Зазвичай alpha відповідає за згладжування контуру, beta за його гнучкість, а gamma за облік градієнта яскравості зображення. Значення за замовчуванням (alpha=0.015, beta=10, gamma=0.001) можуть бути змінені відповідно до вимог конкретного завдання або зображення.

```
# Завантаження зображення
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Цей фрагмент коду завантажує зображення із зазначеного шляху image_path за допомогою функції cv2.imread(), а потім перетворює його у відтінки сірого за допомогою функції cv2.cvtColor().

cv2.imread(image_path): Завантажує зображення із зазначеного шляху image_path. Під час завантаження зображення воно перетворюється на формат BGR.

cv2.cvtColor(img, cv2.COLOR_BGR2GRAY): Перетворює зображення img із формату BGR на відтінки сірого. Результат зберігається у змінній gray.

Після виконання цього фрагмента коду змінна gray міститиме відтінки сірого зображення, яке використовуватиметься для подальших операцій з алгоритмом активних контурів..

```
# Вибір прямокутної області для ініціалізації контуру
    rect = cv2.selectROI("Select ROI", img)
    cv2.destroyAllWindows()
```

Цей фрагмент коду дозволяє користувачеві вибрати прямокутну область (Region of Interest, ROI) на зображенні для ініціалізації контуру.

cv2.selectROI("Select ROI", img): Ця функція відображає вікно із зображенням img, де користувач може вибрати прямокутну область. Користувач може виділити область за допомогою миші, затиснувши ліву кнопку миші та виділяючи потрібну область. Після завершення виділення області користувач повинен натиснути клавішу "Enter" або "Space" для

завершення операції. Вибрана область зберігається у змінній rect у вигляді кортежу (x, y, w, h), де (x, y) – координати верхнього лівого кута прямокутника, а (w, h) – його ширина та висота відповідно.

cv2.destroyAllWindows(): Закриває усі відкриті вікна. Це викликається для закриття вікна вибору прямокутної області після завершення операції вибору.

Після виконання цього фрагмента коду змінна rect міститиме координати та розміри прямокутної області, обраної користувачем. Ця область може бути використана для ініціалізації контуру в алгоритмі активних контурів

Отримання контуру на основі обраної області

points = cv2.convexHull(np.array([[rect[0], rect[1]], [rect[0], rect[1] + rect[3]], [rect[0] + rect[2], rect[1] + rect[3]], [rect[0] + rect[2], rect[1]]]))

Цей фрагмент коду використовує алгоритм побудови опуклої оболонки (cv2.convexHull()) для отримання контуру на основі вибраної прямокутної області.

np.array([[rect[0], rect[1]], [rect[0], rect[1] + rect[3]], [rect[0] + rect[2], rect[1] + rect[3]], [rect[0] + rect[2], rect[1]]]): Цей код створює масив NumPy з координатами вершин прямокутника, що описує вибрану прямокутну область. Координати вершин виходять із змінної rect, де rect[0] та rect[1] - це координати верхнього лівого кута прямокутника, а rect[2] та rect[3] - його ширина та висота відповідно.

cv2.convexHull(): Ця функція приймає масив координат точок контуру та повертає контур опуклої оболонки (convex hull). В даному випадку вона використовується для побудови контуру на основі обраної прямокутної області.

Результат виконання цього фрагмента коду – це змінна points, яка містить контур опуклої оболонки на основі обраної прямокутної області. Цей контур може бути використаний як початкове наближення контуру в алгоритмі активних контурів.

```
# Ініціалізація активного контуру
    snake = points.reshape((-1, 1, 2))
```

Цей фрагмент коду виконує ініціалізацію активного контуру (snake) на основі опуклої контуру оболонки (points), отриманого раніше.

points.reshape((-1, 1, 2)): Цей метод змінює форму масиву points таким чином, щоб він став тривимірним масивом з розміром (n, 1, 2), де n - кількість точок контуру. Це необхідно для створення структури даних, сумісної з алгоритмом активних контурів OpenCV.

Після виконання цього фрагмента коду змінна snake міститиме ініціалізований активний контур, який можна використовувати для подальшої роботи з алгоритмом активних контурів

```
# Процес еволюції контуру
    epsilon = 0.1  # Параметр для апроксимації
    snake = cv2.approxPolyDP(snake, epsilon, closed=True)
```

Цей фрагмент коду виконує процес еволюції контуру за допомогою апроксимації.

cv2.approxPolyDP(snake, epsilon, closed=True): Ця функція використовується для апроксимації контуру. Вона приймає три аргументи:

snake: вихідний контур, який потрібно апроксимувати.

epsilon: параметр апроксимації, який вказує максимальну відстань від вихідного контуру до апроксимованого. Чим менше значення epsilon, тим точнішою буде апроксимація. У даному випадку epsilon дорівнює 0.1.

closed=True: цей аргумент показує, що контур замкнутий.

Функція cv2.approxPolyDP() повертає апроксимований контур, який замінює вихідний контур snake.

Після виконання цього фрагмента коду змінна snake міститиме апроксимований контур, який може бути використаний для подальшої обробки або візуалізації.

```
# Відображення контуру на зображенні
    img_contour = img.copy()
```

cv2.polylines(img_contour, [np.int32(snake)], isClosed=True, color=(0, 255, 0), thickness=2)

Цей фрагмент коду відображає контур на вихідному зображенні.

img.copy(): Створює копію вихідного зображення img. Це робиться для того, щоб зміни, внесені до img_contour, не впливали на вихідне зображення.

cv2.polylines(): Ця функція використовується для відображення полілінії (що складається з кількох ліній) на зображенні. Вона приймає кілька аргументів:

img_contour: вихідне зображення, на якому буде відображено полілінію.

[np.int32(snake)]: контур, який потрібно відобразити. np.int32(snake) використовується для перетворення координат контуру snake у цілий формат.

isClosed=True: вказує, що контур замкнутий.

color=(0, 255, 0): колір контуру (зелений у форматі BGR).

thickness=2: товщина контуру в пікселях.

Після виконання цього фрагмента коду змінна img_contour міститиме вихідне зображення з намальованим контуром.

```
# Відображення вихідного зображення та зображення з контуром
cv2.imshow('Original Image', img)
cv2.imshow('Active Contour', img_contour)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Цей фрагмент коду відображає вихідне зображення та зображення з намальованим контуром на двох різних вікнах.

cv2.imshow(): Ця функція використовується для відображення зображення у вікні. Перший аргумент – це назва вікна, в якому буде відображено зображення, а другий аргумент – саме зображення.

cv2.waitKey(0): Цей метод очікує натискання клавіші на клавіатурі перед закриттям вікон. Тут він очікує натискання будь-якої клавіші (0), перш ніж продовжити виконання програми.

cv2.destroyAllWindows(): Цей метод закриває всі відкриті вікна.

Після виконання цього фрагмента коду два вікна з вихідним зображенням та зображенням із намальованим контуром будуть відображені, і виконання програми призупиниться до натискання клавіші на клавіатурі

Приклад використання

image_path = 'Pic/12.jpg'

active_contour(image_path)

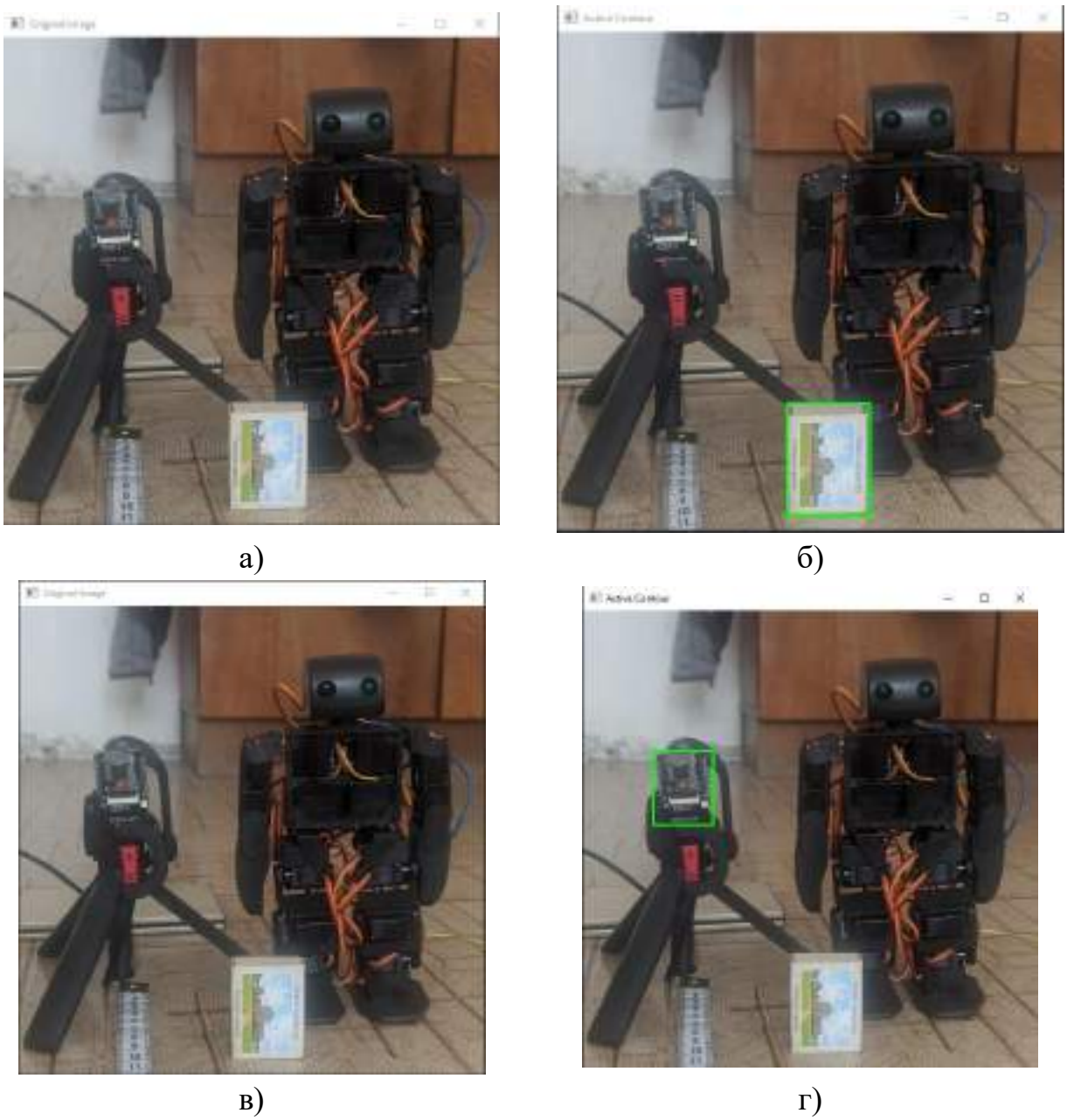
Цей фрагмент коду є прикладом використання функції active_contour, яка була визначена раніше.

image_path = 'Pic/12.jpg': Тут вказується шлях до зображення, на якому буде використано алгоритм активних контурів. Зазвичай, це шлях до файлу зображення на комп'ютері.

active_contour(image_path): Цей виклик функції active_contour передає шлях до зображення image_path. Функція виконуватиме алгоритм активних контурів на цьому зображенні.

Після виконання цього фрагмента коду функція active_contour буде застосована до зображення, вказаного шляхом image_path, і результати будуть відображені на екрані

Для проведення досліджень використовувалось наступнк апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 2). Результати роботи програми представлені рисунку 3.4.



а) б)

в) г)

а),в) Оригінальне зображення; б), г) виділений кордон об'єкту;

Рисунок 3.4 - Результат роботи розробленої програми заснований на методі активних контурів

Як можна бачити з отриманих результатів експерименту, виділення об'єктів на базі методу активних контурів, дозволяє визначити наявність об'єкта, отримати його контур для подальшого розпізнавання. Приклад програмного коду наведено у додатку Б

Метод активних контурів, або змійок, є ефективним підходом для пошуку контуру об'єкта в робочій зоні мобільного робота. Цей метод заснований на ідеї енергетичної мінімізації, де контур об'єкта представлений

замкнутою кривою, яка змінюється таким чином, щоб мінімізувати певну енергію, пов'язану з контуром.

Перевагою методу активних контурів є його здатність адаптуватися до різних форм об'єктів та змін у їхньому контурі [79]. Це робить його придатним для пошуку контуру об'єкта у різних умовах та сценаріях використання.

Для успішної реалізації методу активних контурів рекомендується враховувати кілька аспектів [80]. По-перше, необхідно правильно вибирати параметри моделі енергії, такі як жорсткість контуру та сили, що діють на контур. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб мінімізувати затримки та забезпечити плавне виконання.

Також рекомендується використання оптимізації, такі як паралельні обчислення та використання апаратного прискорення для покращення продуктивності методу на мобільних роботах. Важливо також враховувати умови освітлення та фон зображення для точного виявлення контурів об'єктів[81].

В цілому, метод активних контурів є ефективним інструментом для пошуку контуру об'єкта в робочій зоні мобільного робота. Його здатність адаптуватися до різних умов та форм об'єктів робить його придатним для широкого спектру завдань у галузі робототехніки.

3.4 Метод оптичного потоку та алгоритму Грехема

Оптичний потік (optical flow) – це векторне поле, яке описує рух об'єктів між двома послідовними кадрами відео [82-84]. Він дозволяє відстежувати рух пікселів від одного кадру до іншого і виявляти об'єкти, що рухаються. Принцип роботи методу оптичного потоку можна описати наступним чином, для кожного пікселя (x, y) на зображенні в момент часу t ми можемо знайти його нове становище $(x + \Delta x, y + \Delta y)$ у момент часу $t + \Delta t$ за допомогою наступного виразу:

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x, y, t) \tag{3.21}$$

де: $I(x, y, t)$ - інтенсивність пікселя на момент часу t.

Алгоритм Грехема використовується для пошуку контуру об'єкта на зображенні. Він ґрунтується на побудові опуклої оболонки для заданого набору крапок. Принцип роботи алгоритму Грехема можна описати так, на першому етапі вибирається стартова точка контуру (P_0), яка є найлівішою та нижньою точкою або точкою з найменшою координатою (x_0, y_0). Наступним кроком проводимо сортування $P_1, P_2, ..., P_n$ по полярному кутку щодо стартової точки. Нехай $P_0 = (x_0, y_0)$ - координати стартової точки, а $P_i = (x_i, y_i)$ - координати інших точок. Тоді полярний кут θ_i для кожної точки P_i може бути розрахований як:

$$\theta_i = \arctan(\frac{y_i - y_0}{x_i - x_0}) \tag{3.22}$$

де: θ_i - полярний кут P_i щодо стартової точки P_0.

Після обчислення полярних кутів для всіх інших точок, ми можемо відсортувати їх за зростанням або зменшенням кутів. Таким чином, точки будуть упорядковані по полярному куту щодо стартової точки P_0.

Після цього проводимо побудову опуклої оболонки, використовуючи стек для зберігання вершин оболонки, для цього на базі всіх відсортованих точок ($P_1, P_2, ..., P_n$) у порядку їх полярного кута (θ_i) щодо стартової точки (P_0). Далі, починаючи з самої лівої та нижньої точки (P_0), ми йдемо по відсортованих точках. Додаємо перші дві точки у стек. Для кожної наступної точки перевіряємо, чи є вона "лівою" або "правою" щодо попередніх двох точок у стеку. Це можна зробити за допомогою визначника матриці повороту для трьох точок. P_i, P_j, P_k, де P_i і P_j - дві останні додані точки в стек, а P_k - поточна точка. Якщо визначник (O_p) позитивний, то точка P_k

знаходиться "ліворуч" відрізка P_i, P_j і вона додається до стеку. Якщо визначник негативний або дорівнює нулю, то точка P_k знаходиться "праворуч" відрізка P_i, P_j, і останні дві точки витягуються зі стека, поки виконується умова, після чого точка P_k додається до стек. Математично це може бути представлено через визначник матриці повороту для трьох точок. $P_i(x_i, y_i), P_j(x_j, y_j), P_k(x_k, y_k)$.

$$O_p = (x_j - x_i)(y_k - y_i) - (y_j - y_i)(x_k - x_i) \tag{3.33}$$

Якщо визначник позитивний, то точка P_k знаходиться "ліворуч" відрізка P_i, P_j і вона додається до стеку. Якщо визначник негативний або дорівнює нулю, то останні дві точки витягуються з стека, поки виконується умова, після чого точка P_k додається до стек. Приклад побудови опуклої оболонки, використовуючи стек для зберігання вершин оболонки, представлений на рисунку 3.5.

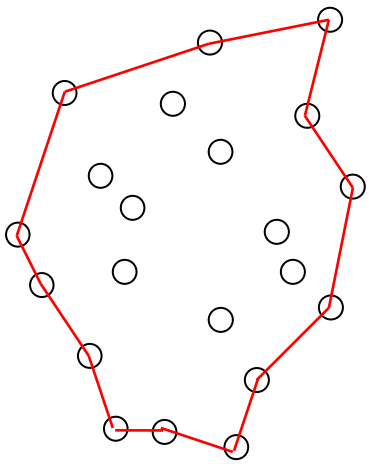


Рисунок 3.5 - Приклад побудови опуклої оболонки, використовуючи стек для зберігання вершин оболонки

На останньому кроці проводимо видалення неконвексних кутів, якщо вони присутні шляхом перевірки на поворот вліво (або вправо) для кожних трьох послідовних точок P_i, P_j, P_k. Алгоритм Грехема дозволяє ефективно

виділити контур об'єкта на зображенні, що може бути корисно для мобільних роботів в робочих зонах, де необхідно виявляти перешкоди та інші об'єкти для безпечної навігації. Застосування оптичного потоку для попередньої фільтрації об'єктів, що рухаються, дозволяє поліпшити продуктивність алгоритму Грехема і точність виявлення контуру об'єкта [85-87].

Для перевірки правильності міркувань розробимо програму мовою Python в середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації вище описаних математичних виразів.

```
import cv2
import time
import numpy as np
```

Здійснили імпортування бібліотек cv2 (OpenCV), time і numpy.

```
def main():
```

Створили функцію main(). Зазвичай функція main() використовується у Python для запуску основної логіки програми. Давайте продовжимо та створимо тіло функції.

```
# Відкриття відеопотоку з камери
    cap = cv2.VideoCapture(0)
```

Цей фрагмент коду відкриває відеопотік із камери.

cv2.VideoCapture(0): Створює об'єкт захоплення відео (cap), який відкриває відеопотік із камери. Аргумент 0 вказує на те, що потрібно використовувати першу доступну камеру. Якщо у вас є кілька камер, ви можете вказати номер потрібної камери, наприклад, 1, 2 тощо.

Зверніть увагу, що перед використанням камери переконайтеся, що вона підключена та доступна для використання на вашому пристрої.

```
# Перевірка успішного відкриття відеопотоку
    if not cap.isOpened():
        print("Помилка: Неможливо відкрити відеопотік.")
        return
```

Цей фрагмент коду перевіряє, чи був успішно відкритий відеопотік з камери.

cap.isOpened(): Цей метод повертає логічне значення (True або False), що вказує, чи був успішно відкритий відеопотік. Якщо відеопотік успішно відкрито, метод поверне True, інакше False.

if not cap.isOpened(): Ця конструкція перевіряє, якщо cap.isOpened() повернув False, що означає, що відеопотік не був успішно відкритий.

Якщо відеопотік не був успішно відкритий, виводиться повідомлення про помилку, і виконання функції завершується за допомогою оператора return. Це дозволяє уникнути подальшого виконання коду, якщо відеопотік не доступний, і запобігти можливим помилкам у подальшому коді.

```
# Створення вікон для відображення відеопотоку та контуру
    cv2.namedWindow('Video', cv2.WINDOW_NORMAL)
    cv2.namedWindow('Graham Scan Contour', cv2.WINDOW_NORMAL)
```

Цей фрагмент коду створює вікна для відображення відеопотоку та контуру на зображенні.

cv2.namedWindow('Video', cv2.WINDOW_NORMAL): Цей метод створює вікно із заданою назвою 'Video', в якому відображатиметься відеопотік. Аргумент cv2.WINDOW_NORMAL вказує на те, що розмір вікна може бути змінений користувачем.

cv2.namedWindow('Graham Scan Contour', cv2.WINDOW_NORMAL): Цей метод створює вікно під назвою 'Graham Scan Contour', в якому відображатиметься контур на зображенні. Аргумент cv2.WINDOW_NORMAL також вказує на розмір вікна, що змінюється.

Створення цих вікон дозволить відображати відеопотік та контур на зображенні в реальному часі.

```
start_time = time.time()
    frame_count = 0
```

Цей фрагмент коду фіксує час початку обробки відеопотоку та ініціалізує змінну frame_count, яка використовуватиметься для підрахунку кількості оброблених кадрів.

time.time(): Функція time.time() повертає поточний час у секундах з початку епохи (починаючи з 1 січня 1970 року). Цей метод використовується для фіксації часу початку обробки відеопотоку.

frame_count = 0: Цей код ініціалізує змінну frame_count значенням 0. Ця змінна використовуватиметься для підрахунку кількості оброблених кадрів. Змінна frame_count збільшуватиметься на 1 після кожного успішного читання кадру з відеопотоку.

Читання першого кадру для ініціалізації методу оптичного потоку.

```
ret, prev_frame = cap.read()
if not ret:
    print("Помилка: Неможливо отримати перший кадр.")
    return
```

Цей фрагмент коду читає перший кадр із відеопотоку для ініціалізації методу оптичного потоку.

cap.read(): Цей метод зчитує кадр із відеопотоку. Він повертає два значення: ret (логічне значення, що показує, чи успішно прочитали кадр) і prev_frame (сам кадр).

if not ret: Ця конструкція перевіряє, чи успішно було прочитано перший кадр. Якщо ret дорівнює False, тобто якщо кадр не був успішно прочитаний, виводиться повідомлення про помилку і виконання функції завершується за допомогою оператора return. Це допомагає уникнути помилок під час спроби обробки відеопотоку без кадрів.

Змінна prev_frame міститиме перший кадр відеопотоку, який використовуватиметься для ініціалізації методу оптичного потоку

```
# Конвертація першого кадру у відтінки сірого
prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
```

Цей фрагмент коду конвертує перший кадр prev_frame у відтінки сірого.

cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY): Ця функція перетворює зображення prev_frame з колірного простору BGR на відтінки сірого. Результат зберігається у змінній prev_gray.

Після виконання цього фрагмента коду змінна prev_gray міститиме відтінки першого сірого кадру, які будуть використовуватися для подальшого обчислення оптичного потоку.

```
# Читання та обробка кадрів з відеопотоку
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Помилка: Неможливо отримати кадр.")
            break
```

Цей фрагмент коду починає цикл, в якому відбувається читання та обробка кадрів із відеопотоку.

while True: Це нескінченний цикл, який буде виконуватися доти, доки не буде перерваний явно оператором break.

ret, frame = cap.read(): Цей код зчитує наступний кадр із відеопотоку. Аналогічно попередньому коду, він повертає два значення: ret (чи успішно прочитали кадр) і frame (сам кадр).

if not ret: Ця конструкція перевіряє, чи успішно було прочитано наступний кадр. Якщо ret дорівнює False, тобто якщо кадр не був успішно прочитаний, виводиться повідомлення про помилку і виконання циклу переривається за допомогою оператора break.

Цей фрагмент коду забезпечує циклічне читання та обробку кадрів із відеопотоку. Якщо відбувається помилка при читанні кадру (наприклад, коли досягнуто кінець відеопотоку), цикл завершується

```
# Конвертація поточного кадру у відтінки сірого
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Цей фрагмент коду конвертує поточний кадр frame у відтінки сірого.

cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY): Ця функція перетворює зображення frame з колірного простору BGR на відтінки сірого. Результат зберігається у змінній gray.

Після виконання цього фрагмента коду змінна gray міститиме відтінки сірого поточного кадру, які будуть використовуватися для подальшої обробки.

Застосування оптичного потоку для отримання точок, що рухаються

flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)

Цей фрагмент коду застосовує метод оптичного потоку Farneback для обчислення точок, що рухаються між двома послідовними кадрами.

cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0): Ця функція приймає два послідовні кадри у відтінках сірого (prev_gray та gray) і обчислює оптичний потік між ними. Вона повертає масив, що містить вектори руху кожного пікселя. Аргументи функції визначають різні параметри методу Farneback, такі як розмір вікна, стандартне відхилення фільтра Гаусса та інші.

Результат виконання цього фрагмента коду міститиме масив flow, який містить інформацію про рух між двома послідовними кадрами. Ця інформація може бути використана для подальшого аналізу руху об'єктів відеопотоку.

Перетворення оптичного потоку на список точок

points = np.argwhere(np.abs(flow) > 2)

Цей фрагмент коду перетворює оптичний потік, обчислений на попередньому кроці, в список точок, що рухаються.

np.abs(flow) > 2: Ця умова перевіряє, чи є абсолютне значення вектора оптичного потоку для кожного пікселя більше 2. Це може бути пороговим значенням для визначення точок, що рухаються.

np.argwhere(): Ця функція повертає індекси елементів масиву, що задовольняють задану умову. Тут вона використовується для отримання індексів пікселів, вектор оптичного потоку для яких перевищують порогове значення.

Результат виконання цього фрагмента коду буде містити список points, який містить координати пікселів, що відповідають точкам, що рухаються на відеопотоці. Ці точки можуть бути оброблені або використані для візуалізації руху об'єктів.

Застосування алгоритму Грехем для отримання контуру об'єкта з візуалізацією точок

contour_image = apply_graham_scan(frame.copy(), points)

Цей фрагмент коду застосовує алгоритм Грехем для отримання контуру об'єкта з використанням точок, отриманих з оптичного потоку.

apply_graham_scan(frame.copy(), points): Це функція, яка приймає копію поточного кадру frame і список точок points, що рухаються, а потім застосовує алгоритм Грехема для отримання контуру об'єкта. Результат зберігається у змінній contour_image.

Після виконання цього фрагмента коду змінна contour_image міститиме зображення з контуром об'єкта, отриманим за допомогою алгоритму Грехема.

Відображення оригінального відео у вікні "Video"

cv2.imshow('Video', frame)

Цей фрагмент коду відображає оригінальне відео у вікні "Video".

cv2.imshow('Video', frame): Ця функція відображає поточний кадр frame у вікні під назвою "Video". Таким чином, вікно "Video" показуватиме оригінальне відео без змін.

Цей фрагмент коду оновлює вікно Video з кожним новим кадром, показуючи оригінальне відео в реальному часі.

Відображення контуру з візуалізацією крапок у вікні "Graham Scan Contour"

cv2.imshow('Graham Scan Contour', contour_image)

Цей фрагмент коду відображає контур об'єкта з візуалізацією крапок у вікні під назвою "Graham Scan Contour".

cv2.imshow('Graham Scan Contour', contour_image): Ця функція відображає зображення contour_image, яке містить контур об'єкта з візуалізацією точок, що рухаються, у вікні з назвою "Graham Scan Contour".

Цей фрагмент коду оновлює вікно Graham Scan Contour з кожним новим кадром, показуючи контур об'єкта з візуалізацією точок у реальному часі.

frame_count += 1

Цей фрагмент коду збільшує значення змінної frame_count на 1 після обробки кожного кадру.

frame_count += 1: Цей рядок збільшує значення frame_count на 1 після кожної обробленої ітерації у циклі. Це використовується для розрахунку загальної кількості оброблених кадрів.

Підрахунок кількості кадрів може бути корисним, наприклад, для обчислення швидкості обробки відеопотоку.

```
# Вихід із програми при натисканні клавіші ESC
 if cv2.waitKey(1) & 0xFF == 27:
 break
```

Цей фрагмент коду дозволяє завершити виконання програми, якщо користувач натисне клавішу ESC.

cv2.waitKey(1): Цей метод очікує натискання кнопки протягом зазначеного часу (у мілісекундах). У разі очікування становить 1 мілісекунду.

& 0xFF == 27: Ця умова перевіряє, чи натиснуто клавішу ESC. При натисканні клавіші ESC метод cv2.waitKey(1) повертає значення 27 (це ASCII-код для клавіші ESC).

if ...: break: Якщо натиснуто клавішу ESC, виконання програми завершується за допомогою оператора break, який виходить з циклу while True.

Таким чином, цей фрагмент коду забезпечує вихід із програми при натисканні клавіші ESC користувачем.

Оновлення попереднього кадру та відтінків сірого

prev_gray = gray.copy()

Цей фрагмент коду оновлює змінні prev_gray та prev_frame, які містять попередній кадр та його відтінки сірого відповідно.

prev_gray = gray.copy(): Цей рядок копіює поточні відтінки сірого gray у змінну prev_gray. Таким чином, змінна prev_gray тепер містить відтінки поточного сірого кадру, які будуть використовуватися на наступній ітерації для обчислення оптичного потоку.

Цей крок важливий, оскільки обчислення оптичного потоку потрібен попередній кадр. Копіювання поточного кадру змінної prev_gray гарантує, що ми будемо використовувати актуальні відтінки сірого при наступному обчисленні.

end_time = time.time()

elapsed_time = end_time - start_time

Цей фрагмент коду фіксує час закінчення обробки відеопотоку та обчислює витрачений на обробку час.

end_time = time.time(): Цей рядок фіксує поточний час у змінній end_time після закінчення обробки відеопотоку.

elapsed_time = end_time - start_time: Цей рядок обчислює різницю між часом початку обробки відеопотоку (зафіксованим у змінній start_time) та часом закінчення обробки (зафіксованим у змінній end_time). Результат зберігається у змінній elapsed_time.

Після виконання цього фрагмента коду змінна elapsed_time міститиме загальний час, витрачений на обробку відеопотоку. Це дозволяє оцінити продуктивність вашого алгоритму обробки відео

Обчислення швидкості обробки кадрів

processing_speed = frame_count / elapsed_time

Цей фрагмент коду обчислює швидкість обробки кадрів відеопотоку.

processing_speed = frame_count / elapsed_time: Цей рядок ділить загальну кількість оброблених кадрів (frame_count) на час, витрачений на обробку відеопотоку (elapsed_time). Результат зберігається у змінній processing_speed.

Ця операція дозволяє визначити швидкість обробки кадрів відеопотоку в кадрах за секунду (FPS - Frames Per Second). Чим вище значення processing_speed, тим швидше обробляється відеопотік.

Виведення результатів розрахунків

print("Швидкість обробки кадрів: {:.2f} кадрів за секунду".format(processing_speed))

Цей фрагмент коду відображає результати обчислень на екрані.

print("Швидкість обробки кадрів: {:.2f} кадрів за секунду".format(processing_speed)): Цей рядок форматує виведення, щоб відобразити швидкість обробки кадрів (processing_speed) з двома знаками після коми, а також додає одиниці виміру ("кадрів за секунду"). Результат виводиться на екран за допомогою функції print().

Цей крок корисний для відображення інформації про швидкість обробки кадрів відеопотоку.

Звільнення ресурсів та закриття вікон

cap.release()

cv2.destroyAllWindows()

Цей фрагмент коду звільняє ресурси, пов'язані з потоком відео, і закриває всі вікна, відкриті за допомогою OpenCV.

cap.release(): Цей метод звільняє ресурси, пов'язані з об'єктом захоплення відео (cap). Це включає звільнення відеопотоку і закриття камери.

cv2.destroyAllWindows(): Этот метод закрывает все окна, которые были открыты с помощью OpenCV, включая окна с видеопотоком и другими изображениями.

Цей фрагмент коду важливий для коректного завершення роботи програми та звільнення ресурсів після завершення обробки відеопотоку

```
def apply_graham_scan(frame, points):
 # Тут має бути ваш код для застосування алгоритму Грехема до точок об'єкта
 # У цьому прикладі відображаються точки об'єкта
 for point in points:
 cv2.circle(frame, (point[1], point[0]), 5, (0, 0, 255), -1)
```

Функція apply_graham_scan застосовує алгоритм Грехема до точок об'єкта, щоб знайти їх опуклу оболонку. У цьому прикладі передбачається, що масив points містить координати точок, що рухаються на зображенні. Алгоритм Грехема використовується для виділення контуру об'єкта.

```
def apply_graham_scan(frame, points):
 # Знаходимо початкову точку (найліву)
 start_point = min(points, key=lambda x: x[1])
 # Сортуємо інші точки по полярному куту
 sorted_points = sorted(points, key=lambda x: (np.arctan2(x[0] - start_point[0], x[1] - start_point[1])))
 # Ініціалізуємо стек для побудови опуклої оболонки
 hull = [sorted_points[0], sorted_points[1]]
 # Проходимо по відсортованих точках і будуємо опуклу оболонку
 for i in range(2, len(sorted_points)):
 while len(hull) > 1 and np.cross(np.array(hull[-2]) - np.array(hull[-1]), np.array(sorted_points[i]) - np.array(hull[-1])) <= 0:
 hull.pop()
 hull.append(sorted_points[i])
 # Малюємо контур об'єкта на кадрі
   for point in hull:
      cv2.circle(frame, (point[1], point[0]), 5, (0, 0, 255), -1)
```

Функція apply_graham_scan завершується поверненням зміненого кадру frame, який містить контур об'єкта, виділений з використанням алгоритму Грехема.

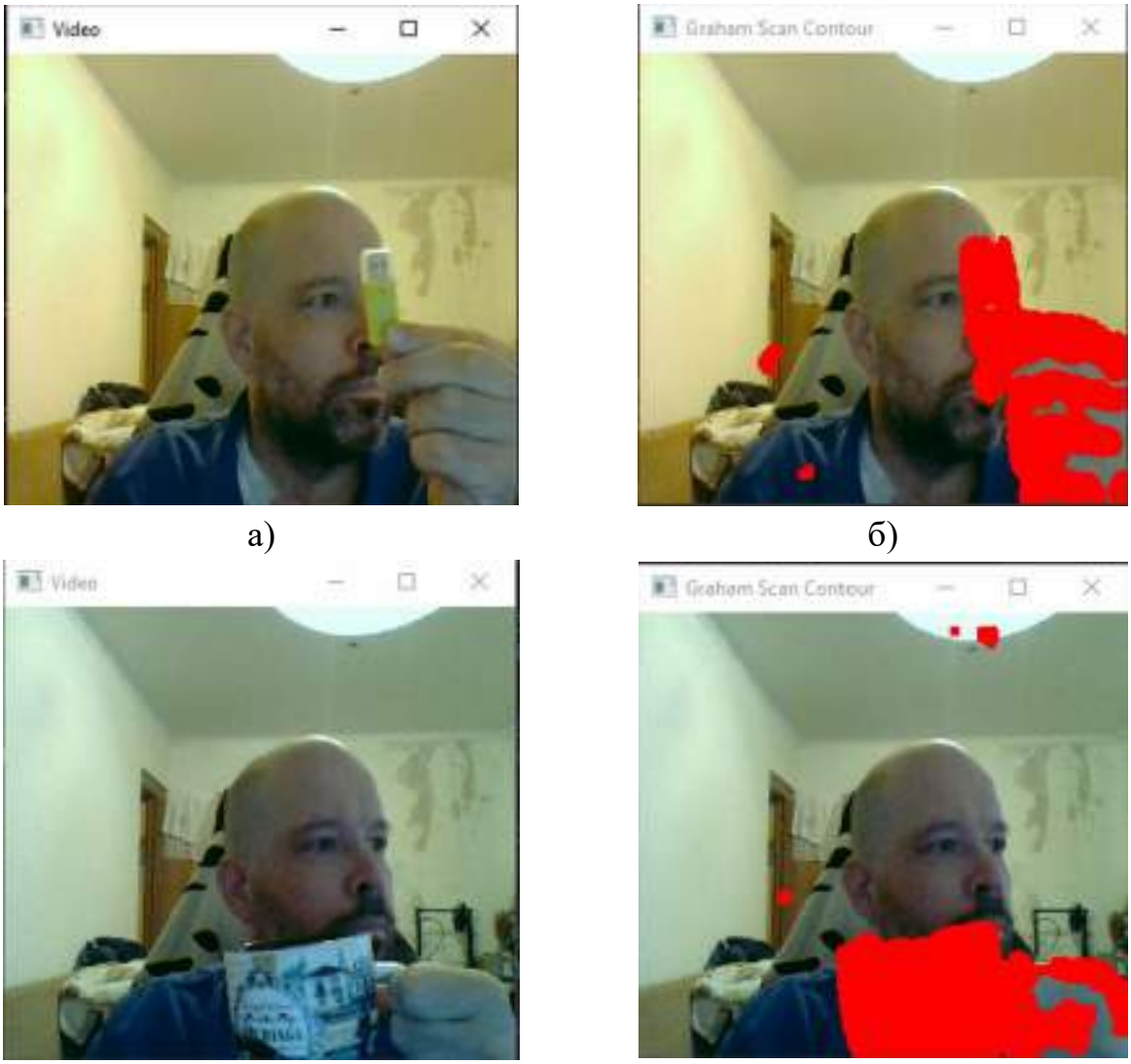
Далі ви викликаєте функцію main(), що означає, що ви запускаєте основний код вашої програми. Передбачається, що функція main() управляє виконанням основної частини вашої програми, такої як відкриття відеопотоку, обробка кадрів, застосування алгоритмів та відображення результатів.

Якщо ваша програма повинна виконуватися як скрипт (тобто прямо з командного рядка), то ви використовуєте умову if __name__ == "__main__":, щоб переконатися, що код усередині цього блоку виконується тільки в тому випадку, якщо скрипт запускається безпосередньо, а не імпортується в інший скрипт як модуль.

Останні рядки коду виконують весь процес: обробку відеопотоку, застосування алгоритму Грехема і відображення результатів на екрані.

```
 return frame
if __name__ == "__main__":
 main()
```

Для проведення досліджень використовувалось наступне апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 2). Програма для реалізації методу оптичного потоку та алгоритму Грехема для пошуку контуру об'єкта в робочій зоні мобільного робота з камери розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Результати роботи програми представлені рисунку 3.6.



а) б)

в) г)

а),в) – Original Video; б), г) – Graham Scan Control

Рисунок 3.6. – Результати роботи програми реалізацій методу оптичного потоку та алгоритму Грехема для пошуку контуру об'єкта в робочій зоні мобільного робота.

Отримані результати (рис.3.6.а,б) роботи методу оптичного потоку та алгоритму Грехема для пошуку контуру об'єкта у робочій зоні мобільного робота показали швидкість обробки кадрів 10.32 кадрів на секунду, що підтверджує ефективність алгоритму у реальному часі. У цьому результати представлені малюнку 3.6.в,г виявили швидкість побудови контуру, що становить 8.17 кадрів на секунду, свідчить про потенційне зниження продуктивності при оконтурюванні великих об'єктів у кадрі. Повний код програми наведено у додатку В.

Метод оптичного потоку та алгоритм Грехема є ефективними підходами для пошуку контуру об'єкта в робочій зоні мобільного робота. Метод оптичного потоку заснований на аналізі руху точок на зображенні, що дозволяє виявляти об'єкти, що рухаються, і їх контури. Алгоритм Грехема, своєю чергою, використовується виділення контуру об'єкта на зображенні шляхом апроксимації контуру ламаною лінією [88-92].

Перевагою методу оптичного потоку є його здатність виявляти об'єкти, що рухаються в реальному часі, що робить його придатним для мобільних роботів, що працюють в динамічних середовищах. Алгоритм Грехема, у свою чергу, забезпечує високу точність виділення контуру об'єкта, що робить його ефективним інструментом для роботів, які потребують точного розпізнавання об'єктів[93,94].

Для успішної реалізації цих методів рекомендується враховувати кілька аспектів. По-перше, необхідно вірно налаштовувати параметри методу оптичного потоку, такі як розмір вікна та поріг для визначення точок руху. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб мінімізувати затримки та забезпечити швидке виконання.

Також рекомендується використовувати оптимізації, такі як паралельні обчислення та використання апаратного прискорення для покращення продуктивності алгоритмів на мобільних роботах. Важливо також враховувати умови освітлення та фон зображення для точного виявлення контурів об'єктів [95,96].

В цілому, метод оптичного потоку та алгоритм Грехема є ефективними інструментами для пошуку контуру об'єкта в робочій зоні мобільного робота. Їхня успішна реалізація може значно підвищити ефективність та точність роботи мобільних роботів у різних сценаріях використання..

3.5 Метод розпізнавання та відстеження об'єкта

Найбільш поширений метод опису зображення складається з трьох окремих каналів кольору, що представляють червоні кольори (R), зелений (G) та синій (B) [97-100]. WIC забезпечує підтримку цих трьох каналів у порядку червоного зеленого синього (RGB) або синьо-зеленого червоного кольору (BGR). Нехай у нас є вхідне зображення потокового відео з камери мобільного робота в кольоровому форматі BGR. Позначимо через $B(x,y)$, $G(x,y)$ і $R(x,y)$ значення пікселя на зображення. Для реалізації розпізнавання та відстеження об'єкта в режимі реального часу на потоковому відео, на першому кроці проведемо перетворення в колірний простір HSV (Hue, Saturation, Value) [101-103]. Математична функція для перетворення з BGR HSV може бути записана наступним чином:

$$(H(x,y),S(x,y),V(x,y)) = f((B(x,y),G(x,y),R(x,y),color\_BGR2HSV) \quad (3.34)$$

$B(x,y)$, $G(x,y)$, $R(x,y)$ - значення пікселя зображення у форматі BGR;

$H(x,y),S(x,y),V(x,y)$ - значення пікселя зображення у форматі HSV;

f - функція перетворює значення зображення BGR в HSV, в даному дослідженні буде використовуватися функція cv2.cvtColor для бібліотеки cv2 мови Python.

Після цього введемо опис порогових значень діапазонів відтінків, насиченості та яскравості для виділення необхідного об'єкта в кадрі.

$$\begin{cases} H_{low} \le x_i \le H_{high} \\ S_{low} \le x_k \le H_{high} \\ V_{low} \le x_q \le V_{high} \end{cases} \quad (3.35)$$

Де: x_i, x_k, x_q - відповідні значення відтінків, насиченості та яскравості для колірного простору HSV (Hue, Saturation, Value), за умови що значення $H_{low}, S_{low}, V_{low} \geq 0$ и $H_{high}, S_{high}, V_{high} \leq 255$.

На отримане зображення в колірний простір HSV проведемо бінаризацію, за умови, що бінаризація виконується з використанням порогових значень для кожного каналу HSV, як описано у виразі 3.35. і створимо маску зображення відповідно до наступного виразу:

$$M(x,y) = \begin{cases} 1, \text{if } H_{low} \leq x_i \leq H_{high}, S_{low} \leq x_k \leq S_{high}, V_{low} \leq x_q \leq V_{high} \\ 0 \end{cases} \quad (3.36)$$

де: $M(x,y)$ - маска з координатами (x,y);

x_i, x_k, x_q - відповідні значення відтінків, насиченості та яскравості для колірного простору HSV (Hue, Saturation, Value), за умов що значення $H_{low}, S_{low}, V_{low} \geq 0$ и $H_{high}, S_{high}, V_{high} \leq 255$.

По отриманій масці необхідно визначити послідовність знаходження контурів. Нехай M - бінарна маска, що є зображенням після бінаризації. Пікселі в цій масці приймають значення 0 або 1 де 1 позначає об'єкт, а 0 - фон. Спочатку визначимо операцію виділення контуру об'єкта. Нехай C - контур об'єкта на масці M, тоді операцію виділення можна подати у такому вигляді:

$$C = \{(x,y) | M(x,y) = 1\} \quad (3.37)$$

де: (x,y)- координати пікселя на масці,

$M(x,y)$ - значення пікселя.

Наступним кроком використовуючи виділені контури, отримуємо список контурів, де кожен контур представлений безліччю точок.

$$C_{list} = \{C_1, C_2, ..., C_m\} \quad (3.38)$$

Де: C_{list} - список контурів, де кожен контур представлений безліччю точок;

C_m - m-й контур, представлений як безліч точок.

Таким чином, послідовність математичних визначень для знаходження контурів маски виглядає наступним чином:

$$M \to C \to C_{list} \to \{C_1, C_2, ..., C_m\} \to n_j \tag{3.39}$$

Де: M - бінарна маска; C - контур об'єкта на масці; C_{list} - список контурів; C_m - m-й контур зі списку контурів; n_j - кількість точок у j-м контуру.

Для перевірки правильності міркувань розробимо програму мовою Python в середовищі розробки PyCharm 2022.2.3 (Professional Edition). Приклад реалізації з описом програмного коду представлений нижче.

```
import cv2
import numpy as np
import time
```

Підключення бібліотек cv2, numpy та time забезпечує доступ до функцій та класів, які дозволяють працювати із зображеннями, виконати математичні операції та працювати з часом відповідно.

cv2: Бібліотека OpenCV надає широкий спектр функцій для роботи із зображеннями та відеопотоками. Вона часто використовується для обробки зображень, аналізу відеопотоків, комп'ютерного зору та машинного навчання.

numpy: Це бібліотека Python для виконання операцій з масивами та матрицями. Вона забезпечує підтримку багатовимірних масивів, а також математичні функції для роботи з ними. У контексті обробки зображень, numpy часто використовується для представлення зображень у вигляді масивів та виконання операцій над ними.

time: Це стандартна бібліотека Python, яка надає функції для роботи з часом. Вона дозволяє вимірювати час, ставити затримки в коді, отримувати поточний час тощо. У контексті обробки відеопотоку, time може використовуватися, наприклад, для вимірювання часу обробки кожного кадру.

def process_frame(frame):

Ця функція process_frame приймає один кадр frame на вхід і виконує деяку обробку цього кадру.

Засікаємо час початку обробки кадру

start_time = time.time()

Цей фрагмент коду засікає час початку обробки кадру, щоб виміряти час, витрачений на обробку кадру.

time.time(): Ця функція повертає поточний час у секундах з початку епохи (зазвичай 1 січня 1970). Виклик time.time() повертає поточний час під час виклику цієї функції.

start_time = time.time(): Цей рядок зберігає поточний час у змінній start_time, щоб засікти час початку обробки кадру.

Це корисно для оцінки продуктивності алгоритмів обробки зображень та визначення, чи виконується обробка кадрів у реальному часі.

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

Цей фрагмент коду виконує перетворення колірного простору з BGR HSV.

cv2.cvtColor(frame, cv2.COLOR_BGR2HSV): Ця функція з бібліотеки OpenCV приймає зображення frame та код колірного простору cv2.COLOR_BGR2HSV як аргументи і повертає зображення в новому колірному просторі HSV (відтінок, насиченість, яскравість).

Колірний простір HSV (Hue, Saturation, Value) - це спосіб представлення кольору, який розділяє колір на три компоненти: відтінок (hue), насиченість (saturation) та яскравість (value) [104-106]. Відтінок є

колірним тоном, насиченість визначає "чистоту" кольору, а яскравість визначає яскравість кольору..

Перетворення з BGR на HSV може бути корисним для виконання операцій аналізу кольорів, сегментації об'єктів за кольором та інших операцій обробки зображень, які краще виконувати в просторі HSV [107].

lower_yellow = np.array([20, 100, 100])

upper_yellow = np.array([30, 255, 255])

Цей фрагмент коду визначає нижній і верхній пороги визначення жовтого кольору в колірному просторі HSV.

lower_yellow = np.array([20, 100, 100]): Це масив NumPy, який представляє нижні граничні значення для відтінку (hue), насиченості (saturation) та яскравості (value) жовтого кольору в колірному просторі HSV. У разі це [20, 100, 100].

upper_yellow = np.array([30, 255, 255]): Це масив NumPy, який представляє верхні граничні значення для відтінку (hue), насиченості (saturation) та яскравості (value) жовтого кольору в колірному просторі HSV. У разі це [30, 255, 255].

Ці значення використовуються для створення маски, яка виділяє області зображення, які відповідають зазначеним діапазонам кольору. Маска може бути використана для виділення жовтих об'єктів або областей зображення.

mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

Цей фрагмент коду використовує функцію cv2.inRange() для створення маски, яка виділяє області зображення, що містять кольори у вказаному діапазоні.

cv2.inRange(hsv, lower_yellow, upper_yellow): Ця функція приймає зображення в колірному просторі HSV (hsv) та нижній і верхній пороги (lower_yellow та upper_yellow) у вигляді масивів NumPy. Вона повертає бінарне зображення (маску), в якому пікселі, що відповідають кольорам у

заданому діапазоні, встановлені у максимальне значення (255), а решта пікселів встановлена у мінімальне значення (0).

Отримана маска може бути використана для виділення лише тих частин зображення, які містять жовті кольори у вказаному діапазоні. Це може бути корисним для подальшої обробки зображень, таких як виділення об'єктів або аналіз кольорів.

contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

Цей фрагмент коду використовує функцію cv2.findContours() для пошуку контурів на бінарному зображенні (масці).

cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_ APPROX_ SIMPLE): Ця функція приймає бінарне зображення (mask) та режим пошуку контурів (cv2.RETR_EXTERNAL) як аргументи. Режим cv2.RETR_EXTERNAL означає, що функція знайде лише зовнішні контури, ігноруючи внутрішні контури. Також визначається метод апроксимації контурів (cv2.CHAIN_APPROX_SIMPLE), який апроксимує контури з використанням методу упаковки.

Функція cv2.findContours() повертає два значення: контури (список усіх знайдених контурів) та ієрархію контурів (зазвичай ми використовуємо лише контури, тому друге значення ігнорується, вказується знак підкреслення). Кожен контур представлений у вигляді масиву точок, які утворюють контур об'єкта на зображенні.

```
if len(contours) > 0:
    largest_contour = max(contours, key=cv2.contourArea)
    M = cv2.moments(largest_contour)
    if M["m00"] != 0:
       cx = int(M["m10"] / M["m00"])
       cy = int(M["m01"] / M["m00"])
       cv2.drawMarker(frame, (cx, cy), (0, 0, 255),
cv2.MARKER_CROSS, markerSize=40, thickness=3)
```

Цей фрагмент коду виконує такі дії:

- перевіряє, що було знайдено хоча б один контур (if len(contours) > 0:).

- знаходить найбільший контур серед усіх знайдених контурів (largest_contour = max(contours, key=cv2.contourArea)).

- обчислює моменти контуру за допомогою функції cv2.moments(). Якщо момент нульового порядку (M["m00"]) не дорівнює нулю (що означає, що знайдено контур і має площа), то обчислюються координати центру мас контуру (cx і cy).

- малює хрест на зображенні в центрі мас контуру за допомогою функції cv2.drawMarker().

Цей фрагмент коду призначений для візуалізації на зображенні центру мас найбільшого контуру, що може бути корисним для відстеження об'єктів на відео.

Засікаємо час закінчення обробки кадру

end_time = time.time()

Цей фрагмент коду засікає час закінчення кадру, щоб виміряти час, витрачений на обробку кадру.

time.time(): Ця функція повертає поточний час у секундах з початку епохи (зазвичай 1 січня 1970). Виклик time.time() повертає поточний час під час виклику цієї функції.

end_time = time.time(): Цей рядок зберігає поточний час у змінній end_time, щоб засікти час закінчення обробки кадру.

Це корисно для оцінки продуктивності алгоритмів обробки зображень та визначення, чи виконується обробка кадрів у реальному часі.

Розраховуємо часові інтервали

processing_time = end_time - start_time

detection_speed = 1 / processing_time if processing_time > 0 else 0

Цей фрагмент коду розраховує часові інтервали та швидкість виявлення об'єктів на основі часу, витраченого на обробку кадру.

processing_time = end_time - start_time: Цей рядок обчислює час, витрачений на обробку кадру, віднімаючи час початку обробки кадру (start_time) з часу закінчення обробки кадру (end_time).

detection_speed = 1 / processing_time if processing_time > 0 else 0: Цей рядок визначає швидкість виявлення об'єктів, використовуючи час обробки кадру. Якщо час обробки кадру більший за нуль, то обчислюється швидкість як зворотне значення часу обробки кадру (оскільки швидкість - це кількість кадрів на секунду). Якщо час обробки кадру дорівнює нулю (наприклад, якщо обробка кадру зайняла менше однієї секунди), швидкість виявлення встановлюється в нуль.

Ці часові інтервали та швидкість можуть бути корисними для аналізу продуктивності та ефективності вашого алгоритму виявлення об'єктів на відео.

```
# Виводимо результати
    print(f"Processing Time: {processing_time:.4f} seconds")
    print(f"Detection Speed: {detection_speed:.2f} FPS")
```

Цей фрагмент коду виводить результати обробки кадру, включаючи час обробки кадру та швидкість виявлення об'єктів на екрані.

print(f"Processing Time: {processing_time:.4f} seconds"): Цей рядок виводить час обробки кадру з точністю до чотирьох знаків після коми у форматі рядка. Час обробки кадру вимірюється у секундах.

print(f"Detection Speed: {detection_speed:.2f} FPS"): Цей рядок виводить швидкість виявлення об'єктів у кадрах за секунду (FPS - frames per second). Швидкість виявлення виражається як кількість кадрів, оброблених за секунду, з точністю до двох знаків після коми.

```
# Відображення маски у вікні
    cv2.imshow('Color Mask', mask)
```

Цей фрагмент коду відображає маску в новому вікні під назвою "Color Mask".

cv2.imshow('Color Mask', mask): Ця функція відображає зображення mask у новому вікні під назвою "Color Mask". mask – це бінарне зображення (маска), яке було створено раніше за допомогою функції cv2.inRange ().

Відображення маски може бути корисним для візуальної оцінки того, які області зображення були виділені в результаті роботи алгоритму обробки зображень, наприклад для перевірки правильності налаштування параметрів колірного фільтра.

```
# Відображення кадру з прицілом
    cv2.imshow('Video Stream', frame)
```

Цей фрагмент коду відображає кадр із прицілом (центром мас найбільшого контуру) у новому вікні під назвою "Video Stream".

cv2.imshow('Video Stream', frame): Ця функція відображає зображення frame у новому вікні під назвою "Video Stream". frame - це оригінальний кадр із відеопотоку, на якому був намальований приціл.

```
# Запуск відеопотоку з камери
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    process_frame(frame)
```

Цей код відкриває відеопотік із камери (зазвичай із пристрою з індексом 0), читає кадри з цього відеопотоку в циклі і передає кожен кадр функції process_frame() для обробки.

cv2.VideoCapture(0): Створює об'єкт відеозахоплення, який дозволяє читати кадри з камери. Аргумент 0 вказує на індекс пристрою захоплення (зазвичай це індекс вбудованої камери).

cap.read(): Читає наступний кадр із відеопотоку. Повертає два значення: прапор ret, який свідчить про успішність читання кадру, і сам кадр frame.

process_frame(frame): Викликає функцію process_frame() для обробки кожного кадру. Ця функція має бути визначена раніше та виконувати необхідні операції обробки кадру

```
# Вихід із циклу при натисканні клавіші 'q' або 'Esc'
    key = cv2.waitKey(1)
    if key == ord('q') or key == 27:  # 27 - код клавіші 'Esc'
        break
```

Цей код дозволяє вийти з циклу, коли користувач натискає 'q' або 'Esc'.

cv2.waitKey(1): Ця функція очікує натискання клавіші протягом зазначеної кількості мілісекунд (у даному випадку 1 мілісекунд). Якщо за вказаний час натиснуто клавішу, вона повертає код цієї клавіші. Якщо жодна кнопка не була натиснута, вона повертає -1.

ord('q'): Ця функція повертає числове значення Unicode для символу 'q'. У цьому випадку використовується для перевірки натискання кнопки 'q'.

key == ord('q') or key == 27: Ця умова перевіряє, чи була натиснута клавіша 'q' або клавіша 'Esc'. Якщо було натиснуто одну з цих клавіш, умова стає істинною і цикл завершується за допомогою break.

```
# Визволення ресурсів
cap.release()
cv2.destroyAllWindows()
```

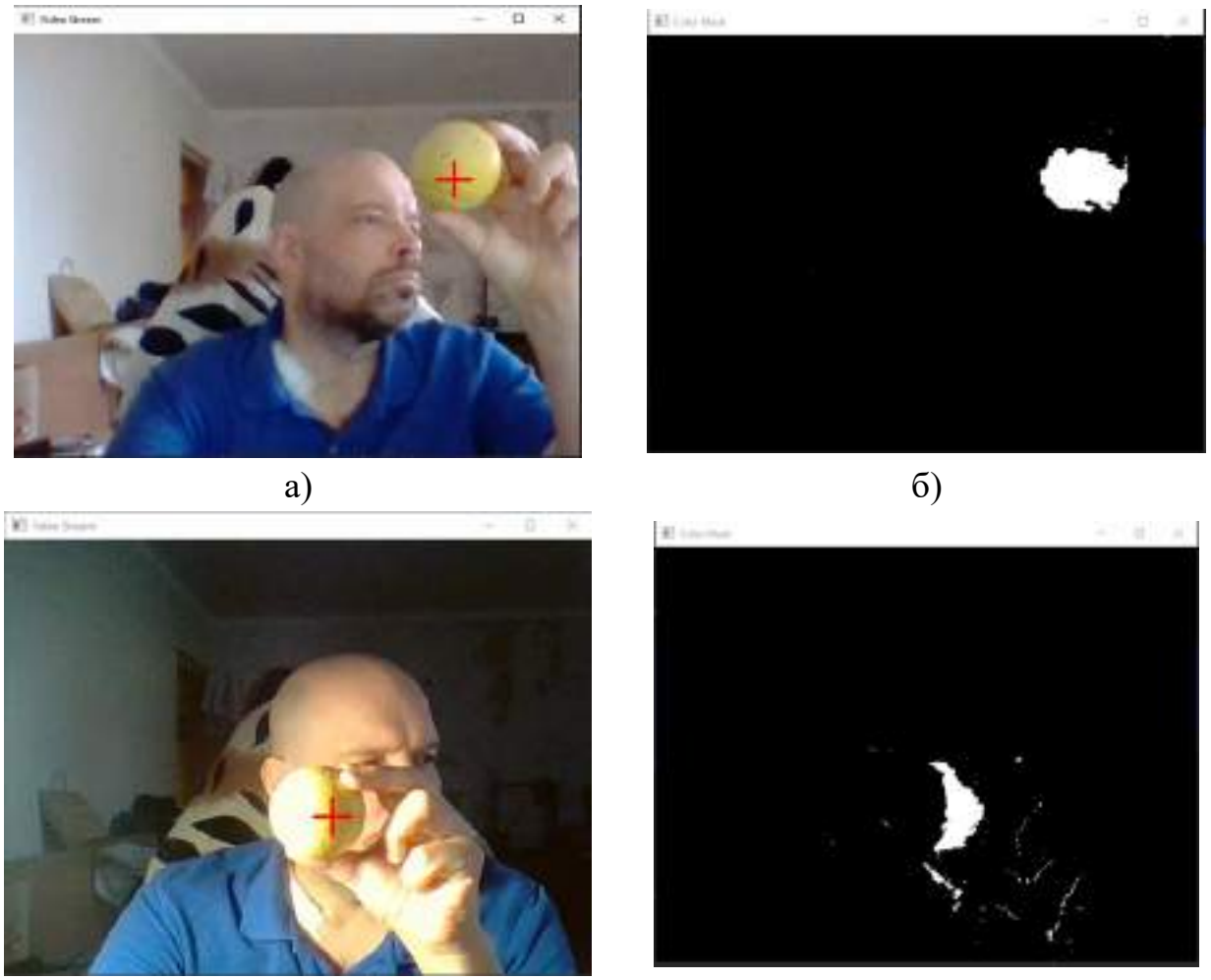
Цей код звільняє ресурси, пов'язані з об'єктом відеозахоплення cap, та закриває всі відкриті вікна OpenCV.

cap.release(): Цей метод звільняє ресурси, пов'язані з об'єктом відеозахоплення cap, такі як камера або відеофайл. Це важливо зробити, коли робота з камерою або відеофайлом завершується, щоб інші програми могли використовувати ці ресурси.

cv2.destroyAllWindows(): Ця функція закриває всі вікна OpenCV, які були відкриті під час виконання програми. Це важливо зробити в кінці програми, щоб звільнити ресурси та уникнути "зависання" вікон.

Для проведення досліджень використовувалося наступнк апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 2). Програма розпізнавання та відстеження об'єкта в

режимі реального часу розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Об'єкт для стеження являє собою круглий об'єкт з наступними кодами в колірному просторі: HSB (48,39,82), RGB(208,192,126), CMYK (18,21,56,3), Web # d0c07e. Отримані результати розпізнавання та відстеження об'єкта в режимі реального часу у робочій зоні мобільного робота представлені на рисунку 3.7.



а) б)

в) г)

а), в) – кадр потокового відео з об'єктом; б),г) – маска об'єкта

Рисунок 3.7 – Результати розпізнавання та відстеження об'єкта в режимі реального часу у робочій зоні мобільного робота

У ході проведення експериментів з розпізнавання та відстеження об'єкта в режимі реального часу було отримано такі показники:

- processing time цей показник вимірює час, витрачений обробку кожного кадру з відеопотоку. Він дозволяє оцінити, наскільки швидко

алгоритм може аналізувати та обробляти вхідні дані. У ході проведеного експерименту він коливався в межах від 0.0010 до 0.0020 секунди.

- detection speed Цей параметр вимірює кількість виявлених об'єктів за секунду (FPS - кадри за секунду). Висока швидкість виявлення важлива для забезпечення оперативності системи в реальному часі, особливо при роботі з об'єктами, що швидко рухаються, або в динамічних сценах. У ході проведеного експерименту він коливався в межах від 501.47 до 1037,42 FPS. В основному 501.47 FPS було отримано при різкому переміщенні об'єкта всередині кадру, при повільному переміщенні FPS прагнув 1037,42. Повний приклад коду програми представлений у додатку Г.

Розпізнавання та відстеження об'єкта за рахунок виділення маски в режимі реального часу є важливим завданням для мобільних роботів, особливо у контексті комп'ютерного зору та навігації. Для ефективної реалізації цього завдання рекомендується використовувати сучасні методи обробки зображень та машинного навчання [108-111].

Для успішної реалізації цього підходу в реальному часі рекомендується враховувати швидкість обробки зображень, щоб мінімізувати затримки та забезпечити плавне відстеження об'єкта. Також важливо враховувати зміни освітлення та умов довкілля, щоб забезпечити стабільну роботу алгоритму.

Рекомендується використовувати оптимізації, такі як паралельні обчислення та апаратне прискорення для покращення продуктивності алгоритму. Також важливо проводити регулярне калібрування камери та оновлювати моделі машинного навчання для забезпечення точного та стабільного розпізнавання об'єкта [112-114].

В цілому, розпізнавання та відстеження об'єкта за рахунок виділення маски в режимі реального часу є важливим завданням для мобільних роботів, що потребує комбінації сучасних методів обробки зображень та машинного навчання. Її успішна реалізація може значно підвищити ефективність та точність роботи мобільних роботів у різних сценаріях використання..

3.6 Алгоритм Sobel

Алгоритм Sobel є оператором виявлення меж у зображенні [115-118]. Він використовується для виділення змін інтенсивності зображення, що часто відповідає межам об'єктів. Оператор Sobel заснований на пакунку зображення з двома ядрами (масками) Собеля, одним виділення меж по горизонталі, іншим по вертикалі. Застосування цих ядер дозволяє обчислити похідні інтенсивності по обох осях, і потім поєднує результати, зазвичай використовуючи метод градієнтної апроксимації [119-122].

Опишемо алгоритм Sobel для отримання контуру об'єкта в режимі реального часу у вигляді математичних виразів.

Нехай $I(x, y)$ - це інтенсивність пікселя в координатах (x, y). Тоді градієнт зображення по горизонталі (G_x) і по вертикалі (G_y) може бути виражений такими формулами:

$$G_x = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x+i, y+j) \cdot S_x(i, j) \tag{3.40}$$

$$G_y = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x+i, y+j) \cdot S_y(i, j) \tag{3.41}$$

Де: G_x, G_y - це градієнт зображення по горизонталі та по вертикалі, який обчислюється з використанням оператора Собеля;

$\sum_{i=-1}^{1}$ - операція підсумовування горизонтального напрямку на матриці розміром 3х3 навколо кожного пікселя;

$\sum_{j=-1}^{1}$ - операція підсумовування за вертикальним напрямком на матриці розміром 3х3 навколо кожного пікселя;

$I(x+i, y+j)$ - це інтенсивність пікселя у зображенні в координатах $(x+i, y+j)$;

$S_x(i, j)$ - елемент матриці ядра Собеля по горизонталі у позиції (i, j);

$S_y(i, j)$ - елемент матриці ядра Собеля по вертикальній позиції (i, j)

Наступним кроком необхідно провести обчислення градієнтів G_x, G_y по вертикалі та по горизонталі відповідно:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}; \quad S_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.42}$$

На базі результатів обчислення градієнтів G_x, G_y по вертикалі та по горизонталі, дозволяє зробити «обчислення градієнта» (G) яка є величиною зміни інтенсивності (яскравості) зображення в кожній точці. В алгоритмі Собеля, градієнт обчислюється по горизонталі. G_x і по вертикалі G_y та величина градієнта (G) визначається як евклідова норма цих двох компонентів:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4.43}$$

Це означає, що для кожної точки на зображенні величина градієнта є довжиною вектору градієнта в даній точці. Величина градієнта показує, наскільки швидко змінюється інтенсивність зображення у цій точці та у якому напрямку.

Чим вище значення величини градієнта, тим яскравіше виражені межі об'єктів. Величина градієнта може бути використана для виявлення країв, меж та текстур у зображенні, що робить її корисною в операціях обробки зображень, таких як виділення контурів [123-126].

Далі необхідно обчислити кут нахилу градієнта (θ) - це кут, який вказує напрямок найбільшої зміни інтенсивності (яскравості) у зображенні в кожній точці. У контексті алгоритму Собеля, кут нахилу градієнта

обчислюється з використанням компонентів градієнта по горизонталі. G_x і по вертикалі G_y, та розраховується за формулою:

$$\theta = \arctan(\frac{G_y}{G_x}) \tag{3.44}$$

Кут нахилу градієнта (θ) вимірюється відносно горизонтальної осі і вказує напрямок, в якому змінюється інтенсивність. Наприклад, якщо кут дорівнює 0 градусів, це означає, що найбільша зміна інтенсивності відбувається вздовж горизонтальної осі.

Значення кута нахилу градієнта є важливим параметром, використовуваним у різних алгоритмах обробки зображень, таких як виділення меж, розпізнавання об'єктів та інші завдання комп'ютерного зору.

Таким чином, алгоритм Собеля виділяє межі об'єктів, використовуючи операції згортки з ядрами Собеля по горизонталі та вертикалі. G_x і G_y які можуть бути об'єднані для отримання підсумкового зображення градієнта G. Це зображення градієнта широко використовується для виявлення кордонів у Computer Vision Systems, у тому числі для обробки зображень у реальному часі на мобільних роботах.

Для перевірки правильності міркувань розробимо програму мовою Python в середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації вище описаних математичних виразів.

```
import cv2
import time
```

Цей фрагмент коду імпортує модуль cv2, який є бібліотекою OpenCV для роботи із зображеннями та відео в Python. cv2 надає широкий набір функцій для обробки зображень, включаючи завантаження, збереження, обробку та відображення зображень, а також для роботи з відео.

Також фрагмент коду імпортує модуль time, який надає функції для роботи з часом. В даному випадку модуль time може використовуватися для виміру часу виконання певних ділянок коду або для створення затримок у програмі.

```
# Функція для обробки відеопотоку
def process_video():
```

Цей фрагмент коду визначає початок функції з ім'ям process_video(), яка використовуватиметься для обробки відеопотоку. Всередині цієї функції буде міститися код для читання кадрів із відеопотоку, застосування алгоритмів обробки зображень, відображення результатів та інші дії, пов'язані з обробкою відео.

```
# Відкриття відеопотоку з камери (зазвичай 0 для вбудованої камери)
    cap = cv2.VideoCapture(0)
```

Цей фрагмент коду відповідає за відкриття відеопотоку з камери.

cv2.VideoCapture(0) ініціалізує об'єкт відеопотоку cap, який використовуватиметься для читання кадрів з відеопотоку з пристрою захоплення зображення, зазвичай із вбудованої камери. Аргумент 0 означає індекс пристрою захоплення, де 0 зазвичай відповідає першій доступній камері. Якщо ви маєте кілька камер або інші джерела відео, ви можете вказати відповідний індекс для вибору потрібного пристрою.

```
# Перевірка успішності відкриття відеопотоку
    if not cap.isOpened():
        print("Помилка: Неможливо відкрити камеру.")
        return
```

Цей фрагмент коду перевіряє успішність відкриття відеопотоку.

cap.isOpened() - це метод об'єкта cap, який повертає булеве значення: True, якщо відеопотік був успішно відкритий, і False, якщо відкриття відеопотоку не представлене.

Якщо відеопотік не відкрито, виводиться повідомлення про помилку ("Помилка: Неможливо відкрити камеру."), і виконання програми завершується за допомогою оператора return

Створення вікна для вихідного відеопотоку

cv2.namedWindow("Original Video", cv2.WINDOW_NORMAL)

Функція cv2.namedWindow() використовується для створення вікна із заданим ім'ям. Перший аргумент - це рядок з ім'ям вікна, а другий аргумент (cv2.WINDOW_NORMAL) вказує, що вікно матиме нормальні розміри (може бути змінено користувачем). Якщо вказати cv2.WINDOW_AUTOSIZE, вікно автоматично змінюватиме розміри відповідно до розміру відображуваного зображення.

Створення вікна для виведення результату роботи оператора Sobel

cv2.namedWindow("Sobel Edge Detection", cv2.WINDOW_NORMAL)

Функція cv2.namedWindow() використовується для створення вікна із заданим ім'ям. Перший аргумент - це рядок з ім'ям вікна, а другий аргумент (cv2.WINDOW_NORMAL) вказує, що вікно матиме нормальні розміри (може бути змінено користувачем). Якщо вказати cv2.WINDOW_AUTOSIZE, вікно автоматично змінюватиме розміри відповідно до розміру відображуваного зображення.

start_time = time.time() # Час початку обробки відеопотоку

Цей фрагмент коду записує поточний час у змінну start_time за допомогою функції time.time().

Це використовується для відстеження часу початку обробки відеопотоку. Далі, після завершення обробки відеопотоку, можна буде обчислити загальний час обробки, віднімаючи час початку (start_time) від часу завершення.

while True:

ret, frame = cap.read() # Отримання кадру з відеопотоку

if not ret:

print("Помилка: Неможливо отримати кадр.")

```
        break
```

Фрагмент коду створює нескінченний цикл, який виконується, доки не буде перерваний користувачем (наприклад, натисканням клавіші ESC).

Всередині циклу відбувається отримання кадру з відеопотоку за допомогою cap.read(). Змінна ret містить логічне значення, що показує успішність отримання кадру. Якщо значення ret дорівнює False, це означає, що кадр не отримано, і в цьому випадку виводиться повідомлення про помилку, а цикл переривається за допомогою оператора break.

```
# Перетворення зображення на відтінки сірого
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Функція cv2.cvtColor() використовується для зміни кольору зображення. У цьому випадку першим аргументом передається вихідне кольорове зображення frame, а другим аргументом cv2.COLOR_BGR2GRAY вказується цільовий колірний простір, в даному випадку - відтінки сірого.

Результат перетворення зберігається у змінній gray, яка міститиме відтінки сірого зображення.

```
# Застосування оператора Sobel
    sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    edges = cv2.magnitude(sobel_x, sobel_y)
```

Фрагмент коду виконує оператор Sobel для виявлення меж на відтінках сірого зображення.

Перший виклик cv2.Sobel() обчислює горизонтальні похідні, задаючи dx=1 та dy=0. Це дозволяє виділити вертикальні межі. Результат зберігається у змінній sobel_x.

Другий виклик cv2.Sobel() обчислює вертикальні похідні, задаючи dx=0 та dy=1. Це дозволяє виділити горизонтальні межі. Результат також зберігається у змінній sobel_y.

Потім викликається функція cv2.magnitude(), яка обчислює величину градієнта кожного пікселя по горизонтальним (sobel_x) і вертикальним

(sobel_y) похідним. Результат зберігається у змінній edges, яка є зображенням з виявленими межами.

```
# Відображення вихідного відеопотоку
    cv2.imshow("Original Video", frame)
```

Цей фрагмент коду відображає вихідне відео у вікні під назвою "Original Video". Функція cv2.imshow() використовується для відображення зображення frame, яке є поточним кадром відеопотоку. Таким чином користувач бачить поточне відео в реальному часі в цьому вікні.

```
# Відображення результатів роботи оператора Sobel
    cv2.imshow("Sobel Edge Detection", edges)
```

Цей фрагмент коду відображає результат роботи оператора Sobel у вікні "Sobel Edge Detection". Функція cv2.imshow() використовується для відображення зображення edges, яке містить контури, знайдені за допомогою оператора Sobel. Таким чином, користувач бачить оброблений кадр із контурами, виділеними на зображенні.

```
# Обробка події натискання клавіші ESC для завершення програми
    key = cv2.waitKey(1) & 0xFF
    if key == 27:  # 27 - код клавіші ESC
       break
```

Фрагмент коду обробляє подію натискання клавіші ESC для завершення програми. Функція cv2.waitKey(1) очікує натискання клавіші протягом 1 мілісекунди. Якщо клавіша натиснута та її код дорівнює 27 (код клавіші ESC), то програма виходить із циклу while за допомогою оператора break, що призводить до завершення програми.

```
end_time = time.time() # Час закінчення обробки відеопотоку
```

Цей фрагмент коду записує час закінчення відеопотоку. Функція time.time() повертає поточний час за секунди з початку епохи (зазвичай 1 січня 1970 року).

```
# Обчислення швидкості обробки кадрів
   frame_rate = cap.get(cv2.CAP_PROP_FPS)
```

processing_speed = 1 / ((end_time - start_time) / frame_rate)

Цей фрагмент коду обчислює швидкість обробки кадрів. Спочатку він отримує частоту кадрів відеопотоку з допомогою методу cap.get(cv2.CAP_PROP_FPS). Потім він обчислює час, витрачений на обробку всього потоку відео, розділивши різницю між часом закінчення і часом початку обробки на частоту кадрів. Нарешті, він обчислює швидкість обробки кадрів, розділивши частоту кадрів тимчасово обробки одного кадру.

Виведення результатів розрахунків у термінал

print("Швидкість обробки кадрів: {:.2f} кадрів за секунду".format(frame_rate))

print("Швидкість побудови контуру: {:.2f} кадрів за секунду".format(processing_speed))

Цей фрагмент коду виводить результати розрахунків швидкості обробки кадрів та швидкості побудови контуру в термінал. Він виводить частоту кадрів відеопотоку та швидкість обробки кадрів, а також швидкість побудови контуру, яка була обчислена на основі часу, витраченого на обробку всього відеопотоку. Результати виводяться з двома знаками після коми для кращого читання

Звільнення ресурсів та закриття вікон

cap.release()

cv2.destroyAllWindows()

Фрагмент коду звільняє ресурси, пов'язані з потоком відео, і закриває всі вікна, відкриті в процесі виконання програми. Це важливо для звільнення системних ресурсів після завершення роботи з відеопотоком та запобігання витоку пам'яті.

Виклик функції обробки відеопотоку

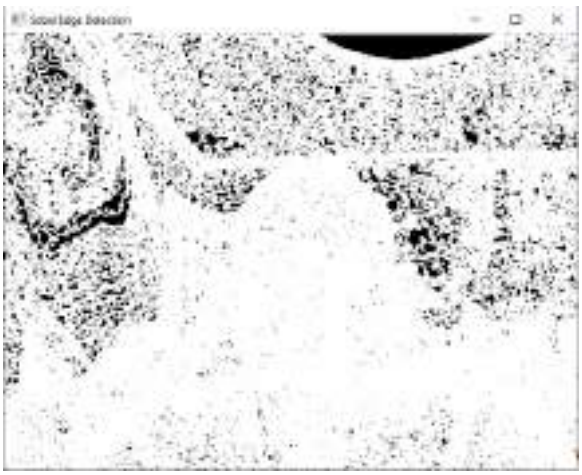
process_video()

Фрагмент коду викликає функцію process_video(), яка містить весь процес обробки відеопотоку: від відкриття відеопотоку з камери до обробки кожного кадру, застосування оператора Sobel для виявлення меж об'єктів на

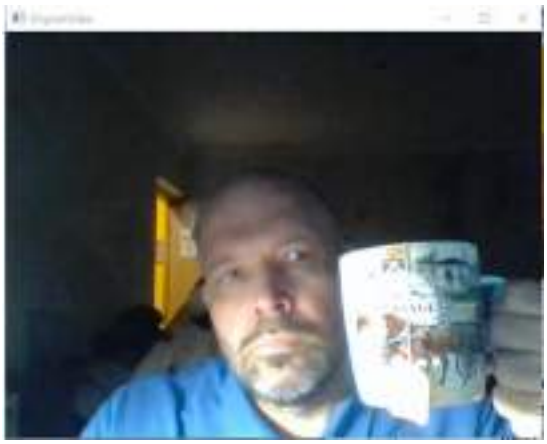
кадрі, відображення вихідного відеопотоку та його обробленої версії з кордонами, а також розрахунку швидкості обробки кадрів та швидкості побудови контуру. Функція process_video() виконує всі ці дії в нескінченному циклі, доки не буде натиснута клавіша ESC для завершення програми.

Для проведення досліджень використовувалось наступне апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 22). Програма для реалізації алгоритму Sobel отримання контуру об'єкта в режимі реального часу з камери розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Результати роботи програми представлені рисунку 3.8

а)



б)



в)

г)

а),в) – Original Video; б), г) – Canny Edge Detection

Рисунок 3.8. – Результати роботи програми реалізації алгоритму Sobel для отримання контуру об'єкта в режимі реального часу.

Отримані результати (рис.3.8.а,б) роботи алгоритму Sobel свідчать про його високу продуктивність при обробці відеопотоку, досягаючи швидкості 30.00 кадрів в секунду. Одночасно швидкість побудови контурів об'єктів становить 1.19 кадрів на секунду, що підтверджує ефективність алгоритму реальному часі. При цьому результати представлені на рисунку 3.8.в, г виявили високу швидкість обробки відеокадрів лише на рівні 30.00 кадрів на секунду, що підкреслює ефективність алгоритму у часі. Однак швидкість побудови контуру, що склала 0.77 кадрів в секунду, вказує на потенційне зниження продуктивності при виконанні додаткових обчислень для формування зображення градієнта. Це визначається що потокове відео отримане у темряві з використанням бічного штучного освітлення. Приклад коду реалізацій наведено у додатку Д.

Алгоритм Sobel є методом який широко використовується для виявлення контурів об'єктів у режимі реального часу для мобільних роботів [127-130]. Він заснований на виділенні меж об'єктів на зображенні шляхом виділення вертикальних та горизонтальних градієнтів яскравості. Перевагою алгоритму Sobel є його відносна простота та висока ефективність, що робить його популярним вибором для мобільних роботів.

Для успішної реалізації алгоритму Sobel для отримання контуру об'єкта в реальному часі рекомендується враховувати такі аспекти [131-135]. По-перше, необхідно правильно вибирати параметри алгоритму, такі як розмір ядра фільтра та поріг для визначення меж. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб мінімізувати затримки та забезпечити швидке виконання.

Також рекомендується використовувати оптимізації, такі як використання паралельних обчислень та апаратного прискорення для покращення продуктивності алгоритму на мобільних роботах. Важливо також враховувати умови освітлення та фон зображення для точного виявлення контурів об'єктів.

В цілому алгоритм Sobel є ефективним і швидким способом отримання контуру об'єкта в режимі реального часу для мобільних роботів. Його простота в реалізації та висока швидкість роботи роблять його чудовим інструментом для завдань комп'ютерного зору на мобільних платформах.

3.7 Метод Лукаса-Канаде

Метод Лукаса-Канаде (Lucas-Kanade) - це класичний алгоритм оптичного потоку в комп'ютерному зорі [136-140]. Він використовується для оцінки руху об'єктів у відеопослідовності, ґрунтуючись на русі яскравих точок (або фіч) між кадрами. Метод Лукаса-Канаде є одним із простих та ефективних методів оптичного потоку, і він знаходить широке застосування у багатьох додатках, таких як відстеження об'єктів, стабілізація відео, оцінка швидкості руху та інші. Однак він має свої обмеження, такі як припущення про локальність руху та неможливість застосування до великих зрушень об'єктів [141-144].

Припустимо, що ми маємо зображення $I(x, y, t)$, де (x, y) - координати пікселя на зображенні, а t - час. Ми хочемо визначити, як кожен піксель переміщається між двома кадрами t и $t + \Delta t$ можна апроксимувати лінійною функцією:

$$I(x, y, t + \Delta t) - I(x, y, t) = \nabla I \cdot \Delta P \tag{3.45}$$

Де: ∇I - градієнт інтенсивності пікселя (вектор-градієнт), що визначається приватними похідними інтенсивності за координатами x та y

ΔP - вектор зміщення пікселя.

Мінімізація помилки [145-147]. Мета полягає в тому, щоб знайти такий вектор зміщення ΔP, який мінімізує помилку між фактичною та передбаченою зміною інтенсивності. Для цього пропонується використати

метод найменших квадратів. Що дозволить мінімізувати суму квадратів помилок E для всіх пікселів на зображенні:

$$E = \sum_{(x,y)} [I(x, y, t + \Delta t) - I(x, y, t) - \nabla I \cdot \Delta P]^2 \quad (3.46)$$

Оскільки модель є лінійною і може бути грубою апроксимацією, метод Лукаса-Канаде використовує ітеративний підхід для уточнення оцінки оптичного потоку. На кожній ітерації k обчислюється вектор зміщення ΔP_k, використовуючи поточну оцінку ΔP_{k-1}, и корегується з урахування остаточної похибки.

З точки зору розробки програми для реалізації методу Лукаса-Канаде мовою Python необхідно врахувати такі параметри, які можуть бути налаштовані для оптимізації продуктивності та точності:

winSize – розмір вікна для обчислення оптичного потоку. Він визначає розмір області, де відбувається пошук оптичного потоку. Більший розмір вікна може покращити точність, але може збільшити обчислювальне навантаження.

maxLevel – максимальний рівень піраміди для багаторівневого обчислення оптичного потоку. Це дозволяє збільшити область пошуку та покращити стійкість алгоритму до великих переміщень об'єктів.

criteria - критерій зупинки ітерацій. Цей параметр дозволяє контролювати кількість ітерацій та зупиняти процес, коли досягається певна умова.

Метод Лукаса-Канаде дозволяє оцінити рух об'єктів на відеопотоці, використовуючи лінійну апроксимацію зміни інтенсивності пікселів між кадрами та ітеративний підхід для уточнення оцінки оптичного потоку. Він заснований на мінімізації помилки між фактичною та передбаченою зміною інтенсивності пікселів, використовуючи метод найменших квадратів [148-150].

Для перевірки правильності міркувань розробимо програму мовою Python в середовищі розробки PyCharm 2022.2.3 (Professional Edition). Наведемо приклад програмної реалізації вище описаних математичних виразів.

```
import cv2
import numpy as np
import time
# Функція обчислення швидкості кадрів
def calculate_fps(prev_time, current_time, frame_count):
    elapsed_time = current_time - prev_time
    fps = frame_count / elapsed_time
    return fps
```

Ця функція, calculate_fps, використовується для обчислення швидкості кадрів (FPS - frames per second) у послідовності відео. Ось що вона робить:

prev_time: Час початку періоду, протягом якого обчислюється швидкість кадрів.

current_time: Поточний час наприкінці періоду, протягом якого обчислюється швидкість кадрів.

frame_count: Кількість кадрів, опрацьованих за цей період.

Функція спочатку обчислює час, що минув elapsed_time шляхом віднімання prev_time з current_time. Потім вона обчислює швидкість кадрів fps шляхом розподілу кількості оброблених кадрів frame_count на час elapsed_time.

Через війну функція повертає значення швидкості кадрів fps. Це значення можна використовувати для моніторингу продуктивності обробки відео та для прийняття відповідних дій у програмі, що базуються на швидкості кадрів. Наприклад, ви можете вирішити збільшити обчислювальні ресурси або змінити алгоритм обробки, якщо швидкість кадрів надто низька.

```
# Функція для обчислення швидкості побудови контуру
def calculate_contour_speed(start_time, end_time, contour_count):
```

```
    elapsed_time = end_time - start_time
    contour_speed = contour_count / elapsed_time
    return contour_speed
```

Ця функція calculate_contour_speed використовується для обчислення швидкості побудови контуру в відеопослідовності. Ось що вона робить:

start_time: Час початку періоду, протягом якого обчислюється швидкість побудови контуру.

end_time: Час закінчення періоду, протягом якого обчислюється швидкість побудови контуру.

contour_count: Кількість контурів, збудованих за цей період.

Функція спочатку обчислює минулий час elapsed_time шляхом віднімання start_time з end_time. Потім вона обчислює швидкість побудови контуру contour_speed шляхом розподілу кількості збудованих контурів contour_count на минулий час elapsed_time.

У результаті функція повертає значення швидкості побудови контуру contour_speed. Це значення також можна використовувати для моніторингу продуктивності обробки відео та для прийняття відповідних дій у програмі. Наприклад, можна оптимізувати алгоритм побудови контуру або розглянути використання паралельних обчислень для поліпшення швидкості обробки.

```
# Відкриття відеопотоку з камери
cap = cv2.VideoCapture(0)
```

Цей фрагмент коду відповідає за відкриття відеопотоку з камери. Давайте розберемо його:

cv2.VideoCapture(0): Цей рядок створює об'єкт відеозахоплення за допомогою функції cv2.VideoCapture(). Як аргумент передається індекс пристрою захоплення, в даному випадку 0, що означає використання першої доступної камери. Якщо у вас є кілька камер, можна вказати індекс відповідної камери (наприклад, 1 для другої камери).

Після виконання цього рядка об'єкт cap буде відеозахоплення з камери. Ви можете використовувати його для подальшого читання кадрів із відеопотоку за допомогою методу read().

```
# Створення вікна для виведення відео
cv2.namedWindow("Video Stream")
```

Цей фрагмент коду створює вікно із заданим ім'ям для виведення відеопотоку. Давайте розберемо його:

cv2.namedWindow("Video Stream"): Цей рядок створює вікно з ім'ям "Video Stream" за допомогою функції cv2.namedWindow(). У цьому вікні буде відображатися відеопотік, який ми будемо обробляти. Якщо вікно з такою назвою вже існує, воно буде створено заново. Створення вікна для виведення відеопотоку дозволяє візуалізувати результати обробки відео в реальному часі. Ми можемо використовувати це вікно для відображення кадрів, а також для відображення візуалізації результату обробки, якщо це необхідно.

```
# Змінні методу Лукаса-Канаде
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS| cv2. TERM_ CRITERIA_
COUNT, 10, 0.03))
```

Цей фрагмент коду створює змінну lk_params, яка містить параметри методу Лукаса-Канаде. Давайте розберемо ці параметри:

winSize=(15, 15): Цей параметр визначає розмір вікна, що використовується для пошуку оптичного потоку. Значення (15, 15) вказує на ширину та висоту вікна в пікселях. Вікно визначає область, де алгоритм шукає зміщення об'єктів між кадрами.

maxLevel=2: Цей параметр визначає максимальний рівень піраміди для багаторівневого аналізу зображень. Багаторівневий аналіз допомагає алгоритму працювати з різними масштабами об'єктів та покращити точність оцінки оптичного потоку. Значення 2 вказує на максимальну кількість рівнів піраміди.

criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03): Цей параметр визначає критерії зупинки ітераційного процесу алгоритму. У цьому випадку використовується комбінація двох критеріїв:

cv2.TERM_CRITERIA_EPS: Зупинка ітерацій, коли досягається необхідна точність (параметр epsilon). Значення 0.03 вказує на потрібну точність.

cv2.TERM_CRITERIA_COUNT: Зупинення ітерацій після заданої кількості ітерацій (параметр max_iter). Значення 10 вказує на максимальну кількість ітерацій.

Ці параметри визначають поведінку алгоритму Лукаса-Канаде для обчислення оптичного потоку між кадрами відеопослідовності. Встановлення їх правильних значень може вплинути на точність та швидкість роботи алгоритму.

Змінні алгоритму Грехема

corners = np.array([], dtype=np.float32)

Цей фрагмент коду створює змінну corners, яка використовуватиметься для зберігання опорних точок, виявлених алгоритмом Грехема (Harris corner detection) чи іншими методами виявлення кутів. Давайте розберемо його:

corners: Це масив NumPy, призначений зберігання координат опорних точок як масиву двовимірних координат (x, y). У цьому випадку corners ініціалізується порожнім масивом.

np.array([], dtype=np.float32): Цей рядок створює масив NumPy, спочатку порожній (без опорних точок). dtype=np.float32 вказує тип даних елементів масиву, який тут встановлено на float32. Цей тип використовується для зберігання координат точок з плаваючою комою.

Пізніше цей масив заповнюватиметься опорними точками, знайденими у кадрах відеопотоку за допомогою алгоритму Грехема або іншими методами виявлення кутів. Ці опорні точки будуть використовуватися для обчислення оптичного потоку методом Лукаса-Канаде або інших завдань комп'ютерного зору, таких як відстеження об'єктів.

```
while True:
    ret, frame = cap.read()
    if not ret:
      break
```

Цей фрагмент коду представляє нескінченний цикл, який читає кадри з відеопотоку, отриманого за допомогою об'єкта cap (відеозахоплення), та перевіряє успішність читання кожного кадру.

Ось як працює цей код:

ret, frame = cap.read(): Цей рядок читає наступний кадр із відеопотоку та присвоює його змінній frame. При цьому також повертається булеве значення ret, яке вказує на успішність читання кадру. Якщо кадр був успішно прочитаний, ret дорівнюватиме True, інакше - False.

if not ret: break: Цей блок коду перевіряє, чи кадр успішно прочитаний з відеопотоку. Якщо змінна ret дорівнює False, що означає невдале читання кадру, цикл переривається з допомогою інструкції break. Це забезпечує вихід із нескінченного циклу, коли відеопотік закінчується або виникає помилка під час читання кадру.

Таким чином, цей код забезпечує безперервне читання кадрів з відеопотоку доти, поки відеопотік доступний і кадри можуть бути успішно прочитані.

```
# Перетворення зображення на відтінки сірого
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Цей фрагмент коду перетворює кольорове зображення на відтінки сірого. Давайте розберемо його:

cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY): Цей рядок використовує функцію cv2.cvtColor() для перетворення зображення frame із кольорового формату BGR (Blue-Green-Red) у відтінки сірого. Другий аргумент, cv2.COLOR_BGR2GRAY, вказує на те, що ми хочемо виконати перетворення на відтінки сірого.

Після виконання цього рядка змінна gray_frame міститиме відтінки сірого версію зображення frame.

Перетворення у відтінки сірого часто виконується перед обробкою зображень, оскільки багато алгоритмів комп'ютерного зору працюють тільки з одноканальними зображеннями, і відтінки сірого є простим способом скоротити обчислювальне навантаження.

```
# Якщо є кути, використовуємо їх у методі Лукаса-Канаді
    if len(corners):
        new_corners, status, _ = cv2.calcOpticalFlowPyrLK(old_gray, gray_frame, corners, None, **lk_params)
        good_new = new_corners[status == 1]
        good_old = corners[status == 1]
```

Цей фрагмент коду перевіряє, чи є опорні точки (кути), і якщо вони є, використовує їх у методі Лукаса-Канаде для обчислення оптичного потоку між двома кадрами.

Давайте розберемо, що відбувається у цьому фрагменті:

if len(corners):: Цей рядок перевіряє довжину масиву corners, що містить опорні точки. Якщо довжина масиву не дорівнює нулю, тобто якщо corners є опорні точки, умова виконується.

new_corners, status, _ = cv2.calcOpticalFlowPyrLK(old_gray, gray_frame, corners, None, **lk_params): Цей рядок використовує функцію cv2.calcOpticalFlowPyrLK() для обчислення оптичного потоку методом. Вона приймає як аргументи попередній кадр old_gray, поточний кадр gray_frame, опорні точки corners з попереднього кадру, а також параметри lk_params, визначені раніше. Результати обчислень зберігаються в змінних new_corners, status та _.

good_new = new_corners[status == 1]: Цей рядок відбирає лише ті нові опорні точки (new_corners), для яких статус status дорівнює 1. Це означає, що оптичний потік для цих точок успішно знайдено.

good_old = corners[status == 1]: Цей рядок відбирає відповідні старі опорні точки для успішно знайдених нових точок. Таким чином, змінна good_old містить старі опорні точки, що відповідають успішно знайденим новим точкам.

Цей фрагмент коду дозволяє використовувати вже знайдені опорні точки в методі Лукаса-Канаде на поточному кадрі, що може допомогти підвищити якість і швидкість роботи алгоритму, особливо при відстеженні об'єктів, що рухаються.

```
# Відображення ліній між старими та новими точками
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel()
        c, d = old.ravel()
        frame = cv2.line(frame, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
```

Цей фрагмент коду відображає лінії між старими та новими точками, які були знайдені методом Лукаса-Канаде. Давайте розберемо його:

for i, (new, old) in enumerate(zip(good_new, good_old)):: Цей цикл проходить з кожної пари нових і старих точок. Функція zip() об'єднує good_new і good_old у пари, щоб ітеруватися за ними одночасно. Функція enumerate() додає індекс до кожної пари, щоб ми могли відстежувати номер поточної пари точок.

a, b = new.ravel() і c, d = old.ravel(): Ці рядки розпаковують координати нової та старої точок з їх масивів і надають їх змінним a, b і c, d відповідно. Метод ravel() використовується для перетворення двовимірного масиву точок в одновимірний масив координат.

frame = cv2.line(frame, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2): Цей рядок малює лінію між старою точкою (c, d) та новою точкою (a, b) на кадрі frame. Колір лінії задається кортежем (0, 255, 0), який є зеленим кольором у форматі BGR. Товщина лінії встановлена на 2.

Цей фрагмент коду малює лінії між кожною парою старих та нових точок, знайдених методом Лукаса-Канаде. Це дозволяє візуалізувати оптичний потік та відстежувати рух об'єктів на відеопотоці.

```
# Пошук кутів за допомогою алгоритму Грехема
    corners = cv2.goodFeaturesToTrack(gray_frame, 100, 0.01, 10)
    if corners is not None:
        corners = np.float32(corners)
        for corner in corners:
            x, y = corner.ravel()
            cv2.circle(frame, (int(x), int(y)), 3, (0, 0, 255), -1)
```

Цей фрагмент коду виконує пошук кутів на кадрі зображення з використанням алгоритму Харріса (алгоритму Грехема). Давайте розберемо його:

cv2.goodFeaturesToTrack(gray_frame, 100, 0.01, 10): Цей рядок викликає функцію cv2.goodFeaturesToTrack(), яка виконує пошук кутів на сірому зображенні gray_frame. Перший аргумент - це сіре зображення, у якому проводиться пошук кутів. Другий аргумент – це максимальна кількість кутів, які потрібно знайти. Третій аргумент – це мінімальна якість кута в діапазоні від 0 до 1. Четвертий аргумент – це мінімальна відстань між знайденими кутами.

if corners is not None:: Цей рядок перевіряє, що змінна corners, що містить знайдені кути, не є порожньою. Якщо кути були знайдені, умова виконується.

corners = np.float32(corners): Цей рядок перетворює знайдені кути на тип даних np.float32, щоб їх можна було використовувати далі в алгоритмі Лукаса-Канаде.

for corner in corners:: Цей цикл проходить по кожному знайденому кутку.

x, y = corner.ravel(): Цей рядок розпаковує координати кута з масиву і надає їх змінним x та y.

cv2.circle(frame, (int(x), int(y)), 3, (0, 0, 255), -1): Цей рядок малює круговий маркер на зображенні frame у місці знайденого кута. Радіус кола дорівнює 3 пікселям. Колір кола задається у форматі BGR – червоний (0, 0, 255). Значення -1 вказує на те, щоб коло було заповнене кольором.

cv2.imshow("Video Stream", frame)

Цей фрагмент коду відображає поточний кадр frame у вікні "Video Stream" за допомогою функції cv2.imshow(). Давайте розберемо його:

cv2.imshow("Video Stream", frame): Цей рядок використовує функцію cv2.imshow() для відображення зображення frame у вікні з ім'ям "Video Stream". Перший аргумент - це рядок з ім'ям вікна, в якому відображатиметься зображення. Другий аргумент – це саме зображення, яке потрібно відобразити.

Цей код дозволяє нам у реальному часі переглядати відеопотік та результати обробки, включаючи визначення кутів та відображення оптичного потоку між старими та новими точками.

Оновлення попереднього кадру

old_gray = gray_frame.copy()

Цей фрагмент коду оновлює попередній кадр old_gray, привласнюючи поточне сіре зображення gray_frame.

Давайте розберемо, що відбувається:

old_gray = gray_frame.copy(): Цей рядок створює копію поточного сірого зображення gray_frame і надає її змінній old_gray. Це необхідно для подальшого використання копії поточного кадру як попередній кадр на наступному етапі обробки. Використання копії дозволяє зберегти попередній стан кадру та використовувати його для порівняння з поточним кадром під час обчислення оптичного потоку методом Лукаса-Канаде.

Перевірка натискання клавіші ESC

```
    key = cv2.waitKey(1)
    if key == 27:
        break
```

Цей фрагмент коду перевіряє, чи була натиснута клавіша ESC (код клавіші 27 ASCII) і переривається нескінченний цикл, якщо це так.

Давайте розберемо його:

key = cv2.waitKey(1): Цей рядок очікує натискання клавіші протягом 1 мілісекунди. Функція cv2.waitKey() повертає ціле число, яке представляє код натиснутої кнопки. Якщо жодна клавіша не була натиснута протягом зазначеного часу, повертається -1.

if key == 27:: Цей рядок перевіряє, чи натиснута клавіша ESC. Код клавіші ESC дорівнює 27 ASCII. Якщо так, тобто якщо змінна key дорівнює 27, умова виконується.

break: Ця інструкція переривається нескінченний цикл, якщо натиснуто клавішу ESC. Як тільки цикл переривається, програма виходить із циклу та завершує свою роботу.

Цей фрагмент коду дозволяє користувачам вийти з програми, натиснувши ESC.

```
# Обчислення та виведення в термінал швидкості обробки кадрів
total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
frame_rate = cap.get(cv2.CAP_PROP_FPS)
processing_speed = total_frames / (time.time() - start_time)
print(f"Швидкість обробки кадрів: {processing_speed} кадрів/с")
```

Цей фрагмент коду обчислює та виводить у термінал швидкість обробки кадрів.

Давайте розберемо, що відбувається:

total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT): Цей рядок використовує метод get() об'єкта відеозахоплення (cap) для отримання загальної кількості кадрів у відеопотоці.

frame_rate = cap.get(cv2.CAP_PROP_FPS): Цей рядок використовує метод get() для отримання частоти кадрів (FPS) відеопотоку.

processing_speed = total_frames / (time.time() - start_time): Цей рядок обчислює швидкість обробки кадрів, розділяючи загальну кількість кадрів на час, що минув з початку обробки (start_time), яке було визначено раніше.

print(f"Швидкість обробки кадрів: {processing_speed} кадрів/с"): Цей рядок виводить швидкість обробки кадрів у термінал з використанням f-рядка. Змінна processing_speed містить обчислене значення швидкості обробки, яке виводиться разом з текстом, що пояснює.

Обчислення та виведення в термінал швидкості побудови контуру

contour_speed = calculate_contour_speed(start_time, time.time(), len(corners))

print(f"Швидкість побудови контуру: {contour_speed} точок/с")

Цей фрагмент коду обчислює та виводить у термінал швидкість побудови контуру.

Давайте розберемо, що відбувається:

contour_speed = calculate_contour_speed(start_time, time.time(), len(corners)): Цей рядок викликає функцію calculate_contour_speed() для обчислення швидкості побудови контуру. Як аргументи передаються час початку обробки start_time, поточний час time.time() і кількість опорних точок len(corners).

print(f"Швидкість побудови контуру: {contour_speed} точок/с"): Цей рядок виводить швидкість побудови контуру в термінал з використанням f-рядка. Змінна contour_speed містить обчислене значення швидкості побудови контуру, яке виводиться разом із текстом, що пояснює.

cap.release()

cv2.destroyAllWindows()

Цей фрагмент коду виконує звільнення ресурсів відеозахоплення та закриття всіх вікон OpenCV. Давайте розберемо його:

cap.release(): Цей метод звільняє ресурси, пов'язані з об'єктом відеозахоплення cap. Це важливо зробити наприкінці роботи з відеопотоком, щоб звільнити ресурси та уникнути витоків пам'яті.

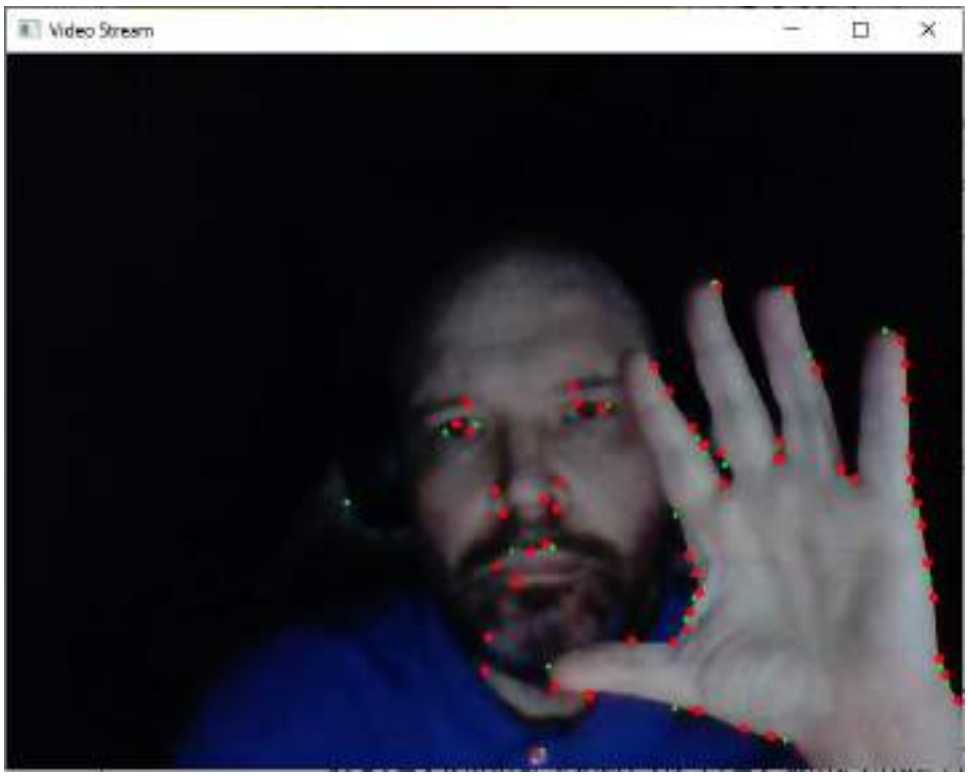
cv2.destroyAllWindows(): Цей метод закриває всі вікна, відкриті за допомогою функції cv2.imshow(). Це важливо зробити в кінці програми, щоб завершити роботу з вікнами OpenCV та уникнути зависання програми.

Цей фрагмент коду забезпечує коректне завершення роботи програми та звільнення всіх ресурсів, пов'язаних із відеозахопленням та вікнами OpenCV.

Метод Лукаса-Канаде є ефективним методом виявлення контурів об'єктів як реального часу для мобільних роботів. Він заснований на обчисленні оптичного потоку, який є векторним полем, що відображає рух точок на зображенні. Перевагою цього є його відносна простота і швидкість роботи, що робить його придатним для застосування на мобільних роботах.

Для успішної реалізації методу Лукаса-Канаде виявлення контурів у часі рекомендується враховувати такі аспекти. По-перше, необхідно правильно вибирати параметри методу, такі як розмір вікна та поріг для визначення точок руху. По-друге, важливо враховувати особливості обробки зображень у реальному часі, щоб мінімізувати затримки та забезпечити плавне виконання.

Для проведення досліджень використовував Далі представлено наступні апаратне забезпечення: CPU Intel Core i5-9300H CPU @ 2.40GHz, RAM 16Gb, GPU NVideo GeForce GTX 1660Ti (Ram 8Gb), Web-camera HD WebCam, OS Windows 10 Pro (Версія 2). Програма для реалізації методу Лукаса-Канаде виявлення контуру об'єкта в режимі реального часу з камери розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Результати роботи програми представлені рисунку 3.9



а)

б)

а) – у темряві (мінімальне освітлення); б) – при штучному освітленні

Рисунок 3.9. – Результати роботи реалізацій методу Лукаса-Канаде виявлення контуру об'єкта в режимі реального часу.

Рекомендується використовувати оптимізації, такі як паралельні обчислення та використання апаратного прискорення для покращення

продуктивності методу на мобільних роботах. Важливо також враховувати довкілля та умови освітлення для обробки зображень для більш точного виявлення контурів.

В цілому, метод Лукаса-Канаде є ефективним і швидким способом виявлення контурів об'єктів у режимі реального часу для мобільних роботів. Його простота в реалізації та висока швидкість роботи роблять його чудовим вибором для завдань комп'ютерного зору на мобільних платформах.

Приклад повного коду реалізацій програми представлений у додатку Е.

4. РЕАЛІЗАЦІЯ МЕТОДІВ І АЛГОРИТМІВ ПОБУДУВАННЯ МАРШРУТІВ РУХУ МОБІЛЬНОЇ РОБОТИ

4.1 Алгоритм знаходження короткого маршруту A*

Алгоритм A* - це модифікація алгоритму Дейкстри, оптимізована для єдиної кінцевої точки. Алгоритм Дейкстри може знаходити шляхи до всіх точок, A* знаходить шлях до однієї точки. Він віддає пріоритет шляхам, які ведуть ближче до мети [151-155].

У найпростішому вигляді роботу алгоритм A * можна представити наступним чином [156-160]. Нехай S - початкова вершина, то тоді $g(S)=0$ відстань від початкової вершини S до самої себе дорівнює 0. Помістимо вершину S у відкритий список O, містить вершини, які ще не були перевірені.

Проведемо пошук оптимального шляху, нехай поки що відкритий список O не порожній, виконуємо такі дії:

- вибираємо вершину n з найменшим значенням оціночної функції $f(n)$.

$$f(n)=g(n)+h(n) \tag{4.1}$$

Де: $g(n)$ - довжина колії від початкової вершини S до вершини n

$h(s)$ - евристична оцінка відстані від вершини n до кінцевої вершини G.

На наступному кроці проводимо перевірку, якщо n - це кінцева вершина G, то алгоритм завершується, інакше для кожного сусіда m вершини n проводимо перевірку чи належить m вже відкритому чи закритому списку вершин. Якщо m вже у відкритому списку, ми

перевіряємо, чи можна покращити його поточну оцінку $f(m)$, а якщо не входить ні до відкритого, ні до закритого списку, ми додаємо його у відкритий список і обчислюємо для нього значення $g(m)$ та $f(m)$.

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import networkx as nx
import matplotlib.pyplot as plt
import random
```

Фрагмент коду імпортує необхідні бібліотеки:

networkx – бібліотека Python для роботи з графами та мережами. Вона надає функції та структури даних для створення, маніпулювання та аналізу складних мереж.

matplotlib.pyplot - це модуль бібліотеки Matplotlib, який надає функції для створення графіків та візуалізації даних у вигляді графічних зображень.

random - модуль Python для створення псевдовипадкових чисел. Він використовується для створення випадкових точок та інших випадкових елементів в алгоритмах, які вимагають випадковості, наприклад, алгоритми планування шляху

```
# Функція для побудови маршруту на графі
def plot_route(G, path):
    pos = nx.spring_layout(G)  # Визначення позицій вузлів
    nx.draw(G, pos, with_labels=True, node_color='lightblue',
node_size=700)  # Малювання графу
    # Малювання маршруту
    edge_labels = {(path[i], path[i+1]): i+1 for i in range(len(path)-1)}
    nx.draw_networkx_edges(G, pos, edgelist=edge_labels.keys(),
edge_color='red', width=2)
```

nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red')

plt.show()

Цей фрагмент коду представляє функцію plot_route, яка використовується для візуалізації маршруту на графі. Далі наведено що робить кожна частина функції:

pos = nx.spring_layout(G): Визначає позиції вузлів на графі з використанням алгоритму розподілу вузлів за сіткою.

nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=700): Малює граф G із зазначеними параметрами: вузли з мітками, кольори та розміру вузлів.

edge_labels = {(path[i], path[i+1]): i+1 for i in range(len(path)-1)}: Створює словник edge_labels де ключами є кортежі ребер (пари вузлів), а значеннями є номери ребер (порядкові номери в списку path).

nx.draw_networkx_edges(G, pos, edgelist=edge_labels.keys(), edge_color='red', width=2): Малює ребра графа G зі списку ребер edgelist, який представлений ключами словника edge_labels. Ребра будуть пофарбовані в червоний колір і матимуть ширину 2.

nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red'): Малює мітки на ребрах відповідно до значень зі словника edge_labels. Мітки будуть пофарбовані у червоний колір.

plt.show(): Показує граф із намальованим маршрутом

Граф для представлення місцевості та шляхів

G = nx.Graph()

Цей фрагмент коду створює порожній граф G, який використовуватиметься для представлення місцевості та шляхів у ній. Граф G створюється за допомогою бібліотеки NetworkX та ініціалізується як порожній граф без будь-яких вузлів чи ребер. Надалі цей граф буде

заповнюватися вузлами (точками) та ребрами (зв'язками між точками), щоб уявити місцевість та шляхи на ній.

```
# Список точок
points = ['A', 'B', 'C', 'D', 'E']
```

Цей фрагмент коду створює список points, що містить імена точок, які будуть представлені як вузли у графі G. У цьому випадку використовуються літери алфавіту ('A', 'B', 'C', 'D', 'E') для позначення кожної точки. Ці точки будуть представлені як вузли у графі, і можуть бути використані для побудови шляхів чи маршрутів між ними.

```
# Додавання рандомних вершин (крапок) та ребер (шляхів) до графа
for point in points:
    G.add_node(point, pos=(random.randint(0, 10), random.randint(0, 10)))  # Генерація випадкових координат для точок
for i in range(len(points)):
    for j in range(i + 1, len(points)):
        weight = random.randint(1, 10)  # Генерація випадкової ваги для ребра
        G.add_edge(points[i], points[j], weight=weight)
```

Цей фрагмент коду додає вершини (точки) та ребра (шляху) у граф G. У циклі for для кожної точки зі списку points генеруються випадкові координати, які зв'язуються з цією точкою у графі. Потім вкладений цикл for створює ребра між парами точок зі списку points, при цьому визначається випадкова вага для кожного ребра. Кожна вершина являє собою точку на місцевості, а кожне ребро є шлях між двома точками.

```
# Побудова найкоротшого шляху за допомогою алгоритму A*
shortest_path = nx.astar_path(G, 'A', 'D', weight='weight')
```

Цей фрагмент коду використовує алгоритм A* для пошуку найкоротшого шляху між двома заданими вершинами A та D у графі G. Алгоритм A* приймає на вхід граф, початкову вершину, кінцеву вершину та

ваги ребер (у даному випадку weight='weight'). Він знаходить оптимальний шлях, враховуючи ваги ребер та евристичну оцінку відстані до мети, і повертає список вершин, що становлять цей шлях.

```
# Висновок найкоротшого шляху
print("Найкоротший шлях:", shortest_path)
```

Цей фрагмент коду виводить найкоротший шлях, знайдений алгоритмом A* у графі G, між початковою вершиною 'A' та кінцевою вершиною 'D'

```
# Візуалізація маршруту
plot_route(G, shortest_path)
```

Цей фрагмент коду візуалізує найкоротший шлях, знайдений алгоритмом A* на графі G, використовуючи функцію plot_route. Кожна вершина графа позначається як точка, а ребра з-поміж них - як лінії. Червоні лінії позначають найкоротший шлях від початкової вершини до кінцевої.

Алгоритм A* є одним з найбільш ефективних методів планування маршрутів для мобільних роботів. Його здатність знаходити оптимальний шлях з урахуванням перешкод та обмежень робить його ідеальним вибором для мобільних роботів, що працюють у різних умовах [161-163].

Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна

Програма для реалізації алгоритму A* розроблена в середовищі PyCharm 2022.2.3 (Professional Edition) мовою Python. Результати роботи програми представлені на рисунку 4.1.

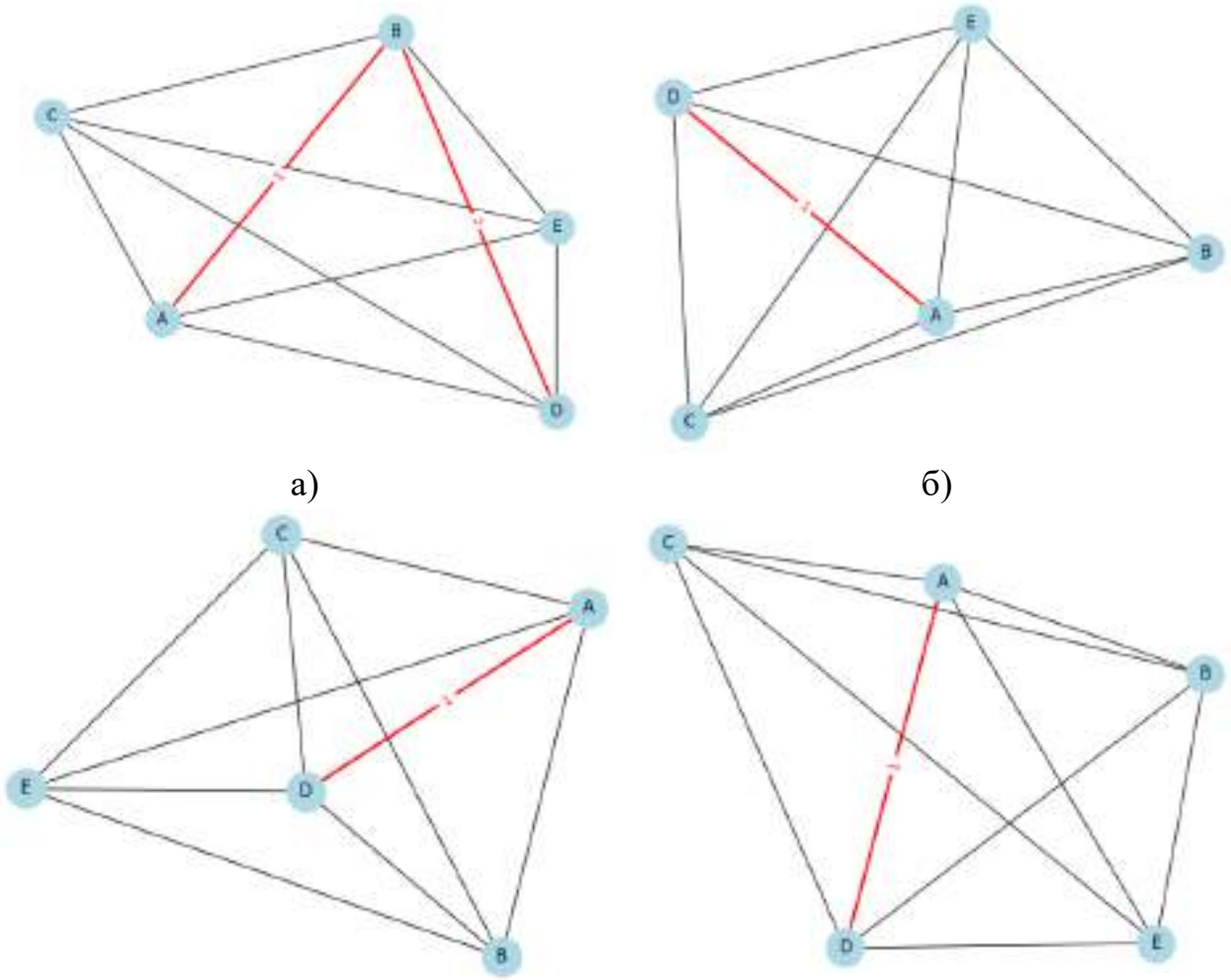


Рисунок 4.1. – Результати роботи реалізацій алгоритму A*.

Перевагою алгоритму A* є його ефективність та оптимальність, що дозволяє знаходити найменший шлях до мети при розумному використанні ресурсів. Це особливо важливо для мобільних роботів, де обмежені обчислювальні ресурси та час виконання [164-166].

Для успішного застосування алгоритму A* для мобільних роботів рекомендується враховувати такі аспекти. По-перше, необхідно ретельно вибирати евристичну функцію, щоб вона була інформативною та ефективною. По-друге, важливо враховувати умови навколишнього середовища, що динамічно змінюються, і оновлювати карту перешкод для актуального планування маршруту.

Також рекомендується використовувати оптимізації, такі як кешування шляхів та зменшення обчислень, щоб покращити продуктивність алгоритму.

Важливо також враховувати особливості робота та його руху під час вибору оптимального шляху [167-169].

В цілому алгоритм A* є потужним інструментом для планування маршрутів мобільних роботів, який забезпечує оптимальне та безпечне переміщення в різних середовищах. Його ефективність та простота в реалізації роблять його ідеальним вибором для широкого спектру завдань у робототехніці.

Приклад реалізацій програми наведено у додатку М.

4.2 Алгоритм BRRT+A*

Позначимо через G - граф, що є картою навколишнього середовища, де вершини - це різні точки, а ребра - зв'язки між ними; V - безліч вершин графа, що є конфігурацією мобільного робота в просторі; E - безліч ребер графа, що є можливими переходами між конфігураціями робота; q_{start} та q_{goal} - початкова та кінцева конфігурація робота відповідно. Під конфігурацією робота розумітимемо унікальне положення та орієнтацію робота в просторі.

Введемо функцію вартості переміщення ($Cost(q_i, q_j)$), яка є математичною моделлю, що описує витрати або витрати при переміщенні від однієї конфігурації робота q_i до іншої конфігурації q_{j}. У контексті планування маршруту для автономного мобільного робота, ця функція відіграє важливу роль у визначенні оптимальності шляху і може бути представлена у вигляді наступного запису:

$$Cost(q_i, q_j) = w_1 \cdot d(q_i, q_j) + w_2 \cdot c_{price}(q_i, q_j) \tag{4.2}$$

де: $d(q_i, q_j)$ - евклідова відстань (або інша метрика відстані) між конфігураціями q_i та q_{j} що представляє довжину переміщення;

$c_{price}(q_i, q_j)$ - функція вартості уникнення перешкод, яка може враховувати наявність перешкод на шляху від q_i до q_j;

w_1 та w_2 - вагові коефіцієнти, які можуть бути налаштовані для керування впливом кожного компонента на загальну вартість.

Запропонована функція (4.2) демонструє, що вартість переміщення залежить як від фізичної відстані, так і від перешкод.

Представимо у вигляді евристичної функції ($H_{price}(q_i, q_{goal})$) математичну модель, що використовується для оцінки приблизної вартості досягнення кінцевої конфігурації q_{goal} з поточної конфігурації q_i. Дане рішення обумовлено тим, що евристична функція повинна бути швидкою в обчисленні і повинна надавати допустиму (нижню) оцінку вартості шляху між двома конфігураціями [170-175]. Таким чином, вона допомагає прискорити пошук шляху, спрямовуючи алгоритм у бік областей, де очікується, що шлях буде оптимальнішим, а сама функція наведена нижче:

$$H_{price}(q_i, q_{goal}) = \left\| q_i - q_{goal} \right\| \tag{4.3}$$

де: $\left\| q_1 - q_{goal} \right\|$ - позначає евклідову відстань між точками у просторі конфігурацій. Це надає приблизну оцінку того, як далеко поточна конфігурація відхилена від кінцевої мети. Одним з обмежень евристичних функцій ($H_{price}(q_i, q_{goal})$) щоб вона була допустимою оцінкою і не завищувала вартість шляху, щоб гарантувати коректність роботи алгоритму A*.

Уявимо через $f(q_i)$ комбінована функція вартості для вершини при використанні алгоритму A* поєднує вартість шляху від початкової вершини до поточної вершини $(g(q_i))$ та евристичну оцінку вартості від поточної вершини до кінцевої цільової вершини $(h(q_i))$, математичне подання даної функцій подано у вираз 4.4.

$$f(q_i) = g(q_i) + h(q_i) \qquad (4.4)$$

де: $(g(q_i))$ - вартість шляху від початкової вершини до поточної вершини. Зазвичай є акумульованою вартістю переміщення від початкової вершини до поточної вершини. У ході виконання алгоритму ця вартість оновлюється в міру розширення графа.

$(h(q_i))$ - евристична оцінка вартості від поточної вершини до кінцевої цільової вершини, при обмеженні вартості досягнення цільової вершини з поточної вершини, і, як правило, вона являє собою наближену (допустиму) оцінку, що не завищує реальну вартість.

Вираз (4.4) дозволяє вибирати вершини для розширення спочатку ті, що мають нижчу комбіновану вартість $f(q_i)$ це забезпечує напрямок пошуку у бік найменш витратних шляхів і прискорює збіжність алгоритму.

Застосування алгоритму A* дозволяє мінімізувати сумарну вартість $f(q_i)$, що означає знаходження оптимального шляху від початкової вершини до цільової вершини.

Наступним кроком опишемо ініціалізацію планування маршруту для робота включає створення двох дерев T_{start} та T_{goal} (для початкової та кінцевої конфігурацій) та визначення безлічі вершин V, представляє простір конфігурацій робота.

Безліч вершин V є простір конфігурацій, який може займати робот, тобто кожна вершина $\upsilon \in V$ являє собою унікальну конфігурацію робота в просторі. Ця множина може бути дискретизована, якщо простір конфігурацій безперервний, та як приклад, складатися з сітки точок.

Дерево T_{start} ініціалізується однією вершиною, що відповідає початковій конфігурації q_{start}, і може бути представлена як:

$$T_{start} = \{\upsilon_{start}\} \qquad (4.5)$$

Де υ_{start} - вершина, відповідна q_{start}.

Дерево T_{goal} ініціалізується однією вершиною, що відповідає кінцевій конфігурації q_{goal}, і може бути представлена як:

$$T_{goal} = \{\upsilon_{goal}\} \tag{4.6}$$

Де: υ_{goal} - вершина, що відповідає кінцевій конфігурації q_{goal}

Для більш конкретного опису того, як встановлюються початкові значення дерев T_{start} и T_{goal} за допомогою операції додавання вершин, уявимо, що кожна вершина в дереві є об'єктом з певними характеристиками, такими як:

$$Вершина : \upsilon = (q, батько) \tag{4.7}$$

де: q - конфігурація робота, що відповідає даній вершині;

батько - посилання на вершину-батька.

Тоді операція додавання вершини до T_{start} дерева може бути математично описана наступним чином:

$$T_{start}.AddNode(\upsilon_{start}) \tag{4.8}$$

де: $\upsilon_{start} = (q_{start}, null)$, тобто початкова вершина υ_{start} містить початкову конфігурацію q_{start}, і вона не має батька, тому що це початкова вершина.

Додавання вершини в T_{goal} дерево може бути математично описано наступним чином:

$$T_{goal}.AddNode(\upsilon_{goal}) \tag{4.9}$$

де: $\upsilon_{goal} = (q_{goal}, null)$, тобто кінцева вершина υ_{goal} містить кінцеву конфігурацію q_{goal}, і в неї немає батька, тому що це кінцева вершина.

Таким чином, операція додавання вершини (4.8-4.9) створює об'єкт вершини із зазначеними характеристиками та додає його у відповідне дерево. Батьківське посилання на *null* вказує на те, що це початкова або кінцева вершина, і в неї немає батька в даному контексті.

Цикл пошуку та вибору вершини в алгоритмі BRRT є ітеративним процесом, в якому алгоритм прагне розширити дерева T_{start} та T_{goal} у бік випадкової вершини, що належить простору конфігурацій робота. Математично це може бути представлено як вибір випадкової конфігурації з простору конфігурацій $q_{rand} \in V$.

Знайдемо найближчі вершини у кожному дереві для поточної випадкової конфігурації q_{rand}, яку можна представити у вигляді наступного запису:

$$q_{near\_start} = \arg\min_{q_i \in T_{start}} C_{price}(q_i q_{rand}) \quad (4.10)$$

$$q_{near\_goal} = \arg\min_{q_i \in T_{goal}} C_{price}(q_i q_{rand}) \quad (4.11)$$

де: q_{near_start} - це найближча вершина у дереві T_{start} до випадкової конфігурації q_{rand}, знайдена внаслідок мінімізації функції вартості;

q_{near_goal} - це найближча вершина у дереві T_{goal} до випадкової конфігурації q_{rand}, знайдена внаслідок мінімізації функції вартості;

T_{start} - являє собою дерево, початкова вершина якого відповідає початковій конфігурації q_{start}, а кожна наступна вершина додається в міру розширення дерева у бік випадкових конфігурацій;

T_{goal} - є дерево, кінцева вершина якого відповідає початковій конфігурації q_{start};

$C_{price}(q_i, q_{rand})$ - функція вартості вимірює відстань між вершиною q_i та поточною випадковою конфігурацією q_{rand}.

Проведемо розширення дерева у напрямок q_{rand}, для цього пропонується використовувати функцію *Extend* яка виконує роль розширення поточної вершини у бік іншої конфігурації. Вона використовується для генерації нової вершини в дереві, яка близька до випадкової конфігурації, її можна представити в наступному вигляді:

$$q_{new\_start} = Extend(q_{near\_start}, q_{rand}) \quad (4.12)$$

$$q_{new\_goal} = Extend(q_{near\_goal}, q_{rand}) \quad (4.13)$$

де: q_{new_start}, q_{new_goal} - нова вершина, отримана внаслідок розширення дерев T_{start} та T_{goal} відповідно;

q_{near_start} - поточна найближча вершина у дереві T_{start} до випадкової конфігурації;

q_{near_goal} - поточна найближча вершина у дереві T_{goal} до випадкової конфігурації;

q_{rand} - випадкова конфігурація, до якої прагнемо розширити поточну вершину.

Наступною дією необхідно провести перевірку, чи стикається нова вершина з перешкодами в навколишньому середовищі. Перевірка на зіткнення включає оцінку, наскільки нова вершина q_{new_start} або q_{new_goal} відповідає безпечному розташуванню у навколишньому середовищі. Припустимо, ми маємо функцію зіткнення $CollisionFree(q)$, яка приймає на вхід конфігурацію робота і повертає булеве значення: true, якщо конфігурація вільна від зіткнень, і false в іншому випадку.

$$CollisionFree(q_{new}) \quad (4.14)$$

де: q_{new} - нова вершина, отримана внаслідок розширення поточної вершини у бік випадкової конфігурації.

Перевіримо, чи не стикається нова вершина з перешкодами у навколишньому середовищі, якщо зіткнення немає, додаємо нову вершину у відповідне дерево:

$$T_{start}.AddNode(q_{new\_start}) \quad (4.15)$$

$$T_{goal}.AddNode(q_{new\_goal}) \quad (4.16)$$

Де: T_{start} и T_{goal} - дерево, що представляє безліч вершин, що описують простір конфігурацій робота, починаючи від початкової конфігурації та до кінцевої конфігурацій q_{goal}, відповідно;

$AddNode$ - функція додавання нової вершини q_{new_start} и q_{new_goal} у дерево T_{start} та T_{goal} відповідно;

q_{new_start} та q_{new_goal} - являє собою конфігурацію робота, отриману внаслідок розширення поточної найближчої вершини у напрямку випадкової конфігурації.

Проводимо перевірку з'єднання, тобто чи з'єднується нова вершина дерева T_{start} та T_{goal} для цього використовуємо наступний вираз:

$$Connected = ChekConnection(T_{start}, T_{goal}, q_{new\_start}, q_{new\_goal}) \quad (4.17)$$

де: $ChekConnection$ - функція, яка перевірить наявність з'єднання між деревами T_{start} и T_{goal}, повертає булеве значення: $true$ якщо знайдено з'єднання між деревами, $false$ в іншому випадку.

Ця функція оцінює, чи можна з'єднати дерево T_{start} та дерево T_{goal} через нові вершини q_{new_start} и q_{new_goal}. Результат $Connected$ вказує на успішне

з'єднання або його відсутність, що впливає на подальший перебіг алгоритму пошуку шляху.

Для формування маршруту та отримання оптимального шляху з використанням алгоритму A* на об'єднаному графі. Позначимо через G_{start} та G_{goal}, об'єднані графи з дерев T_{start} та T_{goal} відповідно, до загального графа G_{total}. Об'єднання відбувається за допомогою додавання ребра між найближчими вершинами в деревах T_{start} та T_{goal}. Формування графа стану G_{total} представляє простір станів, що включає вершини з обох дерев і ребра між ними. Для кожного ребра у графі G_{total} визначте вартість переміщення між відповідними вершинами, як відстань між вершинами в просторі конфігурацій робота.

Застосуємо алгоритм A* до графа G_{total} для пошуку оптимального шляху від початкової конфігурації q_{start} до кінцевої конфігурації q_{goal}. Алгоритм A* використовує евристичну функцію для ефективного пошуку шляху, враховуючи як вартість до вершини (що вже пройдений шлях), так і евристичну оцінку від поточної вершини до цільової. Математичне представлення алгоритму A* та вилучення шляху можна подати такими виразами:

$$f(n) = g(n) - h(n) \tag{4.18}$$

де: $g(n)$ - вартість шляху від початкової вершини до вершини n

$h(n)$ - евристична оцінка вартості від вершини n до цільової вершини.

Для отримання оптимального шляху, представимо послідовність вершин P, як:

$$P = \{\upsilon_1, \upsilon_2, ..., \upsilon_k\} \tag{4.19}$$

де: υ_i - вершина у послідовності P являє собою стан простору конфігурацій робота на певному етапі руху оптимальним шляхом;

k - індекс вказує на довжину оптимального шляху та кількість етапів, які необхідно пройти від початкової конфігурації до кінцевої.

Шлях вилучається у зворотному порядку, такий порядок забезпечує правильну послідовність переміщень для робота від початкової точки до цільової:

$$q_{start}, q_i, q_{i-1}, ..., q_{goal} \tag{4.20}$$

Де: q_{start} - початкова вершина колії;

q_i и q_{i-1} - представляє конфігурацію робота в момент часу i;

q_{goal} - кінцева вершина колії.

Таким чином, об'єднаний граф, формування маршруту та отримання оптимального шляху з використанням алгоритму A* на цьому графі дозволяють знайти ефективний маршрут для руху робота від початкової конфігурації до цільової [176-180].

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import time
```

Цей фрагмент коду використовує декілька бібліотек:

numpy (import numpy as np) - для роботи з масивами і матрицями.

matplotlib.pyplot (import matplotlib.pyplot as plt) – для візуалізації даних, таких як графіки та діаграми.

networkx (import networkx as nx) - для роботи з графами, включаючи створення, аналіз та візуалізацію.

time (import time) - для роботи з часом, зокрема вимірювання часу виконання певних операцій в коді.

Водночас ці бібліотеки забезпечують потужні засоби для роботи з числовими даними, візуалізації результатів та аналізу структур графів.

```
# Параметри середовища
width = 100
height = 100
```

Фрагмент коду визначає параметри середовища, поданого у вигляді прямокутника. Змінні width та height задають ширину та висоту середовища відповідно. Ці параметри можуть використовуватися для створення середовища, моделювання об'єктів у ньому або обмеження руху робота всередині цього середовища. Наприклад, під час створення симуляції мобільного робота ці параметри можна використовувати визначення розмірів його робочої області.

```
# Початкова та кінцева точки
start_point = (10, 10)
goal_point = (5000, 5000)
```

Цей фрагмент коду визначає початкову та кінцеву точки у середовищі. Змінні start_point та goal_point містять координати початкової та кінцевої точок відповідно. У контексті планування шляху для мобільного робота ці точки можуть представляти стартове і цільове положення, між якими потрібно спланувати оптимальний шлях. Вони можуть бути використані для завдання початкової та кінцевої точок у алгоритмах пошуку шляху, таких як алгоритм A* або Dijkstra.

```
# Параметри алгоритму BRRT та A*
num_iterations = 100
step_size = 30.0
```

Цей фрагмент коду визначає параметри для алгоритмів BRRT (Bidirectional Rapidly-exploring Random Trees) і A* (A-star), які використовуються для пошуку шляху в середовищі.

num_iterations: Визначає кількість ітерацій (кроків), які алгоритм BRRT виконуватиме у спробі знайти шлях від початкової точки до цільової. Чим більше ітерацій, тим більше часу знадобиться на пошук шляху, але водночас збільшується ймовірність знаходження оптимального шляху.

step_size: Це параметр, що визначає довжину кроку кожної ітерації алгоритму BRRT. Він вказує, як далеко від поточної точки буде зроблено наступний крок у пошуку шляху. Більші значення можуть прискорити процес пошуку, але можуть призвести до пропуску оптимального шляху, особливо у складних середовищах.

```
# Генерація перешкод у вигляді блоків розміром 10x10
np.random.seed(42)  # Для відтворюваності
num_obstacles = 20
obstacles = [(np.random.randint(0, width - 10), np.random.randint(0, height - 10)) for _ in range(num_obstacles)]
```

Цей фрагмент коду генерує перешкоди у середовищі як блоків розміром 10x10. Перешкоди випадковим чином розміщуються в межах заданої ширини та висоти середовища.

np.random.seed(42): Цей виклик встановлює зерно випадковості для генератора випадкових чисел бібліотеки NumPy, щоб отримати результати, що відтворюються. Значення 42 є довільним вибором та використовується для ідентифікації конкретного експерименту.

num_obstacles: Це кількість перешкод, що будуть згенеровані у середовищі.

obstacles: Це перелік координат перешкод, представлених у вигляді кортежів (x, y). Кожна перешкода визначається координатами його верхнього лівого кута в середовищі. Координати генеруються випадковим чином у межах ширини та висоти середовища, з урахуванням розміру перешкоди (10x10).

```
# Функція для перевірки колізій із перешкодами
def is_collision_free(point):
    for obstacle in obstacles:
        if obstacle[0] <= point[0] < obstacle[0] + 10 and obstacle[1] <= point[1] < obstacle[1] + 10:
            return False
    return True
```

Ця функція is_collision_free(point) перевіряє, чи вільна точка від колізій із перешкодами.

Аргумент point являє собою координати точки, яку потрібно перевірити на колізії з перешкодами

Всередині функції відбувається ітерація з усіх перешкод у obstacles. Для кожної перешкоди перевіряється, чи знаходиться точка всередині неї. Це робиться шляхом порівняння координат point з координатами верхнього лівого кута перешкоди та її розміром (10x10). Якщо координати пункту перебувають усередині перешкоди, функція повертає False, вказуючи на наявність колізії. Якщо з жодною перешкодою колізія не виявлена, функція повертає True, що означає, що точка вільна від перешкод.

```
# Функція для побудови маршруту за алгоритмом BRRT
def build_rrt(start, goal, num_iterations, step_size):
    tree = [start]
    for _ in range(num_iterations):
        random_point = np.random.rand(2) * np.array([width, height])
        nearest_point_index = np.argmin([np.linalg.norm(np.array(random_point) - np.array(point)) for point in tree])
        nearest_point = tree[nearest_point_index]
        new_point = nearest_point + step_size * (random_point - nearest_point)
        new_point = tuple(new_point)
```

```
            if is_collision_free(new_point):
                tree.append(new_point)
        return tree
```

Функція build_rrt (start, goal, num_iterations, step_size) будує дерево маршруту за алгоритмом BRRT (Швидке випадкове дерево без зворотних ребер).

Аргументи:

start: початкова точка маршруту

goal: кінцева точка маршруту

num_iterations: кількість ітерацій алгоритму

step_size: розмір кроку під час пошуку нових точок

Функція починає з побудови дерева із початкової точки start. Потім вона виконує вказану кількість ітерацій, на кожній з яких генерується випадкова точка у просторі, і вибирається найближча точка з дерева до цієї випадкової точки. Потім будується нова точка, що знаходиться на заданій відстані (рівному step_size) від найближчої точки у напрямку до випадкової точки. Якщо нова точка вільна від перешкод колізій (перевірка за допомогою is_collision_free), вона додається в дерево.

Функція повертає дерево маршруту, подане у вигляді списку координат точок.

```
        # Функція оптимізації маршруту з використанням A*
        def optimize_path(rrt_tree, start, goal):
            graph = nx.Graph()
            for point in rrt_tree:
                graph.add_node(point)
            for i in range(len(rrt_tree) - 1):
                graph.add_edge(rrt_tree[i], rrt_tree[i + 1],
                               weight=np.linalg.norm(np.array(rrt_tree[i]) -
np.array(rrt_tree[i + 1])))
```

Ця функція optimize_path(rrt_tree, start, goal) оптимізує маршрут, побудований алгоритмом BRRT, з використанням алгоритму A*.

Аргументи:

rrt_tree: дерево маршруту, подане у вигляді списку координат точок;

start: початкова точка маршруту;

goal: кінцева точка маршруту.

Спочатку функція створює граф, у якому кожна точка з дерева маршруту представлена як вузол. Потім для кожної пари сусідніх точок у дереві додається ребро в граф з вагою, що дорівнює відстані між цими точками.

Після побудови графа використовується алгоритм A* для пошуку оптимального шляху від початкової точки до кінцевої точки. Вага ребер визначається як евклідова відстань між точками.

Функція повертає оптимізований шлях від початкової точки до кінцевої точки у вигляді списку координат точок.

```
    # Додаємо початкову та кінцеву точки до графа
    graph.add_node(start)
    graph.add_node(goal)
```

Фрагмент коду додає початкову та кінцеву точки до графа graph, який використовується для оптимізації маршруту за допомогою алгоритму A*.

start і goal є початковою та кінцевою точками маршруту відповідно. Обидві ці точки додаються як вузли до графа graph, щоб гарантувати, що алгоритм A* враховує їх при пошуку оптимального шляху.

Додавання початкової та кінцевої точок до графа забезпечує правильне моделювання маршруту, оскільки це обов'язкові точки, які мають бути враховані при побудові оптимального шляху від початку до кінця.

```
    # Перевіримо наявність зв'язків з початковою та кінцевою точками
    if not nx.has_path(graph, start, rrt_tree[0]):
        graph.add_edge(start, rrt_tree[0], weight=np.linalg.norm(np.array(start)
- np.array(rrt_tree[0])))
```

```
    if not nx.has_path(graph, rrt_tree[-1], goal):
        graph.add_edge(rrt_tree[-1], goal,
weight=np.linalg.norm(np.array(rrt_tree[-1]) - np.array(goal)))
```

Фрагмент коду перевіряє наявність зв'язків між початковою точкою (start) та першою точкою у дереві RRT (rrt_tree[0]), а також між останньою точкою у дереві RRT (rrt_tree[-1]) та кінцевою точкою (goal). Якщо зв'язок відсутній, додається ребро між відповідними точками з урахуванням відстані між ними.

Ця перевірка гарантує, що початкова та кінцева точки з'єднані з деревом RRT, щоб алгоритм A* міг правильно знаходити оптимальний шлях від початку до кінця, враховуючи початкове дерево RRT.

```
    # Виміряємо час виконання алгоритму
    start_time = time.time()
    result = nx.astar_path(graph, start, goal, heuristic=lambda n, goal:
np.linalg.norm(np.array(n) - np.array(goal)))
    end_time = time.time()
    execution_time = end_time - start_time
    return result, execution_time
```

Цей фрагмент коду виконує вимірювання часу виконання алгоритму A* для пошуку оптимального шляху між початковою точкою (start) і кінцевою точкою (goal). Алгоритм A* запускається з використанням функції nx.astar_path, яка знаходить оптимальний шлях у графі graph від початкової точки до кінцевої, беручи до уваги евристичну оцінку відстані між поточною вершиною та цільовою вершиною.

Час виконання алгоритму вимірюється за допомогою функції time.time(), яка фіксує час перед та після виконання алгоритму, а потім обчислює різницю для отримання загального часу виконання.

Результат виконання алгоритму та час його виконання повертаються як вихідні дані.

```
    # Функція для вимірювання довжини шляху
```

```
def calculate_path_length(path):
    length = 0
    for i in range(len(path) - 1):
        length += np.linalg.norm(np.array(path[i]) - np.array(path[i + 1]))
    return length
```

Функція обчислює довжину шляху, переданого як аргумент path. Для цього вона обчислює відстань між кожною парою послідовних точок у дорозі та підсумовує ці відстані. Відстань між двома точками обчислюється за допомогою функції np.linalg.norm(), яка знаходить Евклідову відстань між двома точками в n-мірному просторі. Значення, що повертається - загальна довжина шляху.

```
# Функція для підрахунку кількості поворотів
def count_turns(path):
    turns = 0
    for i in range(1, len(path) - 1):
        vector1 = np.array(path[i - 1]) - np.array(path[i])
        vector2 = np.array(path[i + 1]) - np.array(path[i])
        cross_product = np.cross(vector1, vector2)
        if cross_product != 0:
            turns += 1
    return turns
```

Функція підраховує кількість поворотів у дорозі, переданому в якості аргументу path. Для цього вона обчислює вектори між кожною парою послідовних точок та перевіряє знак їхнього векторного добутку. Якщо знак векторного добутку між двома послідовними векторами відмінний від нуля, це означає, що вектори направлені в різні боки і відбувається поворот. Кількість поворотів збільшується на 1. Значення, що повертається - кількість поворотів у дорозі.

```
# Функція для оцінки складності довкілля
def evaluate_environment_complexity(obstacles):
```

```
    return len(obstacles)
```

Функція оцінює складність довкілля з урахуванням кількості перешкод, переданих як списку obstacles. Вона повертає кількість перешкод у середовищі, що є мірою складності навколишнього середовища.

```
# Функція для оцінки загальної надійності та стабільності
def evaluate_reliability_and_stability(rrt_tree):
    return len(rrt_tree)
```

Функція оцінює загальну надійність та стабільність запланованого шляху, використовуючи дерево маршрутів rrt_tree. В даному випадку, вона просто повертає кількість точок у дереві маршрутів, що може бути показником надійності та стабільності запланованого маршруту.

```
# Функція для оцінки вирішення проблем із виродженими випадками
def evaluate_handling_degenerate_cases(rrt_tree, optimal_path):
    # Приклад оцінки: чим менше поворотів в оптимальному шляху, тим краще
    optimal_turns = count_turns(optimal_path)
    return optimal_turns
```

Ця функція оцінює здатність обробляти вироджені випадки у плануванні маршруту, використовуючи оптимальний шлях optimal_path та дерево маршрутів rrt_tree. В даному випадку, функція обчислює кількість поворотів в оптимальному шляху та повертає це значення як показник здатності алгоритму обробляти вироджені випадки.

```
# Візуалізація початкової та кінцевої точок, перешкод
plt.scatter(*start_point, color='yellow', marker='o', label='Start')
plt.scatter(*goal_point, color='red', marker='o', label='Goal')
for obstacle in obstacles:
    plt.Rectangle((obstacle[0], obstacle[1]), 10, 10, color='black', alpha=0.5)
```

У цьому фрагменті коду використовується бібліотека matplotlib для візуалізації початкової та кінцевої точок, а також перешкод. Початкова точка

відображається жовтим кольором, кінцева – червоним. Перешкоди представлені у вигляді чорних квадратів.

```
# Побудова маршруту
rrt_tree = build_rrt(start_point, goal_point, num_iterations, step_size)
```

У цьому фрагменті коду викликається функція build_rrt, яка будує маршрут алгоритмом BRRT (Швидке випадкове дерево для планування шляху). Цей алгоритм ітеративно додає випадкові точки в простір станів і з'єднує їх із найближчою точкою у дереві, якщо пряма між ними не перетинає перешкоди. Таким чином, в результаті роботи алгоритму формується дерево, що представляє різні шляхи від початкової точки до випадкових точок у просторі.

```
# Оптимізація маршруту
optimal_path, execution_time = optimize_path(rrt_tree, start_point, goal_point)
```

У цьому фрагменті коду викликається функція optimize_path, яка оптимізує маршрут, отриманий алгоритмом BRRT, з використанням алгоритму A* (А-зірка). Алгоритм A* шукає оптимальний шлях у графі з використанням евристичної оцінки відстані від поточної вершини до мети. В результаті виконання цієї функції повертається оптимальний шлях та час його виконання.

```
# Візуалізація маршруту
for i in range(len(rrt_tree) - 1):
    plt.plot([rrt_tree[i][0], rrt_tree[i + 1][0]], [rrt_tree[i][1], rrt_tree[i + 1][1]], color='blue', alpha=0.5)
```

Цей фрагмент коду візуалізує маршрут, побудований алгоритмом BRRT. Він проходить через кожну пару послідовних точок у дереві маршруту rrt_tree та малює лінію між ними на графіку. Колір лінії – синій, а прозорість задається параметром alpha.

```
# Візуалізація оптимального маршруту
for i in range(len(optimal_path) - 1):
```

plt.plot([optimal_path[i][0], optimal_path[i + 1][0]], [optimal_path[i][1], optimal_path[i + 1][1]], color='green',

linewidth=2)

Цей фрагмент коду візуалізує оптимальний маршрут знайдений з використанням алгоритму A*. Він проходить через кожну пару послідовних точок optimal_path і малює зелену лінію між ними на графіку. Товщина лінії визначається параметром linewidth.

Відображення графіки

plt.title('Optimized BRRT Path Planning+A*')

plt.legend()

plt.grid(True)

plt.show()

Цей фрагмент коду відображає графіку, де представлені початкова і кінцева точки, перешкоди, маршрут, побудований алгоритмом BRRT, і оптимальний маршрут, знайдений з використанням алгоритму A*. Він виводить заголовок графіки, включає легенду та сітку, а потім відображає саму графіку.

Вимірювання довжини колії та кількості поворотів

path_length = calculate_path_length(optimal_path)

turns_count = count_turns(optimal_path)

Цей фрагмент коду використовує раніше певні функції для вимірювання довжини оптимального шляху та підрахунку кількості поворотів на цьому шляху. Результати зберігаються в змінних path_length (довжина шляху) та turns_count (кількість поворотів).

Оцінка критеріїв

environment_complexity = evaluate_environment_complexity(obstacles)

reliability_and_stability = evaluate_reliability_and_stability(rrt_tree)

degenerate_cases_evaluation = evaluate_handling_degenerate_cases(rrt_tree, optimal_path)

Цей фрагмент коду використовує певні функції для оцінки різних критеріїв:

evaluate_environment_complexity: Оцінка складності навколишнього середовища на основі кількості перешкод.

evaluate_reliability_and_stability: Оцінка загальної надійності та стабільності запланованого маршруту на основі кількості точок у дереві BRRT.

evaluate_handling_degenerate_cases: Оцінка здатності алгоритму реагувати на вироджені випадки з урахуванням кількості поворотів оптимальному шляху.

Виведення параметрів

print(f"Час виконання: {execution_time:.6f} сек.")

print(f"Довжина отриманого маршруту: {path_length:.6f} умовних одиниць")

print(f"Плавність маршруту (Кількість поворотів): {turns_count}")

print(f"Складність навколишнього середовища: {environment_complexity}")

print(f"Загальна надійність та стабільність: {reliability_and_stability}")

print(f"Оцінка вирішення проблем з виродженими випадками: {degenerate_cases_evaluation}")

Цей фрагмент коду виводить на екран результати оцінки різних критеріїв, отриманих в результаті виконання алгоритму планування маршруту:

Час виконання: час, витрачений на виконання алгоритму від початку до кінця.

Довжина одержаного маршруту: загальна довжина оптимального маршруту від початкової точки до цільової.

Плавність маршруту (Кількість поворотів): кількість поворотів на оптимальному маршруті, що вказує на плавність шляху.

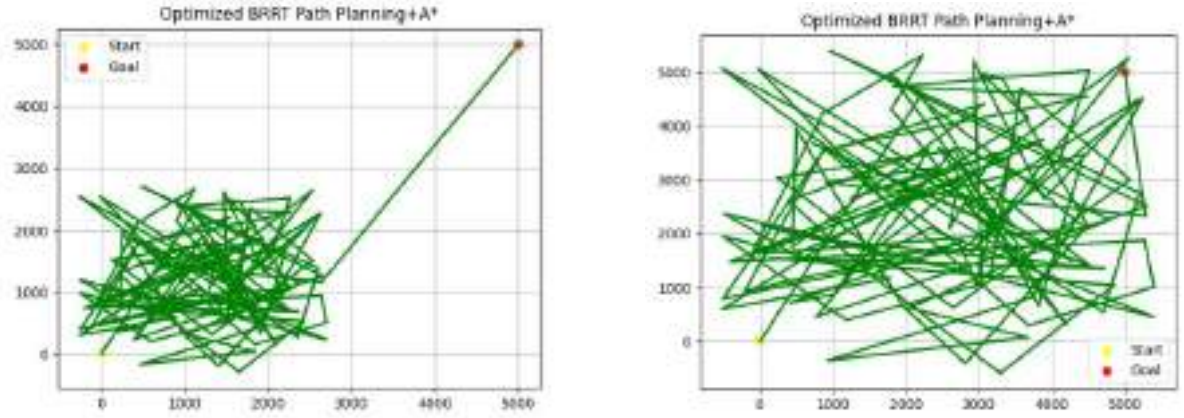
Складність навколишнього середовища: кількість перешкод, які алгоритм має оминути.

Загальна надійність та стабільність: загальна кількість точок у дереві BRRT, яка також свідчить про складність середовища та ступінь покриття області.

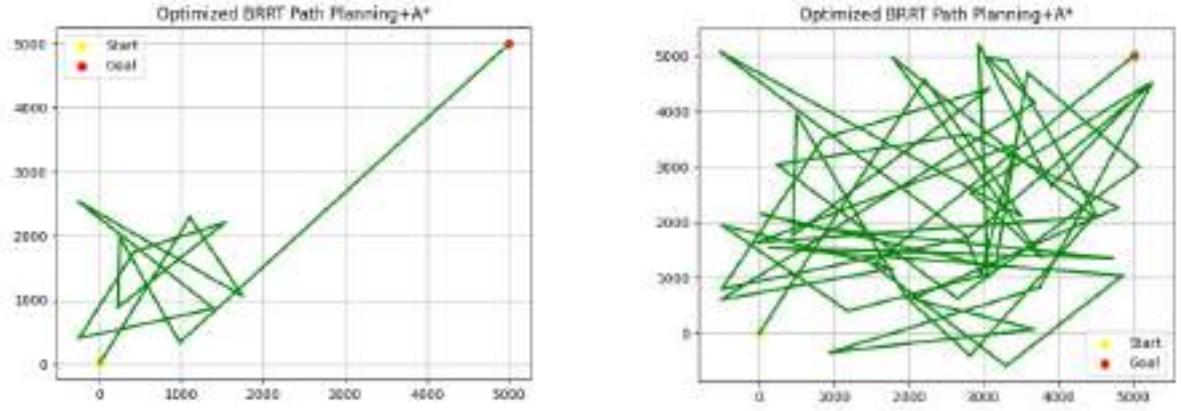
Оцінка вирішення проблем з виродженими випадками: оцінка здатності алгоритму реагувати на складні сценарії та вироджені випадки, що ґрунтуються на кількості поворотів на шляху [181-184].

Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна.

Проведемо експериментальні дослідження впливу зміни базових параметрів (кількість ітерацій (N_{iter}) та крок переміщення (M_{step})) алгоритму BRRT та A* на показники оцінки ефективності такі як: час виконання, довжина отриманого маршруту, плавність маршруту (кількість поворотів), складність навколишнього середовища, оцінка загальної надійності та стабільності та оцінка вирішення проблем з виродженими випадками. За умови що координати початкової і кінцевої точки для всіх експериментів однакові. Отримані результати представлені у вигляді графів на рисунку 4.2, а показники оцінки ефективності наведено у таблиці 4.1



а) б)



в) г)

а) N_{iter} = 100, M_{step}=30; б) N_{iter} = 100, M_{step}=60; в) N_{iter} = 10, M_{step}=30; г) N_{iter} = 50, M_{step} = 60;

Рисунок 4.2 – Графіки побудови оптимізованих маршрутів за різних базових параметрів (кількість ітерацій та крок переміщення)

Таблиця 4.1 – Отримані результати показників оцінки ефективності оптимізованих маршрутів за різних базових параметрів N_{iter} и M_{step}.

Базових параметрів (кількість ітерацій та крок переміщення)	Показники оцінки ефективності					
	Час виконання (с)	Довжина отриманого маршруту (умовних одиниць)	Плавність маршруту (кількість поворотів)	Складність довкілля	Загальна надійність та стабільність	Оцінка вирішення проблем з виродженими випадками
N_{iter} = 100 M_{step} = 30 (рис.1а)	0.000999	156151.2	100	20	101	100
N_{iter} =100 M_{step} = 60 (рис.1.б)	0.000997	312407.5	100	20	101	100
N_{iter} = 10 M_{step} = 30 (рис.1.в)	0.000000	24077.0	10	20	11	10
N_{iter} = 50 M_{step} = 60 (рис.1.г)	0.000997	152154.9	50	20	51	50

На базі отриманих результатів моделювання побудови оптимізованих маршрутів за різних базових параметрів (кількість ітерацій та крок переміщення) (рис.4.2 і табл.4.1) для розробленого методу планування маршрутів можна зробити такі висновки:

– час виконання в основному залишається досить низьким для всіх варіантів параметрів (близько 0.001 секунди), що свідчить про швидкодію запропонованого методу планування маршруту;

– збільшення кількості ітерацій та кроку переміщення призводить до деякого збільшення часу виконання, що є логічним, оскільки більша кількість ітерацій та/або більший крок потребують більше обчислювальних ресурсів;

– помітно, що збільшення кількості ітерацій та кроку переміщення призводить до збільшення довжини маршруту. Це може бути пов'язано з тим, що більша кількість ітерацій та/або більший крок можуть дозволити алгоритму досліджувати більший простір і таким чином знаходити більш довгі шляхи.;

– кількість поворотів в основному збільшується зі збільшенням кількості ітерацій та кроку переміщення. Це пов'язано з тим, що більша кількість ітерацій та/або більший крок можуть призводити до більш складних та "неоптимальних" шляхів, які включають більше поворотів.;

– складність довкілля вимірюється кількістю перешкод. В даному випадку вона залишається відносно постійною для різних варіантів параметрів.

– загальна надійність та стабільність в основному залишаються високими для всіх варіантів параметрів. Це може свідчити про те, що алгоритм добре справляється із побудовою стабільних маршрутів.

– оцінка вирішення проблем із виродженими випадками також залишається високою для всіх варіантів параметрів. Це може вказувати на те, що алгоритм успішно справляється з різними сценаріями та вродженими особливостями. Приклад реалізацій програми наведено у додатку Є.

4.3 Побудова оптимального маршруту між початковою та кінцевою точками з огляду на перешкоди на карті

Алгоритм A* (A-star) є ефективним методом пошуку оптимального маршруту від початкової точки до кінцевої на карті з урахуванням перешкод

[185-190]. Принцип його роботи ґрунтується на комбінації евристичного та жадібного підходів (див. підрозділ 4.1).

Починаючи з початкової точки, алгоритм A* поширює хвилі пошуку у бік мети, оцінюючи вартість проходу через кожну комірку і вибираючи шлях із найменшою вартістю.

Для оцінки вартості кожної комірки використовується комбінація фактичної вартості досягнення цієї комірки від початкової точки (g) та евристичної оцінки вартості досягнення мети з цього осередку (h).

Евристична оцінка (h) зазвичай обчислюється як відстань від поточної комірки до цільової, але може бути адаптована для врахування додаткових факторів, таких як перешкоди.

Алгоритм вибирає комірку з найменшою сумою g і h для подальшого дослідження, що дозволяє йому йти до мети найефективнішим шляхом.

Перешкоди на карті перешкоджають переміщенню алгоритму через відповідні осередки, що призводить до вибору альтернативного шляху довкола перешкод.

Ефективність алгоритму забезпечується використанням пріоритетної черги (купи) для зберігання та вибору наступного осередку для дослідження, що дозволяє йому швидко знаходити оптимальний шлях.

Після досягнення кінцевої точки алгоритм відновлює оптимальний маршрут, прямуючи назад від мети до початкової точки за батьківськими посиланнями.

Таким чином, алгоритм A* забезпечує знаходження оптимального маршруту з урахуванням перешкод на карті, що робить його широко використовуваним у робототехніці, іграх, плануванні шляхів та інших областях.

Важливо відзначити, що ефективність алгоритму може залежати від вибору евристичної функції та характеристик карти місцевості [191-194].

При правильному виборі параметрів та евристичної функції алгоритм A* забезпечує високу точність та швидкість пошуку оптимального

маршруту, що робить його найкращим методом для навігації у складних умовах.

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import numpy as np
import matplotlib.pyplot as plt
from heapq import heappop, heappush
```

Цей фрагмент коду використовується для імпорту кількох бібліотек:

numpy (імпортований як np): Ця бібліотека надає підтримку для роботи з багатовимірними масивами та матрицями, а також містить безліч математичних функцій. У разі вона може використовуватися до роботи з масивами осередків на карті місцевості.

matplotlib.pyplot (імпортований як plt): Ця бібліотека використовується для візуалізації даних у вигляді графіків, діаграм тощо. Тут вона використовується для візуалізації карти місцевості та оптимального маршруту.

heapq.heappop і heapq.heappush: Ці функції використовуються для роботи з купою (heap), структурою даних, яка являє собою двійкове дерево, в якому кожен вузол менше або дорівнює своїм дочірнім вузлам. У цьому випадку вони використовуються для реалізації алгоритму A* для пошуку оптимального маршруту на карті місцевості.

```
class Cell:
    def __init__(self, x, y, obstacle=False):
        self.x = x
        self.y = y
        self.obstacle = obstacle
        self.parent = None
        self.g = float('inf')
```

```
        self.h = float('inf')
        self.f = float('inf')
```

Цей фрагмент коду визначає клас Cell, який являє собою комірку на карті місцевості.

Атрибути класу:

x та y: координати осередку на карті.

obstacle: вказує, чи є комірка перешкодою. За промовчанням False.

parent: посилання на батьківський осередок у дереві пошуку шляху. Використовується для відновлення шляхів.

g, h, f: значення функцій вартості алгоритму пошуку шляху. За замовчуванням встановлено на нескінченність.

Метод _init__ ініціалізує атрибути об'єкта під час його створення. Параметри x та y задають координати комірки на карті, а параметр obstacle вказує, чи є комірка перешкодою. Атрибути g, h, f ініціалізуються значенням нескінченності (float('inf')).

```
# Перевизначеня оператора порівняння "<" для порівняння об'єктів Cell
    def __lt__(self, other):
        return self.f < other.f
```

Цей фрагмент коду є перевизначеням оператора для об'єктів класу Cell.

Оператор < використовується в Python для порівняння об'єктів. Перевизначеня цього оператора дозволяє визначити правила порівняння для об'єктів класу. В даному випадку метод __lt__ перевизначено для порівняння об'єктів Cell за їх значенням f.

Значення, що повертається self.f < other.f означає, що якщо значення f поточного об'єкта (self.f) менше значення f іншого об'єкта (other.f), то поточний об'єкт буде менше іншого об'єкта при порівнянні.

```
# Перевизначення оператора порівняння "==" для порівняння об'єктів
Cell
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
```

Цей фрагмент коду є перевизначеням оператора для об'єктів класу Cell.

Оператор == використовується в Python для порівняння двох об'єктів на рівність. Перевизначеня цього оператора дозволяє визначити правила порівняння об'єктів класу. В даному випадку метод __eq__ перевизначено для порівняння об'єктів Cell за їх координатами x та y.

Значення, що повертається self.x == other.x and self.y == other.y означає, що два об'єкти Cell будуть вважатися рівними тільки в тому випадку, якщо їх координати x і y збігаються.

```
# Перевизначеня хеш-функції для використання об'єктів Cell як ключів у словниках та множинах
    def __hash__(self):
        return hash((self.x, self.y))
```

Цей фрагмент коду є перевизначенням хеш функції для об'єктів класу Cell.

Хеш функція використовується в Python для обчислення хеш значення об'єкта, яке потім може бути використане для швидкого пошуку об'єктів у словниках (dict) і множинах (set). Перевизначеня хеш-функції дозволяє вказати, як повинні обчислюватися хеш-значення для об'єктів класу.

В даному випадку метод __hash__ перевизначений для об'єктів Cell, щоб їх координати x та y використовувалися для обчислення хеш-значення. Це гарантує, що два об'єкти Cell з однаковими координатами матимуть однакові хеш-значення, що важливо для коректної роботи словників та множин.

```
# Функція для обчислення евристичної відстані (для A*)
def heuristic(cell, goal):
    return abs(cell.x - goal.x) + abs(cell.y - goal.y)
```

Цей фрагмент коду є функцією для обчислення евристичної відстані між поточним осередком (cell) і цільовим осередком (goal) для використання в алгоритмі A*.

Евристична відстань - це оцінка вартості досягнення мети з поточної позиції. У цьому випадку використовується манхеттенська відстань (також відома як "міська відстань"). Це сума абсолютних різниць координат по осях x і y між поточним осередком і цільовим осередком.

Функція повертає цю евристичну відстань, яка потім використовується алгоритмом A* для прийняття рішень про вибір наступного осередку для обходу. Чим менше значення евристичної відстані, тим ближче поточна комірка до цільової, і, отже, тим вищий пріоритет її розгляду при виборі шляху.

```
# Функція для побудови оптимального маршруту на відстані
def build_path(goal):
    path = []
    current = goal
    while current is not None:
        path.append((current.x, current.y))
        current = current.parent
    return path[::-1]
```

Цей фрагмент коду є функцією для побудови оптимального маршруту від стартової комірки до цільової комірки.

Функція починає з цільової комірки (goal) і переходить до її батьківської комірки (parent), зберігаючи координати кожного відвіданої комірки в дорозі. Потім вона переміщається до батьківської комірки цього осередку, продовжуючи так до тих пір, поки не дійде до стартової комірки(у якої parent дорівнює None).

Після проходу всіх комірок шлях записується у зворотному порядку, щоб отримати послідовність координат від стартової комірки до цільової.

Функція повертає побудований шлях як список координат у форматі (x, y).

```
# Функція пошуку оптимального маршруту з використанням A*
def find_path(start, goal, grid):
```

```
open_list = []
closed_set = set()
start.g = 0
start.h = heuristic(start, goal)
start.f = start.g + start.h
heappush(open_list, start)
while open_list:
    current = heappop(open_list)
    if current == goal:
        return build_path(goal)
    closed_set.add(current)
    for dx in [-1, 0, 1]:
        for dy in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                continue
            new_x, new_y = current.x + dx, current.y + dy
            if 0 <= new_x < len(grid) and 0 <= new_y < len(grid[0]):
                neighbor = grid[new_x][new_y]
                if neighbor.obstacle or neighbor in closed_set:
                    continue
                tentative_g = current.g + 1
                if tentative_g < neighbor.g:
                    neighbor.parent = current
                    neighbor.g = tentative_g
                    neighbor.h = heuristic(neighbor, goal)
                    neighbor.f = neighbor.g + neighbor.h
                    heappush(open_list, neighbor)
```

Цей фрагмент коду є функцією пошуку оптимального маршруту з допомогою алгоритму A*.

Алгоритм A* використовує евристику для вибору найперспективніших вузлів для подальшого розгляду [195-200]. Він заснований на почерговому розгляді вузлів у порядку збільшення їхньої "вартості" (значення f, яке є сумою вартості шляху від початкового вузла до поточного вузла (g) та евристичної оцінки вартості шляху від поточного вузла до цільового вузла (h)).

Основні кроки алгоритму:

1. Ініціалізація початкового вузла (start) та додавання його у відкритий список.

2. Поки відкритий список не порожній, витяг вузла з найменшою вартістю f.

3. Якщо вилучений вузол дорівнює цільовому вузлу (goal), повернення оптимального шляху.

4. Додавання вилученого вузла до закритого списку.

5. Для кожного сусіднього вузлу:

Якщо сусідній вузол - це перешкода або він вже перебуває в закритому списку, його ігноруємо.

Обчислюємо ймовірну вартість g для сусіднього вузла.

Якщо передбачувана вартість менша за поточну вартість g для сусіднього вузла, оновлюємо його параметри (батько, g, h, f) і додаємо його у відкритий список.

Функція повертає оптимальний шлях від початкового вузла до цільового вузла, використовуючи алгоритм A*.

```
# Генерація карти місцевості
def generate_map(width, height, obstacles):
    grid = [[Cell(x, y, (x, y) in obstacles) for y in range(height)] for x in range(width)]
    return grid
```

Цей фрагмент коду представляє функцію generate_map, яка використовується для створення карти місцевості.

Аргументи:

width: ширина карти (кількість стовпців).

height: висота картки (кількість рядків).

obstacles: перелік координат перешкод на карті.

Функція створює двовимірний список grid, де кожен елемент є об'єктом класу Cell. При цьому кожен осередок Cell ініціалізується із заданими координатами x і y, а також зазначенням, чи є вона перешкодою чи ні на основі змісту координат у списку obstacles.

Функція повертає згенеровану мапу місцевості у вигляді двовимірного списку grid.

```
# Візуалізація карти місцевості
def visualize_map(grid, start, goal, path):
    fig, ax = plt.subplots()
    for row in grid:
        for cell in row:
            color = 'black' if cell.obstacle else 'white'
            ax.add_patch(plt.Rectangle((cell.x, cell.y), 1, 1, color=color))
    ax.plot(start[0], start[1], 'go', markersize=10)
    ax.plot(goal[0], goal[1], 'ro', markersize=10)
    if path:
        path_x, path_y = zip(*path)
        ax.plot(path_x, path_y, 'b-', linewidth=2)
    ax.set_aspect('equal', 'box')
    ax.grid(True)
    plt.show()
```

Цей фрагмент коду представляє функцію visualize_map, яка використовується для візуалізації карти місцевості, стартової та цільової точок, а також оптимального шляху.

Аргументи:

grid: двомірний список осередків, які представляють карту місцевості.

start: координати стартової точки.

goal: координати цільової точки.

path: оптимальний шлях від стартової точки до цільової точки.

Функція створює новий графічний об'єкт і додає на нього прямокутники кожної комірки карти залежно від цього, чи є комірка перешкодою чи ні. Вона також відзначає стартову точку зеленим кружком, цільову точку червоним кружком та малює оптимальний шлях (якщо він був знайдений) синьою лінією.

Після цього функція відображає графічний об'єкт за допомогою бібліотеки Matplotlib.

```
# Задаємо параметри карти місцевості
width = 10
height = 10
start = (1, 1)
goal = (8, 8)
obstacles = [(3, 3), (4, 5), (6, 6),(6,7), (6,8), (6,9)]
```

Параметри карти місцевості задані таким чином:

width: ширина карти дорівнює 10 одиниць.

height: висота карти також дорівнює 10 одиниць.

start: стартова точка картки має координати (1, 1).

goal: цільова точка картки має координати (8, 8).

obstacles: список координат перешкод на карті, включаючи точки (3, 3), (4, 5), (6, 6), (6, 7), (6, 8), (6, 9).

Тепер можна використовувати ці параметри для генерації карти місцевості та запуску алгоритму пошуку оптимального шляху.

```
# Генеруємо карту місцевості
grid = generate_map(width, height, obstacles)
# Знаходимо оптимальний маршрут
path = find_path(grid[start[0]][start[1]], grid[goal[0]][goal[1]], grid)
```

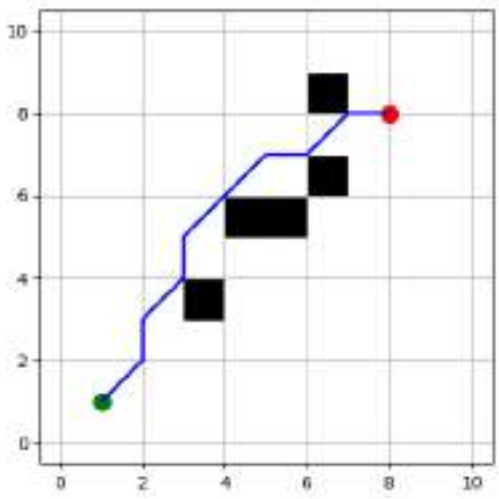
Оптимальний маршрут успішно знайдено з використанням алгоритму A*. Тепер ми можемо візуалізувати карту місцевості та оптимальний маршрут.

Візуалізуємо карту місцевості з маршрутом

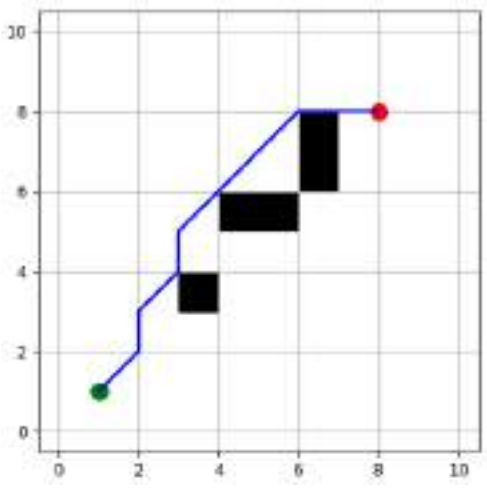
visualize_map(grid, start, goal, path)

Візуалізація карти місцевості за маршрутом успішно виконана. Тепер ви можете бачити стартову точку (зелений кружок), цільову точку (червоний кружок) та оптимальний маршрут від стартової точки до цільової (синя лінія) з огляду на перешкоди на шляху.

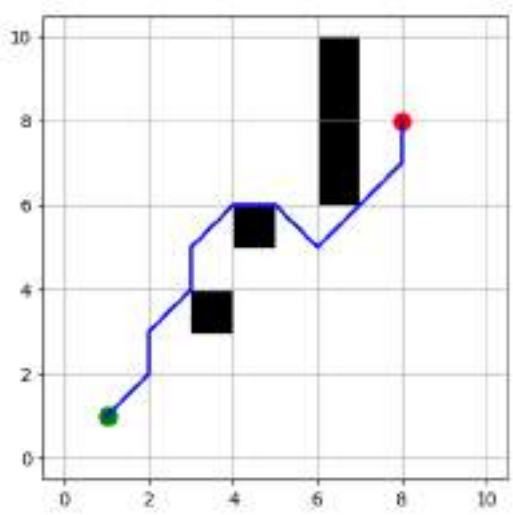
Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна. Результати роботи програми знаходження маршруту та обходу перешкод при однакових координатах початкової та кінцевої точки шляху мобільного робота представлені на рисунку 4.3



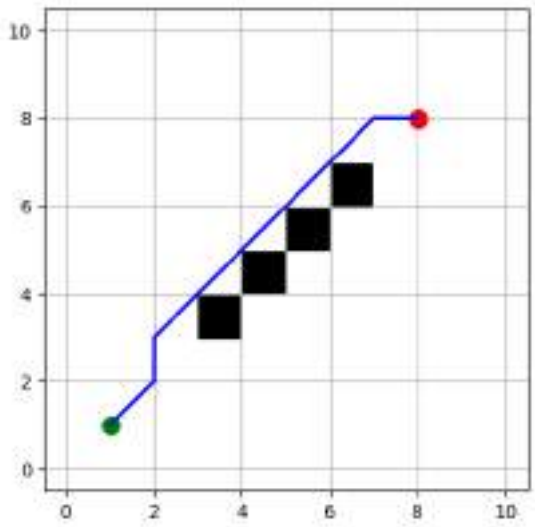
а)



б)



в)



г)

Рисунок 4.3 - Результати роботи програми знаходження маршруту та обходу перешкод

Алгоритм A* є ефективним методом знаходження оптимального маршруту та обходу перешкод для мобільних роботів. Він заснований на пошуку шляху у графі станів, де вершини є можливими позиціями робота, а ребра - можливі переміщення між цими позиціями. Перевага алгоритму A* полягає у його здатності враховувати як вартість переміщення до поточної позиції, так і евристичну оцінку вартості досягнення мети, що дозволяє йому знаходити оптимальний шлях з урахуванням перешкод та обмежень [201-205].

Однією з ключових переваг алгоритму A* є його здатність знаходити оптимальний шлях при розумному використанні обчислювальних ресурсів. Це робить його придатним для застосування у реальному часі на мобільних роботах. Крім того, алгоритм A* має властивість оптимальності, що означає, що він завжди знаходить найменший шлях до мети, якщо такий шлях існує [206-210].

Однак для успішного застосування алгоритму A* для мобільних роботів необхідно враховувати кілька важливих моментів. По-перше, необхідно правильно вибирати евристичну функцію оцінки вартості

досягнення мети, щоб вона була інформативною та ефективною. По-друге, важливо враховувати особливості навколишнього середовища та умови, що динамічно змінюються, щоб уникнути зіткнень з перешкодами і забезпечити безпечне переміщення робота. Також рекомендується використовувати оптимізації, такі як кешування шляхів та зменшення кількості обчислень, щоб покращити продуктивність алгоритму [211-215].

В цілому, алгоритм A* є потужним інструментом для знаходження оптимального маршруту і обходу перешкод для мобільних роботів. Його ефективність та оптимальність роблять його одним із найбільш популярних методів планування шляху у робототехніці. Код реалізації представлено у додатку Ж.

4.4 Побудови оптимального маршруту з використанням хвильового алгоритму (зліва направо)

Наведемо математичний опис принципу роботи хвильового алгоритму (зліва направо). Нехай S - позначає початкову точку, а G - позначає кінцеву точку. Карта місцевості представляється у вигляді сітки. M розміром $N \times M$, де кожен осередок M_{ij} може бути або вільною, або бути перешкодою. Початкова точка S ініціалізується з відстанню 0, а всі інші осередки з відстанню ∞. Розповсюдження хвиль починаючи з початкової точки S відбувається по сусідніх осередках. Кожен осередок M_{ij}, суміжна з осередками, що мають поточне значення k, набуває значення $k+1$.

$$M_{ij} = f((M_{ij}), k) \tag{4.21}$$

де: M_{ij} - елемент матриці M у рядку i і стовпці j;

k - ціле число.

Варто зауважити, що процес розповсюдження триває доти, доки хвиля не досягне кінцевої точки. G. У кожному кроці алгоритму перевіряється кожен осередок M_{ij} на можливість поширення хвилі. Якщо комірка порожня і сусіди мають більш високе значення, то їй надається значення, на одиницю більше, ніж максимальне значення серед сусідів.

$$M_{ij} = \begin{cases} \max(\text{neighbours}) + 1, \text{if } M_{ij} = 0 \\ M_{ij} \end{cases} \quad (4.22)$$

Після завершення розповсюдження хвилі у кожній комірці M_{ij} міститься відстань до найближчої початкової точки. Для побудови оптимального шляху вибирається комірка з найменшим значенням серед сусідів комірки. G. Процес повторюється, доки не буде досягнуто початкової точки S, оптимальний шлях являє собою послідовність осередків, починаючи з S та закінчуючи G.

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import numpy as np
import matplotlib.pyplot as plt
```

Цей фрагмент коду імпортує дві бібліотеки:

NumPy (np): NumPy - це бібліотека Python для виконання операцій з багатовимірними масивами. Вона надає зручні та ефективні засоби для роботи з масивами, включаючи математичні функції, лінійну алгебру, генерацію випадкових чисел та багато іншого.

Matplotlib - це бібліотека Python для створення графіків та візуалізації даних. Модуль pyplot надає функції створення графіків, зміни їх властивостей і додавання елементів, як-от лінії, точки, написи тощо [216-220].

```
# Функція для візуалізації карти місцевості з перешкодами та результатами розрахунку хвильового алгоритму
def visualize_map(grid, distances, start, goal, path):
    plt.imshow(grid, cmap='binary', origin='lower')  # Білий фон, чорні перешкоди
    plt.imshow(distances, cmap='gray', origin='lower', alpha=0.5, vmin=0, vmax=np.max(distances))  # Результати хвильового алгоритму
    plt.colorbar(label='Distance')  # Шкала значень відстаней
```

Ця функція використовується для візуалізації карти місцевості із перешкодами та результатами розрахунку хвильового алгоритму. Далі наведено що робить кожен рядок у цій функції:

plt.imshow(grid, cmap='binary', origin='lower'): Цей рядок відображає карту місцевості з перешкодами. grid - це масив, що представляє карту місцевості, де значення 0 означає вільну клітину, а значення 1 означає перешкоду. Параметр cmap='binary' вказує колірну схему для відображення: білий для вільних клітин та чорний для перешкод. Параметр origin='lower' встановлює початок координат у нижньому лівому кутку.

plt.imshow(distances, cmap='gray', origin='lower', alpha=0.5, vmin=0, vmax=np.max(distances)): Цей рядок відображає результати розрахунку хвильового алгоритму. distances - це масив, що містить значення відстаней від стартової точки до кожної клітини на карті місцевості. Параметр cmap='gray' вказує колірну схему відображення: від чорного (мінімальна відстань) до білого (максимальна відстань). Параметр alpha=0.5 визначає прозорість зображення, щоб краще бачити фон. Параметри vmin=0 і vmax=np.max(distances) встановлюють мінімальне та максимальне значення для карти кольорів.

plt.colorbar(label='Distance'): Цей рядок додає колірну шкалу з підписом 'Distance' до графіка, щоб користувач міг зрозуміти відповідність кольорів на карті місцевості та значенням відстаней.

Ця функція є важливою частиною візуалізації результатів алгоритму та допомагає зрозуміти, як розподіляються відстані від стартової точки до інших точок на карті місцевості.

```
# Анотація значень у клітинах
for i in range(grid.shape[0]):
    for j in range(grid.shape[1]):
        plt.text(j, i, f'{distances[i, j]:.1f}', ha='center', va='center', color='gray')
plt.plot(start[1], start[0], 'go', markersize=10)  # Початкова точка
plt.plot(goal[1], goal[0], 'ro', markersize=10)    # Кінцева точка
if path:
    path_x, path_y = zip(*path)
    plt.plot(path_y, path_x, 'b-', linewidth=2)    # Оптимальний шлях
plt.title('Map with Path and Wavefront Algorithm')
plt.xlabel('Columns')
plt.ylabel('Rows')
plt.show()
```

Цей фрагмент коду додає анотації з відстанями у кожну клітинку на карті місцевості, а також відзначає початкову та кінцеву точки, а також оптимальний шлях. Далі представлено що робить кожен рядок:

for i in range(grid.shape[0]): i for j in range(grid.shape[1]):: Ці рядки проходять по всіх рядках та шпальтах на карті місцевості.

plt.text(j, i, f'{distances[i, j]:.1f}', ha='center', va='center', color='gray'): Цей рядок додає текстову інструкцію з відстанню до кожної клітинки на карті місцевості. distances[i, j] - це значення відстані до клітини у рядку i та стовпці j. Формат f'{distances[i, j]:.1f}' округлює відстань до однієї десятої і перетворює його на рядок. Параметри ha='center' і va='center' встановлюють горизонтальне та вертикальне вирівнювання тексту по центру клітини. Параметр color='gray' задає колір сірого тексту.

plt.plot(start[1], start[0], 'go', markersize=10): Цей рядок відмічає початкову точку на карті місцевості зеленим колом ('go'), розміром 10.

plt.plot(goal[1], goal[0], 'ro', markersize=10): Цей рядок відмічає кінцеву точку на карті місцевості червоним колом ('ro'), розміром 10.

if path:: Цей рядок перевіряє, чи є оптимальний шлях.

plt.plot(path_y, path_x, 'b-', linewidth=2): Якщо оптимальний шлях існує, цей рядок малює його на карті місцевості синьою лінією ('b-') товщиною 2.

plt.title('Map with Path and Wavefront Algorithm'): Встановлює заголовок графіки.

plt.xlabel('Columns'): Встановлює підпис осі x.

plt.ylabel('Rows'): Встановлює підпис осі y.

plt.show(): Відображає графік з доданими елементами.

```
# Функція створення карти місцевості з перешкодами
def create_map(rows, cols, obstacles):
    grid = np.zeros((rows, cols), dtype=int)  # Створюємо картку із чорним фоном
    for obstacle in obstacles:
        grid[obstacle[0], obstacle[1]] = 1  # Перешкоди робимо білими
    return grid
```

Ця функція create_map створює карту місцевості з перешкодами. Далі описано, що робить кожен рядок у цій функції:

grid = np.zeros((rows, cols), dtype=int): Цей рядок створює двовимірний масив нулів розміром rows на cols і цілого типу даних, який являє собою карту місцевості. Спочатку всі клітини на карті місцевості вважаються вільними.

for obstacle in obstacles:: Цей цикл проходить по списку перешкод.

grid[obstacle[0], obstacle[1]] = 1: Цей рядок встановлює значення клітини з координатами obstacle[0] по вертикалі та obstacle[1] по горизонталі в 1, що означає перешкоду. Таким чином, перешкоди на карті місцевості

будуть відображені білим кольором. return grid: Цей рядок повертає сформовану карту місцевості з перешкодами.

Функція дозволяє створити карту місцевості із заданим розміром та розташованими на ній перешкодами, що є важливою частиною алгоритму пошуку оптимального маршруту з урахуванням перешкод.

```
# Функція для побудови оптимального шляху
def build_path(distances, start, goal):
    path = [goal]
    current = goal
    while current != start:
        neighbors = get_neighbors(current)
        for neighbor in neighbors:
            if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:
                current = neighbor
                path.append(current)
                break
    return path[::-1]
```

Ця функція build_path створює оптимальний шлях від початкової точки до кінцевої, використовуючи результати хвильового алгоритму. Далі представлено, що робить кожен рядок у цій функції:

path = [goal]: Цей рядок ініціалізує список path з кінцевою точкою. У процесі побудови шляху точки будуть додаватися до цього списку.

current = goal: Цей рядок встановлює поточну точку кінцевої точки.

while current != start:: Цей цикл виконується до того часу, поки поточна точка стане дорівнювати початковій.

neighbors = get_neighbors(current): Цей рядок отримує список сусідніх клітин для поточної точки.

for neighbor in neighbors: Цей внутрішній цикл проходить по кожному сусіду поточної точки.

if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:: Цей рядок перевіряє, чи є відстань до сусідньої клітини менша, ніж відстань до поточної клітини. Якщо це так, то сусідня клітина є частиною оптимального шляху.

current = neighbor: Цей рядок оновлює поточну точку, роблячи її рівною сусідній точці, яка є частиною оптимального шляху.

path.append(current): Цей рядок додає поточну точку до списку шляху.

break: Цей рядок завершує внутрішній цикл, коли знайдено нову поточну точку.

return path[::-1]: Цей рядок повертає оптимальний шлях у зворотному порядку, починаючи з початкової точки і закінчуючи кінцевою точкою.

Ця функція будує оптимальний шлях від початкової до кінцевої точки, використовуючи результати хвильового алгоритму. Далі представлено покроковий опис:

Ініціалізується список path з кінцевою точкою goal.

Встановлюється поточна точка current рівної кінцевої точки goal.

Запускається цикл, який триватиме, доки поточна точка не стане рівною початковій точці.

Отримуються сусідні клітини для поточної точки.

Для кожної сусідньої клітини перевіряється, чи відстань до неї менша, ніж відстань до поточної клітини. Якщо це так, то ця сусідня клітина стає новою поточною точкою.

Знайдена нова поточна точка додається до списку шляху.

Процес триває, доки не буде досягнуто початкової точки.

Шлях повертається у зворотному порядку, щоб він починався з початкової точки та закінчувався кінцевою.

Функція для одержання сусідніх комірок

def get_neighbors(cell):

```
        neighbors = []
        row, col = cell
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i == 0 and j == 0:
                    continue
                new_row, new_col = row + i, col + j
                if 0 <= new_row < rows and 0 <= new_col < cols:
                    neighbors.append((new_row, new_col))
        return neighbors
```

Ця функція get_neighbors отримує сусідні клітини для заданої клітини cell. Далі представлено як вона працює:

Створюється порожній список neighbors, що міститиме сусідні клітини.

Вилучаються координати рядка та стовпця з переданої клітини cell.

Запускаються вкладені цикли, які проходять по сусідніх рядках та стовпчиках від -1 до 1.

У кожній ітерації перевіряється, чи не є поточна ітерація центральною клітиною (тобто зміщення i та j рівні 0). Якщо так, то ітерація пропускається, оскільки ми хочемо додавати поточну клітинку до списку сусідів.

Обчислюються нові координати new_row та new_col для сусідньої клітини, додаючи зміщення із зовнішніх циклів до поточних координат.

Перевіряється, чи знаходяться нові координати в межах карти місцевості. Якщо так, то ця клітина вважається сусідньою, та її координати додаються до списку сусідів.

Після завершення всіх ітерацій циклів повертається перелік сусідніх клітин.

Ця функція важлива для визначення сусідніх клітин, які будуть використовуватися в алгоритмі хвильового розповсюдження для обчислення відстаней до перешкод.

```
# Функція виконання хвильового алгоритму
def wavefront_algorithm(grid, start, goal):
    rows, cols = grid.shape
    distances = np.full((rows, cols), np.inf)
    distances[start[0], start[1]] = 0
    queue = [start]
    while queue:
        current = queue.pop(0)
        neighbors = get_neighbors(current)
        for neighbor in neighbors:
            if grid[neighbor[0], neighbor[1]] == 0 and distances[neighbor[0], neighbor[1]] == np.inf:
                distances[neighbor[0], neighbor[1]] = distances[current[0], current[1]] + 1
                queue.append(neighbor)
    return distances
```

Ця функція wavefront_algorithm виконує хвильовий алгоритм на карті місцевості з перешкодами. Далі представлено як вона працює:

Створюється масив дистанції, який буде містити відстані від кожної клітини до найближчої перешкоди. Спочатку всі значення встановлюються на нескінченність, крім початкової точки, у якої відстань встановлюється в нуль.

Створюється черга queue та початкова точка start додається до неї.

Запускається основний цикл, який триває, доки черга не спорожніє.

Витягується поточна клітина з черги.

Виходять сусідні клітини для поточної клітини.

Для кожної сусідньої клітини перевіряється, чи є вона прохідною (тобто, дорівнює нулю) і чи має вона нескінченну відстань до перешкоди. Якщо так, то відстань до цієї сусідньої клітини встановлюється як відстань до поточної клітини плюс один, і ця сусідня клітина додається в чергу [221-225].

Після завершення циклу повертається масив distances, що містить обчислені відстані від кожної клітини до найближчої перешкоди.

```
# Параметри карти місцевості
rows = 10
cols = 10
start = (0, 0)
goal = (9, 9)
obstacles = [(3, 3), (4, 5), (5, 7),(6, 6), (6, 7), (6, 8), (6, 9),(7,5),(6,5)]
```

Цей фрагмент коду є визначенням параметрів карти місцевості, яка буде використовуватися для побудови маршруту мобільного робота. Зокрема:

rows та cols визначають кількість рядків та стовпців на карті місцевості.

start і goal вказують початкову та кінцеву точки маршруту, відповідно. Вони представлені у вигляді кортежів з координатами (рядок, стовпець).

obstacles є переліком координат перешкод на карті місцевості.

Ці параметри необхідні для створення карти місцевості з перешкодами та для виконання алгоритмів пошуку шляху, таких як хвильовий алгоритм, який використовується для визначення оптимального маршруту від початкової точки до кінцевої, враховуючи розташування перешкод.

```
# Створення карти місцевості
grid = create_map(rows, cols, obstacles)
```

Цей фрагмент коду викликає функцію create_map() для створення карти місцевості на основі наданих параметрів rows, cols та obstacles. Функція

create_map() створює матрицю, що є картою місцевості, де кожна комірка визначає стан відповідної клітини на карті. У цій матриці перешкоди позначаються значенням 1, а вільні комірки – значенням 0.

Отримана матриця grid використовуватиметься для виконання різних операцій, таких як виконання алгоритмів пошуку шляху чи візуалізація карти місцевості.

Виконання хвильового алгоритму

distances = wavefront_algorithm(grid, start, goal)

Цей фрагмент коду виконує хвильовий алгоритм на створеній карті grid з початковою точкою start та кінцевою точкою goal. Хвильовий алгоритм працює шляхом поширення хвилі з початкової точки до кінцевої. На кожному кроці алгоритм визначає відстань від кожної комірки до початкової точки.

Відстань зберігаються в матриці distances, де кожен елемент являє собою відстань від відповідної комірки до початкової точки. Значення у матриці distances використовуються для визначення оптимального шляху до мети

Побудова оптимального шляху

path = build_path(distances, start, goal)

Цей фрагмент коду будує оптимальний шлях від початкової точки start до кінцевої точки goal з урахуванням результатів хвильового алгоритму.

Функція build_path() приймає матрицю дистанційних відстаней, початкову і кінцеву точки, і повертає список координат, які являють собою оптимальний шлях від початкової точки до кінцевої.

Отриманий шлях зберігається у змінній path і використовуватиметься для візуалізації на карті місцевості.

Візуалізація карти місцевості з перешкодами та оптимальним шляхом

visualize_map(grid, distances, start, goal, path)

Цей фрагмент коду виконує візуалізацію карти місцевості з урахуванням перешкод та оптимального шляху від початкової точки до кінцевої.

Функція visualize_map() приймає такі аргументи:

grid: матриця, що є картою місцевості з перешкодами.

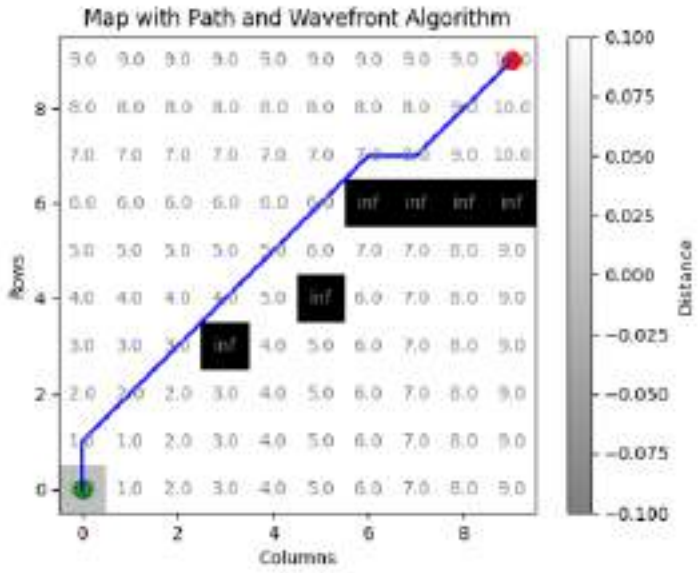
distances: матриця відстаней від початкової точки до кожної клітини на карті.

start і goal: координати початкової та кінцевої точок відповідно.

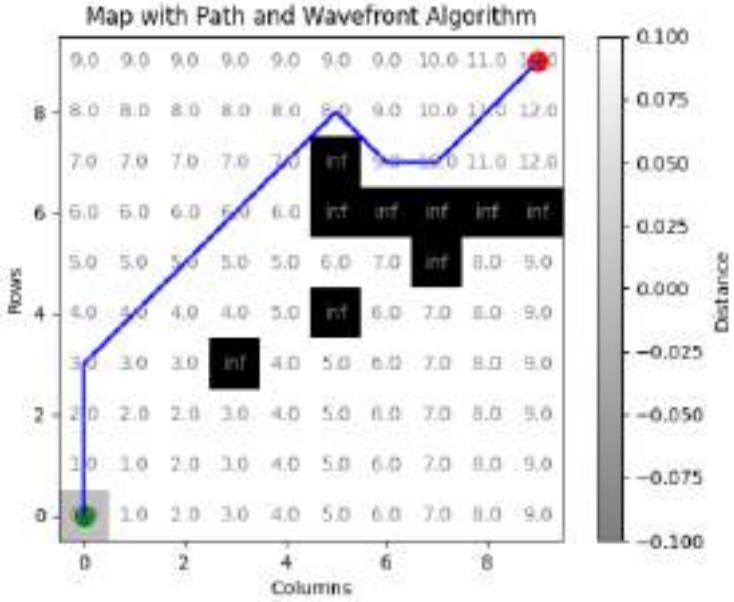
path: список координат, що становлять оптимальний шлях від початкової точки до кінцевої.

В результаті виклику цієї функції буде показано зображення карти місцевості, де перешкоди відображені білим кольором, оптимальний шлях відзначений синім кольором, а числові значення в кожній комірці вказують на відстань від початкової точки.

Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна. Результати роботи програми знаходження оптимального маршруту з використанням хвильового алгоритму (зліва направо) представлені на рисунку 4.4



а)



в)

Рисунок 4.4 - Результати роботи програми побудови оптимального маршруту із використанням хвильового алгоритму (зліва направо)

Хвильовий алгоритм, застосований для побудови оптимального маршруту для мобільних роботів з рухом ліворуч, є ефективним методом планування шляху в статичних і частково динамічних середовищах [226-230]. Він заснований на ідеї поширення хвиль від початкової точки до мети, що дозволяє визначити оптимальний шлях з урахуванням перешкод та обмежень.

Перевагою хвильового алгоритму для руху ліворуч праворуч є його простота і зрозумілість, що робить його застосовним для широкого кола користувачів без глибоких знань у галузі планування маршрутів. Він дозволяє роботу безпечно та ефективно досягати мети, уникаючи зіткнень із перешкодами [231-234].

Проте хвильовий алгоритм також має обмеження. Він припускає, що середовище статичне і не враховує динамічних змін, таких як поява нових перешкод або зміна умов прохідності. Це може призвести до неоптимальних або небезпечних маршрутів у разі змін у середовищі.

Для успішного застосування хвильового алгоритму в русі ліворуч рекомендується регулярно оновлювати карту перешкод і перераховувати маршрут при змінах в середовищі. Також важливо враховувати швидкість та можливості робота при виборі оптимального шляху.

Загалом хвильовий алгоритм є корисним інструментом для побудови оптимального маршруту для мобільних роботів з рухом зліва направо, який забезпечує швидку адаптацію до змін у середовищі та ефективне переміщення робота. Однак для забезпечення безпеки та оптимальності маршруту необхідно враховувати динамічні зміни та особливості навколишнього середовища. Приклад коду реалізації представлено у додатку З.

4.5 Побудови маршруту на основі хвильового алгоритму в динамічному просторі

Побудова маршруту для мобільного робота на основі хвильового алгоритму (див. підрозділ 4.3) у динамічному просторі, де перешкоди генеруються випадковим чином, включає декілька особливостей та проблем, які необхідно враховувати [235-240]:

- динамічне оновлення карти місцевості, оскільки перешкоди генеруються випадковим чином, необхідно регулярно оновлювати карту

місцевості для відображення змін. Це може включати виявлення нових перешкод, переміщення або видалення існуючих перешкод;

- адаптація хвильового алгоритму, що має бути здатний адаптуватися до змін на карті місцевості. Це може вимагати перерахунку відстаней та оновлення шляху в реальному часі при виявленні нових перешкод чи змін до існуючих;

- виявлення та уникнення динамічних перешкод, мобільний робот повинен бути здатний виявляти та уникати динамічних перешкод, які можуть з'являтися на його шляху. Це може включати використання сенсорів, таких як лазерний далекомір або камери, для виявлення перешкод в реальному часі;

- обробка хибних спрацьовувань, у цьому рішенні випадкове генерування перешкод може призвести до хибним спрацьовуванням сенсорів, що може спотворити карту місцевості та призвести до неправильного вибору шляху. Необхідно реалізувати механізми фільтрації помилкових спрацьовувань та корекції карти місцевості за необхідності;

- планування шляху у реальному часі, при динамічному оточенні мобільний робот повинен швидко перераховувати шлях у реальному часі у разі змін на карті місцевості чи появи нових перешкод. Це вимагає ефективних алгоритмів планування шляху та швидкого обчислення нового маршруту;

Успішна реалізація системи побудови маршруту для мобільного робота в динамічному просторі потребує ретельного аналізу та обліку всіх перерахованих вище особливостей, а також використання адаптивних алгоритмів і технологій для забезпечення надійної та ефективної навігації [241-246].

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import random
```

Цей фрагмент коду використовується для імпорту необхідних модулів та бібліотек для роботи з числовими даними, створення графіків та візуалізації даних з використанням бібліотеки matplotlib.

import numpy as np: Цей рядок імпортує бібліотеку NumPy та задає її псевдонім np. NumPy - це бібліотека Python для роботи з масивами, матрицями та високорівневими математичними функціями для операцій із цими структурами даних.

import matplotlib.pyplot as plt: Цей рядок імпортує модуль pyplot із бібліотеки matplotlib і задає його псевдонім plt. matplotlib.pyplot надає функції для створення графіків та візуалізації даних у Python.

from matplotlib.colors import ListedColormap: Цей рядок імпортує клас ListedColormap з модуля colors бібліотеки matplotlib. ListedColormap використовується для створення власних колірних карт для візуалізації даних на графіках.

import random: Цей рядок імпортує модуль random, який надає функції для роботи зі випадковими числами. Цей модуль може використовуватися для створення випадкових даних або для роботи з процесами, де потрібна випадковість.

```
# Функція візуалізації карти місцевості
def visualize_map(grid, distances, start, goal, path):
    cmap = ListedColormap(['white', 'black', 'gray'])  # Білий фон, чорні перешкоди, сірі значення хвильового алгоритму
    plt.figure(figsize=(8, 8))
    plt.imshow(grid, cmap=cmap, origin='lower')
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
```

```
                plt.text(j, i, f'{distances[i, j]:.0f}', ha='center', va='center', color='black' if grid[i, j] == 1 else 'gray')
        plt.plot(start[1], start[0], 'go', markersize=10)  # Початкова точка
        plt.plot(goal[1], goal[0], 'ro', markersize=10)    # Кінцева точка
        if path:
            path_x, path_y = zip(*path)
            plt.plot(path_y, path_x, 'b-', linewidth=2)    # Оптимальний шлях
        plt.title('Map with obstacles, wavefront algorithm, and path')
        plt.xlabel('Columns')
        plt.ylabel('Rows')
        plt.show()
```

Цей код визначає функцію visualize_map, яка використовується для візуалізації карти місцевості з перешкодами, значеннями хвильового алгоритму, початковою та кінцевою точками, а також оптимальним шляхом.

Аргументи функції:

grid: двовимірний масив, що є картою місцевості з перешкодами (значення 1 позначають перешкоди, а значення 0 - вільні осередки).

distances: двовимірний масив з відстанями, отриманими внаслідок роботи хвильового алгоритму.

start: кортеж координат початкової точки.

goal: кортеж координат кінцевої точки.

path: список кортежів координат, що становлять оптимальний шлях від початкової до кінцевої точки.

Ця функція використовує бібліотеку matplotlib для створення графічного представлення карти місцевості та хвильового алгоритму, а також для відображення початкової та кінцевої точок та оптимального шляху.

Коментарі в коді пояснюють, які елементи відображаються на графіку: перешкоди, значення хвильового алгоритму, початкова та кінцева точки, а також оптимальний шлях (якщо він заданий).

Функція виконання хвильового алгоритму

```
def wavefront_algorithm(grid, start, goal):
    rows, cols = grid.shape
    distances = np.full((rows, cols), np.inf)
    distances[start[0], start[1]] = 0
    queue = [start]
    while queue:
        current = queue.pop(0)
        neighbors = get_neighbors(current, rows, cols)
        for neighbor in neighbors:
            if grid[neighbor[0], neighbor[1]] == 0 and distances[neighbor[0], neighbor[1]] == np.inf:
                distances[neighbor[0], neighbor[1]] = distances[current[0], current[1]] + 1
                queue.append(neighbor)
    return distances
```

Цей код визначає функцію wavefront_algorithm, яка реалізує алгоритм поширення хвилі (Wavefront Algorithm) для пошуку оптимального шляху від початкової точки до кінцевої на карті з перешкодами.

Аргументи функції:

grid: двовимірний масив, що є картою місцевості з перешкодами (значення 1 позначають перешкоди, а значення 0 - вільні осередки).

start: кортеж координат початкової точки.

goal: кортеж координат кінцевої точки.

Ця функція використовує алгоритм BFS (пошук в ширину) для пошуку всіх точок, що досяжні від початкової точки, підраховуючи при цьому відстані до кожної точки на карті.

Значення, що повертається:

distances: двомірний масив, в якому зберігаються відстані від початкової точки до кожного осередку на карті місцевості.

```
# Функція для одержання сусідніх комірок
```

```
def get_neighbors(cell, rows, cols):
    neighbors = []
    row, col = cell
    for i in range(-1, 2):
        for j in range(-1, 2):
            if i == 0 and j == 0:
                continue
            new_row, new_col = row + i, col + j
            if 0 <= new_row < rows and 0 <= new_col < cols:
                neighbors.append((new_row, new_col))
    return neighbors
```

Цей фрагмент коду визначає функцію get_neighbors, яка використовується для отримання сусідніх осередків щодо заданого осередку на карті місцевості.

Аргументи функції:

cell: кортеж координат комірки.

rows: кількість рядків на карті місцевості.

cols: кількість стовпців на карті місцевості.

Ця функція сканує околицю заданої комірки розміром 3x3 (включаючи саму комірку) та повертає список координат сусідніх комірок, що задовольняють умовам перебування в межах карти.

Функція виключає саму комірку зі списку сусідів, а також враховує межі карти, щоб уникнути виходу за межі.

Коментарі в коді допомагають зрозуміти його логіку та призначення кожної частини.

```
# Параметри карти місцевості
rows = 20
cols = 20
start = (0, 0)
goal = (rows - 1, cols - 1)
```

Цей фрагмент коду визначає параметри карти місцевості:

rows: кількість рядків на карті місцевості.

cols: кількість стовпців на карті місцевості.

start: кортеж координат початкової точки на карті.

goal: кортеж координат кінцевої точки на карті.

В даному випадку:

rows та cols встановлені на 20, що означає, що карта місцевості має розмір 20x20.

start задає початкову точку карти місцевості у верхньому лівому куті (0, 0).

goal задає кінцеву точку карти місцевості в правому нижньому кутку, що відповідає координатам (19, 19), оскільки індексація починається з 0.

Ці параметри можуть бути налаштовані відповідно до конкретних вимог або характеристик карти місцевості для конкретної задачі або програми.

```
# Створення порожньої карти місцевості
grid = np.zeros((rows, cols), dtype=int)
```

Цей фрагмент коду створює порожню карту місцевості grid розміром rows на cols за допомогою бібліотеки NumPy.

np.zeros((rows, cols), dtype=int): Створює двовимірний масив (матрицю) розміром rows на cols, заповнений нулями, використовуючи функцію zeros із бібліотеки NumPy. Параметр dtype=int вказує на те, що тип даних елементів матриці буде цілим.

Ця порожня карта місцевості буде використовуватися для подальшої побудови карти з перешкодами та виконання алгоритмів пошуку шляху.

```
# Рандомне додавання перешкод на карту місцевості
for _ in range(rows * cols // 5):
    row, col = random.randint(0, rows - 1), random.randint(0, cols - 1)
    grid[row, col] = 1
```

Цей фрагмент коду випадково додає перешкоди на карту місцевості. grid.

for _ in range(rows * cols // 5): Цикл виконується rows*cols//5 разів, що дозволяє випадковим чином додати перешкоди в 20% комірок карти місцевості. Це робиться шляхом перебору випадкових координат осередків на карті місцевості.

row, col = random.randint(0, rows - 1), random.randint(0, cols - 1): Генерує випадкові координати row (рядки) та col (стовпця) у межах діапазону розміру карти місцевості.

grid[row, col] = 1: Встановлює значення комірки з координатами row і col рівним 1, що означає наявність перешкоди у цій комірці.

Цей процес випадкового додавання перешкод допомагає створити різноманітність сценаріїв для тестування алгоритмів пошуку шляху на карті місцевості.

Виконання хвильового алгоритму

distances = wavefront_algorithm(grid, start, goal)

Цей фрагмент коду виконує хвильовий алгоритм на карті місцевості grid з використанням початкової точки start та кінцевої точки goal.

wavefront_algorithm(grid, start, goal): Викликає функцію wavefront_algorithm з передачею параметрів grid, start та goal. Ця функція реалізує алгоритм поширення хвилі для пошуку оптимального шляху від початкової точки до кінцевої карти з перешкодами.

Результат виконання алгоритму зберігається в змінній distances. Це двовимірний масив, в якому зберігаються відстані від початкової точки до кожного осередку на карті місцевості.

Тепер у змінній distances містяться відстані від початкової точки до кожного осередку карти місцевості, розраховані за допомогою хвильового алгоритму. Ця відстань використовується для візуалізації результатів на карті місцевості.

Побудова оптимального шляху

```
path = [(goal[0], goal[1])]
current = goal
while current != start:
    neighbors = get_neighbors(current, rows, cols)
    for neighbor in neighbors:
        if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:
            current = neighbor
            path.append(current)
            break
path = path[::-1]
```

Цей фрагмент коду будує оптимальний шлях від кінцевої точки goal до початкової точки start на основі результатів отриманих в результаті виконання хвильового алгоритму.

path = [(goal[0], goal[1])]: Починаємо побудову шляху з кінцевої точки, додаючи її координати до списку шляху path.

current = goal: Встановлюємо поточну позицію кінцевої точки.

while current != start:: Запускаємо цикл, який буде виконуватися, поки поточна позиція не стане рівною початковій точці.

neighbors = get_neighbors(current, rows, cols): Отримуємо сусідні осередки для поточної позиції.

for neighbor in neighbors:: Перебираємо сусідні комірки.

if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:: Перевіряємо, якщо відстань до сусідньої комірки менша за відстань до поточної комірки.

current = neighbor: Оновлюємо поточну позицію на позицію сусіднього осередку.

path.append(current): Додаємо поточну позицію до списку шляху.

break: Перериваємо цикл після знаходження наступного осередку шляху.

path = path[::-1]: Розгортаємо список шляху, щоб він починався з початкової точки і закінчувався кінцевою точкою.

Тепер змінна path містить список координат, що становлять оптимальний шлях від початкової точки до кінцевої.

Візуалізація карти місцевості з перешкодами, значеннями хвильового алгоритму та маршрутом

visualize_map(grid, distances, start, goal, path)

Цей фрагмент коду викликає функцію visualize_map для візуалізації карти місцевості із перешкодами, значеннями хвильового алгоритму та оптимальним маршрутом.

Параметри функції visualize_map:

grid: мапа місцевості з перешкодами.

distances: відстані, розраховані внаслідок виконання хвильового алгоритму.

start: початкова точка.

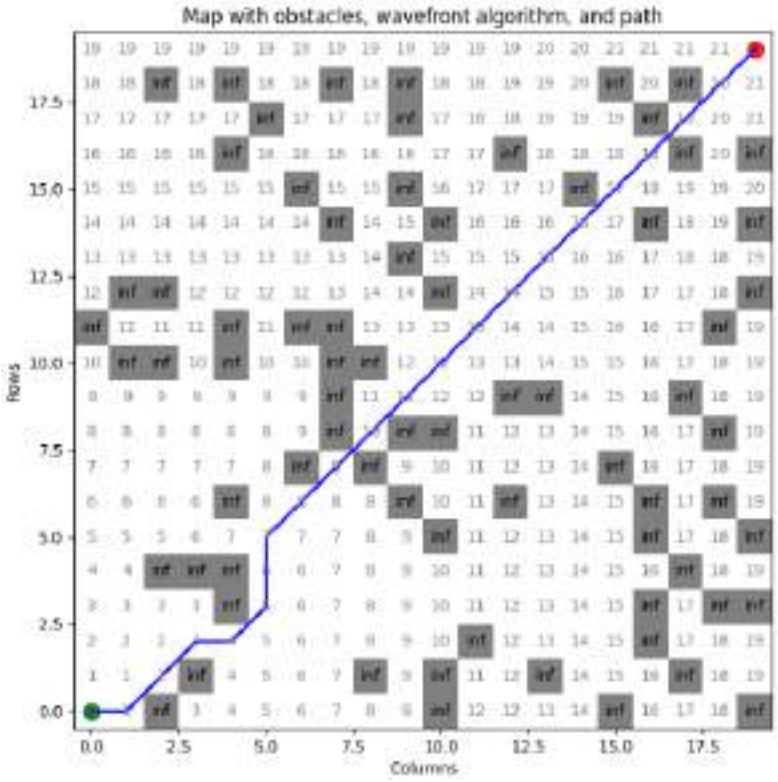
goal: кінцева точка.

path: оптимальний маршрут від початкової до кінцевої точки.

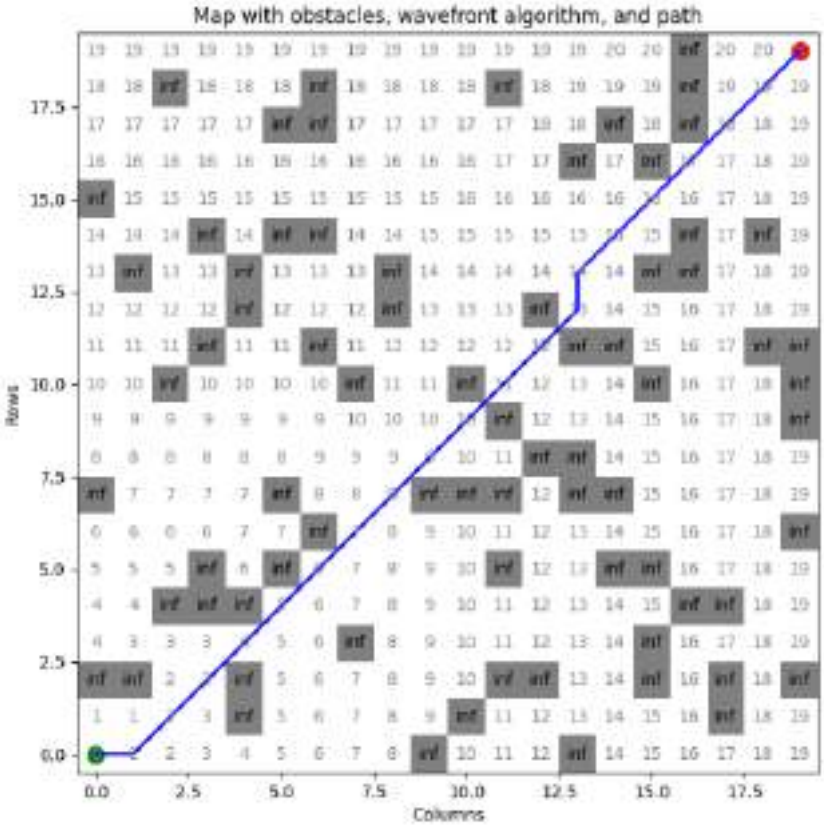
Цей виклик функції призведе до відображення графічного представлення карти місцевості з перешкодами, значеннями хвильового алгоритму та оптимальним маршрутом на графіку.

Візуалізація допоможе зрозуміти, як алгоритм упорався з пошуком оптимального шляху через перешкоди на карті місцевості.

Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна. Результати роботи програми побудови маршруту на базі хвильового алгоритму в динамічному просторі представлені на малюнку 4.5



а)



в)

Рисунок 4.5 - Результати роботи програми побудови маршруту на базі хвильового алгоритму у динамічному просторі

Хвильовий алгоритм (Wavefront Algorithm) є ефективним методом планування маршрутів для мобільних роботів у динамічному просторі. Він заснований на ідеї поширення хвиль по простору від початкової точки до всіх доступних точок, що дозволяє визначити оптимальний маршрут з огляду на поточні умови та перешкоди [247-250].

Перевагою хвильового алгоритму є його простота і ефективність у вирішенні завдань планування маршрутів у середовищах, що динамічно змінюються. Він дозволяє роботу швидко адаптуватися до змін у навколишньому середовищі та вибирати оптимальний шлях навіть за наявності перешкод [251-253].

Проте хвильовий алгоритм має обмеження. Він передбачає, що середовище статичне і не враховує динамічні зміни, такі як перешкоди, що рухаються, або зміна умов прохідності території. Це може призвести до неоптимальних або небезпечних маршрутів у разі динамічних змін у середовищі [254-256].

Для успішного застосування хвильового алгоритму динамічному просторі рекомендується регулярно оновлювати карту перешкод і перераховувати маршрут при зміні умов. Також важливо враховувати швидкість та можливості робота при виборі оптимального шляху.

Загалом хвильовий алгоритм є корисним інструментом для планування маршрутів мобільних роботів у динамічному просторі, який забезпечує швидку адаптацію до змін у середовищі та ефективне переміщення робота. Однак для забезпечення безпеки та оптимальності маршруту необхідно враховувати динамічні зміни та особливості навколишнього середовища. Приклад коду реалізації представлено у додатку И.

4.6 Побудова 3-вимірного маршруту на базі алгоритму PRM

Алгоритм PRM (Probabilistic Roadmap) – це потужний метод планування маршрутів для мобільних роботів у тривимірному просторі [257-

261]. Його основна перевага полягає в тому, що він дозволяє будувати шляхи у складних та динамічних середовищах, враховуючи різні обмеження та перешкоди. Принцип роботи алгоритму у тому, що він створює граф, де вершини є допустимі позиції для робота, а ребра - допустимі шляхи між цими позиціями.

Одним із ключових етапів роботи алгоритму є генерація випадкових точок у просторі, які можуть бути використані як потенційні вершини графа. Потім для кожної точки перевіряється її з'єднання з іншими точками через перевірку колізій з перешкодами. Якщо дві точки можуть бути з'єднані, між ними додається ребро в граф [262-265].

Одним із недоліків алгоритму PRM є те, що він може вимагати великої кількості точок для побудови графа у складних середовищах. Це може призвести до високої обчислювальної складності та вимог до пам'яті. Крім того, PRM не гарантує знаходження оптимального шляху, оскільки він працює на основі випадкових вибірок та може пропустити найкращі шляхи.

Тим не менш, алгоритм PRM залишається популярним вибором для планування маршрутів у тривимірному просторі завдяки своїй ефективності та здатності працювати у різних середовищах. Він дозволяє враховувати складні форми перешкод і умови, що динамічно змінюються, що робить його застосовним для широкого спектру завдань планування руху мобільних роботів [266-269].

Нехай V - безліч вершин графа, де кожна вершина є крапкою в просторі, E - безліч ребер графа, де кожне ребро з'єднує дві вершини, якщо ці вершини можуть бути з'єднані без перетину з перешкодами, тоді:

$$G = (V, E) \tag{4.23}$$

Де: G - граф маршрутів;

V - безліч вершин графа;

E - безліч ребер графа.

Позначимо через P_{start} та P_{end} - початкова та кінцева точки відповідно.

Тоді основні кроки PRM алгоритму можуть бути представлені наступним чином:

- генерація точок

$$V = \{P_{start}, P_{end}, P_1, P_2, ..., P_i, ..., P_n\} \quad (4.24)$$

Де: P_{start} - початкова точка;

P_{end} - кінцева крапки;

P_i - випадково згенеровані точки.

- пошук ребер

$$E = \{(P_i, P_j) \mid C(P_i, P_j) = False\} \quad (4.25)$$

Де: C - функція перевірки колізій між двома точками або їх околицями та перешкодами;

P_i, P_j - випадково згенеровані точки.

- пошук шляху (можна використовувати стандартні алгоритми пошуку шляху, такі як алгоритм Дейкстри або A*, для знаходження оптимального шляху між початковою та кінцевою точками).

Це можна уявити наступним чином, нехай P_{cur} - поточна точка шляху, $N(P_{cur})$ - безліч усіх сусідів точки P_{cur}, $C(P_{cur}, P_{next})$ - функція, що визначає вартість переходу з точки P_{cur} у крапку P_{next}, нехай $D(S_{start}, P_{end})$ - довжина шляху від точки P_{start} до точки P_{end}. Тоді пошук шляхів можна описати в такий спосіб:

- ініціалізація

$$P_{cur} = P_{start}, Path = \{P_{start}\}, Visited = \{P_{start}\} \quad (4.26)$$

За умов що доки $P_{cur} \neq P_{end}$ то:

1. Обираємо $P_{next} \in N(P_{cur})$, таке що $P_{next} \notin Visited$ и $C(P_{cut}, P_{next}) \to \min$

2. Додаєм P_{next} до $Path$ та $Visited$

3. $P_{cur} = P_{next}$ і вертаємо $Path$

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random
```

Цей фрагмент коду містить імпорти необхідних бібліотек для роботи з тривимірною візуалізацією Python:

numpy (import numpy as np): NumPy - це бібліотека Python, яка додає підтримку великих багатовимірних масивів та матриць, разом із великою бібліотекою високорівневих математичних функцій для операцій із цими масивами.

matplotlib.pyplot (import matplotlib.pyplot as plt): Matplotlib - це бібліотека для створення графіків та візуалізації даних у Python. pyplot – це модуль Matplotlib, який надає інтерфейс для створення графіків та інших видів візуалізації.

mpl_toolkits.mplot3d (з mpl_toolkits.mplot3d import Axes3D): Цей модуль містить інструменти для створення тривимірних графіків у Matplotlib. Він надає клас Axes3D, який дозволяє створювати та працювати з тривимірними осями координат.

random (import random): Модуль random використовується для створення випадкових чисел та інших випадкових операцій.

Ці бібліотеки широко використовуються для візуалізації даних, побудови графіків, створення тривимірних моделей та інших завдань, пов'язаних із візуалізацією та аналізом даних. У цьому контексті, вони використовуються для візуалізації тривимірного простору з перешкодами та маршрутом для мобільного робота.

```
# Клас для представлення точки у 3D просторі
class Point:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
    def __repr__(self):
        return f"({self.x}, {self.y}, {self.z})"
```

Цей клас Point представляє крапку у тривимірному просторі. Він має три атрибути x, y та z, які представляють координати точки по осях X, Y та Z відповідно. Метод __repr__ визначений для повернення строкового представлення точки у форматі (x, y, z), що зручно для налагодження та відображення точок під час візуалізації.

```
# Функція перевірки перетину сфер (перешкод)
def check_collision(point, obstacles, radius):
    for obstacle in obstacles:
        if np.linalg.norm([point.x - obstacle.x, point.y - obstacle.y, point.z - obstacle.z]) < radius:
            return True
    return False
```

Ця функція check_collision використовується для перевірки перетину точки з перешкодами у вигляді сфер із заданим радіусом. Вона приймає три параметри:

point: об'єкт типу Point, що представляє точку в тривимірному просторі,

obstacles: перелік перешкод, кожна з яких представлена об'єктом типу Point,

radius: радіус сфери, що визначає відстань, на якій вважається, що точка перетинає перешкоду.

Функція проходить за списком obstacles і для кожної перешкоди обчислює відстань між точкою та перешкодою за допомогою функції np.linalg.norm(), яка обчислює евклідову відстань між двома точками у тривимірному просторі. Якщо ця відстань менша від заданого radius, функція повертає True, що означає перетин. Якщо жодна з перешкод не перетинається з точкою, функція поверне False.

```
# Генерація PRM шляху
def generate_prm_path(start, end, obstacles, num_samples, num_neighbors, radius):
    nodes = [start, end]
    for _ in range(num_samples):
        sample = Point(random.uniform(0, 10), random.uniform(0, 10), random.uniform(0, 10))
        if not check_collision(sample, obstacles, radius):
            nodes.append(sample)
    prm_path = [start]
    for node in nodes:
        if node != start and not check_collision(node, obstacles, radius):
            nearest_neighbors = sorted(nodes, key=lambda x: np.linalg.norm([x.x - node.x, x.y - node.y, x.z - node.z]))[:num_neighbors]
            prm_path.extend(nearest_neighbors)
    prm_path.append(end)
    return prm_path
```

Ця функція generate_prm_path використовується для генерації шляху за допомогою алгоритму PRM (Probabilistic Roadmap) для заданого початкового

та кінцевого вузлів, списку перешкод, кількості зразків (num_samples), кількості найближчих сусідів (num_neighbors) та радіусу колізії (radius).

Вона працює так:

Створює список вузлів nodes, який містить початковий та кінцевий вузли.

Генерує num_samples випадкових точок і додає їх до списку nodes, якщо вони не перетинаються перешкодами.

Для кожного вузла у списку nodes знаходить num_neighbors найближчих сусідів (виключаючи початковий вузол) і додає їх у шлях prm_path.

Повертає згенерований шлях prm_path, який починається з початкового вузла, містить проміжні вузли та закінчується кінцевим вузлом.

```
# Створення простору
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
```

Цей фрагмент коду використовується для створення тривимірного простору візуалізації. Він використовує бібліотеку Matplotlib для створення нового графічного вікна (plt.figure()) і додавання тривимірних осей (fig.add_subplot(111, projection='3d')), які будуть представляти тривимірні координати X, Y і Z. Потім він встановлює підписи для осей X, Y та Z за допомогою методів set_zlabel() відповідно.

```
# Початкова та кінцева точки
start = Point(0, 0, 0)
end = Point(10, 10, 10)
```

Цей код створює початкову точку start з координатами (0, 0, 0) та кінцеву точку end з координатами (10, 10, 10) у тривимірному просторі. Такі

точки використовуються для визначення початку та кінця маршруту мобільного робота або інших об'єктів у трьох вимірах.

```
# Генерація перешкод
num_obstacles = 20
obstacles = []
radius = 1
for _ in range(num_obstacles):
    obstacle = Point(random.uniform(0, 10), random.uniform(0, 10), random.uniform(0, 10))
    while check_collision(obstacle, obstacles, 2 * radius):
        obstacle = Point(random.uniform(0, 10), random.uniform(0, 10), random.uniform(0, 10))
    obstacles.append(obstacle)
```

Цей код генерує перешкоди у вигляді випадкових точок у тривимірному просторі. Він використовує кількість перешкод num_obstacles, радіус колізії radius та функцію check_collision для перевірки перетину з іншими перешкодами.

Цикл for створює num_obstacles перешкод таким чином:

- створює нову випадкову точку, що представляє перешкоду, з координатами в діапазоні від 0 до 10 кожної з осей X, Y і Z.

- перевіряє, чи ця перешкода не перетинається з вже існуючими перешкодами. Якщо перетинається, то генерує нову випадкову точку.

- після того, як знайдена непересічна перешкода, вона додається до списку obstacles.

Після виконання цього коду, список obstacles міститиме вказану кількість перешкод для використання надалі.

```
# Генерація PRM шляху
num_samples = 50
num_neighbors = 1
```

prm_path = generate_prm_path(start, end, obstacles, num_samples, num_neighbors, radius)

Цей код використовує раніше визначені початкову точку start, кінцеву точку end, список перешкод obstacles, а також параметри num_samples, num_neighbors та radius для генерації шляху за допомогою функції generate_prm_path.

Параметр num_samples визначає кількість випадкових точок, які будуть згенеровані для побудови графа PRM, num_neighbors – кількість найближчих сусідів, які враховуються при побудові графа, а radius – радіус колізії для перевірки перетину точок із перешкодами.

Після виконання цього коду змінна prm_path міститиме згенерований шлях від початкової до кінцевої точки з використанням алгоритму PRM.

Малювання простору з перешкодами

for obstacle in obstacles:

ax.plot([obstacle.x], [obstacle.y], [obstacle.z], color='black', marker='s')

Цей код використовується для відтворення перешкод у тривимірному просторі на графіку. Для кожної перешкоди зі списку obstacles він викликає метод plot об'єкта ax (осі координат) із зазначенням координат перешкоди по осях X, Y та Z. Перешкоди відображаються у вигляді чорних кубів (маркер 's').

Цей код слід помістити після створення простору та до відображення графіка, щоб перешкоди були додані на графік.

Відображення початкової та кінцевої точок

ax.scatter(start.x, start.y, start.z, color='g', marker='o')

ax.scatter(end.x, end.y, end.z, color='b', marker='o')

Цей код використовується для відображення початкової та кінцевої точок на графіку. Він викликає метод scatter об'єкта ax (осі координат) для відображення точок з координатами початкової точки start та кінцевої точки end. Початкова точка відображається зеленим кольором (color='g') та

маркером кола (marker='o'), а кінцева точка - синім кольором (color='b') та маркером кола (marker='o').

Цей код слід помістити після відтворення перешкод і до відображення графіка, щоб точки були додані на графік.

```
# Відображення маршруту
prm_path_x = [point.x for point in prm_path]
prm_path_y = [point.y for point in prm_path]
prm_path_z = [point.z for point in prm_path]
ax.plot(prm_path_x, prm_path_y, prm_path_z, color='g')
```

Цей код використовується для відтворення згенерованого шляху на графіку. Він створює списки prm_path_x, prm_path_y та prm_path_z, що містять координати X, Y та Z відповідно для кожної точки шляху prm_path, а потім викликає метод plot об'єкта ax для відображення шляху на графіку. Шлях відображається зеленим (color='g').

Цей код також слід помістити після відображення початкової та кінцевої точок та до відображення графіка, щоб шлях був доданий на графік.

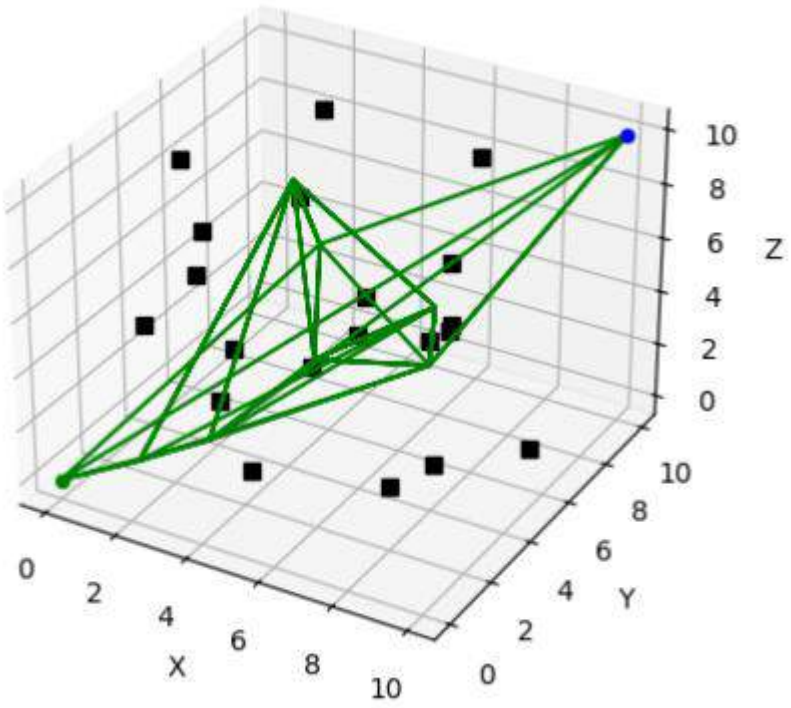
```
# Показати графік
plt.show()
```

Цей код використовується для відображення графіка з побудованим маршрутом та перешкодами. Він викликає функцію show() об'єкта plt (бібліотека Matplotlib), щоб показати вікно з тривимірним графіком.

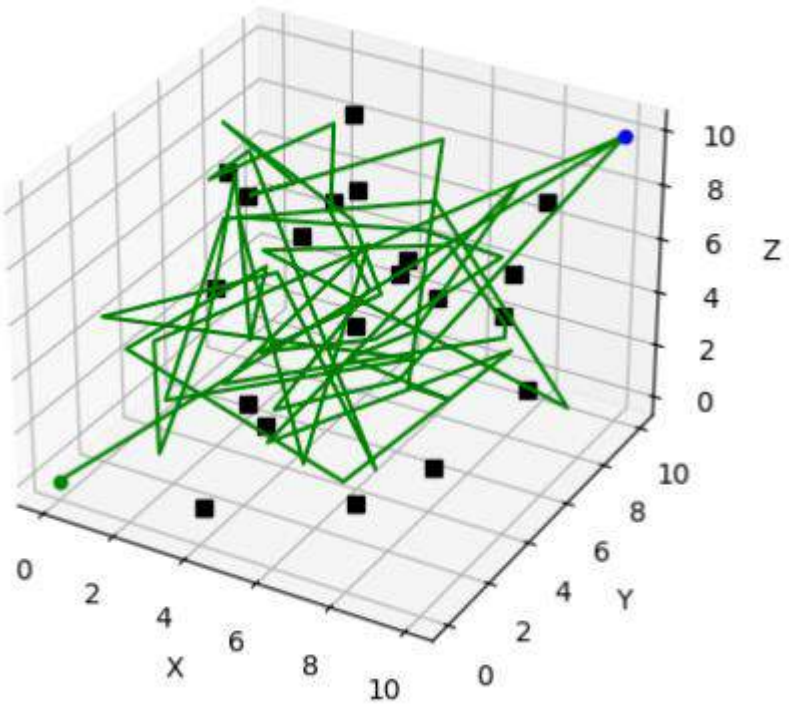
Після виконання цього коду відкриється вікно з тривимірним графіком, на якому будуть відображені початкова та кінцева точки, перешкоди у вигляді чорних кубів та згенерований маршрут між початковою та кінцевою точками.

Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна. Результати роботи програми побудови маршруту в 3-мірному

просторі на базі алгоритму PRM в динамічному просторі, представлені на рисунку 4.6.



а) (num_samples = 10; num_neighbors=5)



в) (num_samples = 50; num_neighbors=1)

Рисунок 4.6 - Результати роботи програми побудови маршруту у 3-вимірному просторі на базі алгоритму PRM

Алгоритм PRM (Probabilistic Roadmap) є ефективним підходом до планування маршрутів для мобільних роботів у тривимірному просторі. Він широко застосовується завдяки своїй здатності працювати в складних середовищах з перешкодами і умовами, що динамічно змінюються. Перевагою алгоритму є його імовірнісний характер, що дозволяє враховувати випадкові фактори та невизначеність у навколишньому середовищі [270-275].

Однією з ключових переваг PRM є можливість побудови шляху, що враховує форму перешкод та оптимальне використання доступного простору. Це робить його придатним для завдань, де необхідно уникати зіткнень із перешкодами та знаходити оптимальний маршрут із урахуванням різних обмежень.

Однак алгоритм PRM має деякі недоліки. Він може вимагати великої кількості обчислювальних ресурсів та часу для побудови графа маршрутів у складних середовищах. Крім того, PRM не гарантує знаходження оптимального шляху, оскільки заснований на випадкових вибірках точок і може пропустити ефективніші маршрути.

Для успішного застосування алгоритму PRM рекомендується провести ретельний аналіз середовища та його особливостей, щоб визначити оптимальні параметри генерації точок та перевірки колізій. Також важливо враховувати обмеження робота та особливості його руху під час побудови маршруту [276-280].

В цілому алгоритм PRM є потужним інструментом для планування маршрутів мобільних роботів у тривимірному просторі, який дозволяє враховувати складні умови середовища та забезпечувати безпечне та ефективне переміщення робота. Приклад коду реалізації представлено у додатку І.

4.6 Алгоритм D*

Алгоритм D* (D-зірка) — це алгоритм пошуку найкоротшого шляху у графі, який є модифікацією алгоритму A* та призначений для роботи з динамічними середовищами [281-285]. На відміну від A*, D* здатний адаптуватися до змін навколишнього середовища під час пошуку шляху. Він використовує дві основні структури даних: граф для представлення сітки та черга з пріоритетом для керування порядком обходу вершин.

Алгоритм D* (D-зірка) є потужним інструментом для побудови оптимальних маршрутів на карті місцевості для переміщення мобільних роботів. Ось кілька ключових особливостей його застосування [286-290]:

- алгоритм D* дозволяє мобільному роботу адаптуватися до змін у середовищі реального часу. Це особливо корисно у разі появи перешкод чи зміни плану руху.

- алгоритм D* має високу ефективність завдяки інкрементному оновленню інформації про шлях та можливість перерахунку шляху без перегляду всієї карти.

- алгоритм D* гарантує знаходження оптимального шляху між початковою та кінцевою точками, що важливо для економії часу та ресурсів.

- робота в реальному часі, завдяки своїй ефективності та здатності обробляти зміни в середовищі, D* може використовуватись для планування шляху в реальному часі без помітних затримок.

- простота реалізації, незважаючи на свою міць, алгоритм D * відносно простий в реалізації і може бути адаптований для різних типів роботів і середовищ.

- стійкість до шуму, алгоритм D* здатний ефективно керувати шумом у даних чи неповними знаннями про середовище завдяки своїй адаптивності.

- алгоритм D* дозволяє працювати з сітками довільної форми та складної топології, що робить його універсальним інструментом для різних завдань планування колії.

- алгоритм D* може враховувати різні чинники ризику під час планування маршруту, такі як швидкість руху, ймовірність появи перешкод та інші.

- алгоритм D* широко застосовується у реальних умовах, як приклад автономна навігація роботів у промислових комплексах, магазинах чи складах.

- алгоритм D* може бути розширений та доопрацьований для врахування додаткових факторів або покращення продуктивності у конкретних сценаріях використання.

У результаті основні кроки алгоритму D* можна наступним чином. Нехай безліч вершин U містить усі вершини графа, тоді позначимо через $g(u)$ - вартість шляху від початкової вершини s до вершини u. Позначимо, що спочатку $g(s)=0$, для всіх інших вершин $g(s)=\inf$. Введемо позначення $rhs(u)$ - " правильна " (актуальна) вартість шляху від вершини u до кінцевої точки «цілі» $goal$.

Для оновлення правильної вартості для всіх вершин $u \in U$ можна уявити так:

$$rhs(u) = \min(g(v)) + c(u,v) \tag{4.27}$$

Де: v - сусід u;

$c(u,v)$ - вартість переходу від u до v.

Наступним кроком проводимо обчислення ключа до вершини u

$$k(u) = [\min(g(u)), rhs(u) + h(start,u) + km] \tag{4.28}$$

Де: km - деяка евристична константа;

$h(start,u)$ - евристична функція оцінки вартості від $start$ до u.

Тоді полиця $U \neq 0$ виконуємо наступний набір дій:

- вибираємо вершину u з найменшим ключем $k(u)$;

- якщо $g(u) > rhs(u)$, то відповідно $g(u)$ стає рівним $rhs(u)$, інакше оновлюємо $g(u)$ і всіх сусідів u. Після цього проводимо оновлення rhs для всіх сусідів u, якщо $g(start) = rhs(start)$, алгоритм завершує роботу.

Після завершення алгоритму шлях можна відновити, рухаючись від мети до старту за найменшою вартістю, при зміні вартості ребра, проводитись перерахунок rhs та ключі вершин, через які проходить змінене ребро.

Для перевірки правильності міркувань розробимо програму для моделювання роботи побудови оптимізованого маршруту руху мобільного робота з використанням об'єктно орієнтованої мови Python у середовищі розробки PhCham 2022.2.3. Фрагменти програмної реалізації наведені нижче:

```
import random
import matplotlib.pyplot as plt
```

Цей фрагмент коду використовується для імпорту двох модулів у Python:

random: Модуль random використовується для створення випадкових чисел. Він містить функції, такі як random(), яка повертає випадкове число з плаваючою точкою в інтервалі [0, 1], та randint(a, b), яка повертає випадкове ціле число в інтервалі [a, b].

matplotlib.pyplot as plt: Модуль matplotlib.pyplot використовується для створення графіків та візуалізації даних у Python. Він містить функції для створення різних типів графіків, керування їх зовнішнім виглядом та відображення результатів. У цьому контексті він, ймовірно, використовуватиметься для візуалізації карти місцевості та шляхи на ній.

```
# Генерація картки з перешкодами
def generate_map(grid_size, num_obstacles):
    obstacles = set()
    for _ in range(num_obstacles):
```

```
            obstacle = (random.randint(0, grid_size-1), random.randint(0, grid_size-1))
            obstacles.add(obstacle)
        return obstacles
```

Цей фрагмент коду є функцією `generate_map`, яка створює карту з перешкодами. Вхідні параметри функції `grid_size` та `num_obstacles` визначають розмір сітки та кількість перешкод відповідно. Функція використовує модуль `random` для генерації випадкових координат перешкод у межах розміру сітки. Кожна перешкода представлена у вигляді кортежу координат `(x, y)` і додається до безлічі `obstacles`. Функція повертає цю множину, що представляє карту з перешкодами.

```
    # Функція для відображення карти з перешкодами та шляхом
    def display_map(grid_size, obstacles, path):
        for i in range(grid_size):
            for j in range(grid_size):
                if (i, j) in obstacles:
                    plt.scatter(i, j, color='red', s=100, edgecolors='black', marker='s') # красный квадрат для препятствий
                elif (i, j) in path:
                    plt.scatter(i, j, color='green', s=100, edgecolors='black')  # зеленая точка для пути
                else:
                    plt.scatter(i, j, color='white', s=100, edgecolors='black')
        plt.xlim(-1, grid_size)
        plt.ylim(-1, grid_size)
        plt.gca().invert_yaxis()
        plt.gca().set_aspect('equal', adjustable='box')
        plt.show()
```

Цей фрагмент коду представляє функцію `display_map`, яка відображає карту з перешкодами та шляхом на ній. Вхідні параметри функції включають

розмір сітки `grid_size`, безліч координат перешкод `obstacles` та список координат шляху `path`.

Функція використовує цикли `for` для проходу по всіх координатах сітки. Для кожної координати перевіряється приналежність до множини `obstacles` або `path`. Якщо координата належить `obstacles`, то вона відображається червоним квадратом за допомогою функції `scatter` з `matplotlib`. Якщо координата належить `path`, вона відображається зеленою точкою. Координати, що не належать жодній з множин, відображаються білим кольором.

Нарешті, функція налаштовує межі графіка, інвертує (щоб координати було відображено правильно) і встановлює співвідношення сторін рівним 1 (щоб графік був квадратним). Функція `show` викликає відображення графіка.

```
# Функція пошуку шляху методом D*
def d_star(start, goal, obstacles):
    grid = {}
    for i in range(grid_size):
        for j in range(grid_size):
            grid[(i, j)] = float('inf')
    grid[start] = 0
    open_set = {start}
    came_from = {}
```

Ця функція є алгоритмом D* для пошуку шляху на сітці з перешкодами. Вона ініціалізує сітку grid з нескінченними значеннями кожної клітини і встановлює значення початкової клітини start в 0.

open_set – це безліч клітин, які потрібно розглянути. На початку алгоритму воно містить лише початкову клітину.

came_from - це словник, який міститиме інформацію про те, з якої клітини прийшли до поточної клітини.

Далі алгоритм поступово оновлюватиме значення в grid, ітеративно розглядаючи клітини з open_set, щоб знайти оптимальний шлях до мети goal.

```
while open_set:
    current = min(open_set, key=lambda x: grid[x])
    open_set.remove(current)
    if current == goal:
        break
```

У цьому фрагменті коду реалізується основний цикл алгоритму D*, який працює доти, доки є клітини в множині `open_set` для розгляду. На кожній ітерації вибирається клітина `current` з `open_set` з найменшою вартістю `grid[x]`. Потім ця клітина видаляється з `open_set`.

Якщо поточна клітина `current` дорівнює цільовій клітині `goal`, то цикл завершується, оскільки знайшли оптимальний шлях до мети. У цьому випадку алгоритм виходить із циклу за допомогою оператора `break`. Якщо цільової клітини не досягнуто, алгоритм продовжує свою роботу, розглядаючи сусідні клітини для поточної клітини та оновлюючи їх вартості, якщо знаходить більш вигідний шлях.

```
neighbors = [(current[0]+1, current[1]), (current[0]-1, current[1]), (current[0], current[1]+1), (current[0], current[1]-1)]
        for neighbor in neighbors:
            if 0 <= neighbor[0] < grid_size and 0 <= neighbor[1] < grid_size and neighbor not in obstacles:
                tentative_g_score = grid[current] + 1
                if tentative_g_score < grid[neighbor]:
                    came_from[neighbor] = current
                    grid[neighbor] = tentative_g_score
                    open_set.add(neighbor)
```

Цей фрагмент коду відповідає за перебір сусідніх клітин поточної клітини `current` та оновлення їх вартості шляху, якщо це можливо і вигідно.

- `neighbors` - список координат сусідніх клітин, які розглядаються: клітина праворуч, ліворуч, зверху та знизу від поточної клітини.

- для кожної сусідньої клітини перевіряється, що вона знаходиться в межах сітки та не є перешкодою. Якщо це так, то для сусідньої клітини розраховується передбачувана вартість шляху `tentative_g_score`, яка дорівнює вартості шляху до поточної клітини плюс один крок.

- якщо передбачувана вартість шляху до сусідньої клітини менша, ніж поточна вартість шляху до сусідньої клітини, то вартість шляху до сусідньої клітини оновлюється, і вона додається до множини `open_set` для подальшого розгляду. Також до словника `came_from` заноситься інформація про те, з якої клітини прийшли до цієї.

Цей процес триває доти, доки всі можливі шляхи не будуть досліджені або поки не буде знайдено оптимальний шлях до мети.

```
# Відновлення шляху
    path = []
    current = goal
    while current != start:
        path.append(current)
        if current not in came_from:  # Перевірка наявності ключа у словнику
            break
        current = came_from[current]
    path.append(start)
    path.reverse()
    return path
```

Цей фрагмент коду відповідає за відновлення оптимального шляху від цільової клітини `goal` до початкової клітини `start` з використанням словника `came_from`, який містить інформацію про попередні клітини на шляху.

- Створюється порожній список `path`, в який будуть додаватися клітини шляху.

- Починається з клітини `goal` і додається в `path`.

- Далі, поки поточна клітина не досягне початкової клітини `start`, в циклі відбувається додавання попередньої клітини в `path`, взятої зі словника `came_from`. Якщо для поточної клітини немає інформації про попередню клітину (наприклад, якщо алгоритм не зміг знайти шлях), цикл переривається за допомогою оператора `break`.

- коли всі клітини шляху додані в `path`, починаючи з мети і закінчуючи стартом, список `path` розгортається, щоб шлях був упорядкований від початку до кінця, і повертається як результат функції.

Розмір сітки та кількість перешкод

grid_size = 20

num_obstacles = 100

Цей фрагмент коду задає параметри карти, де відбуватиметься пошук шляху:

grid_size = 20 визначає розмір сітки, тобто кількість клітин по горизонталі та вертикалі.

num_obstacles = 100 визначає кількість перешкод на карті.

Ці значення впливають на складність пошуку шляху, оскільки великий розмір сітки або велика кількість перешкод можуть збільшити обчислювальну складність алгоритму та час виконання.

Генерація карти та відображення її

obstacles = generate_map(grid_size, num_obstacles)

Ви вже визначили функцію generate_map, яка генерує карту із перешкодами. Щоб створити карту та відобразити її, вам потрібно викликати цю функцію та передати їй параметри.

Завдання початкової та кінцевої точок

start = (0, 0)

goal = (grid_size-1, grid_size-1)

Цей фрагмент коду задає початкову точку `start` як координату `(0, 0)` і кінцеву точку `goal` як координату `(grid_size-1, grid_size-1)`. У разі сітки розміром `grid_size=20`, початкова точка буде `(0, 0)`, а кінцева точка `(19, 19)`. Таким чином, початкова точка знаходиться у верхньому лівому куті карти, а кінцева точка – у нижньому правому кутку. Ці точки будуть використовуватися для пошуку оптимального шляху між ними.

```
# Пошук шляху методом D*
path = d_star(start, goal, obstacles)
```

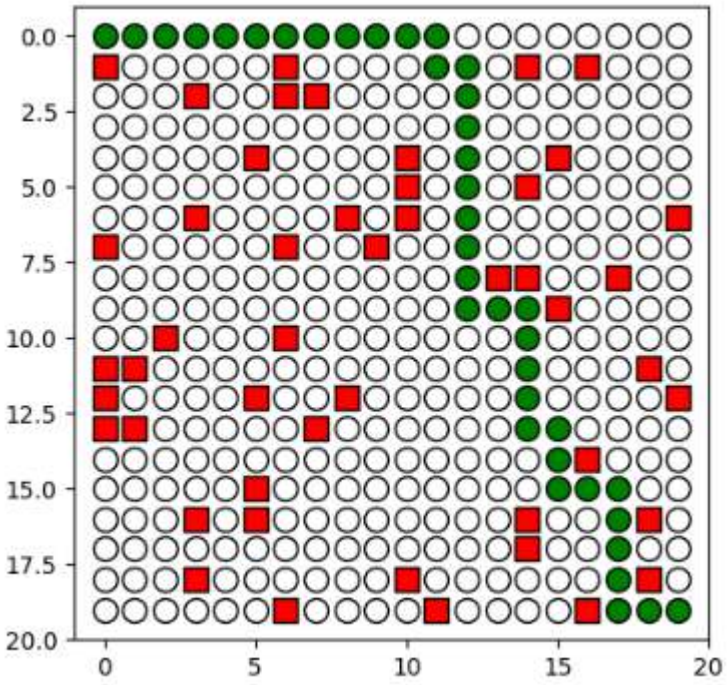
Цей фрагмент коду викликає функцію `d_star` для пошуку оптимального шляху від початкової точки `start` до кінцевої точки `goal` з урахуванням перешкод `obstacles`. Результат пошуку шляху зберігається у змінній `path`.

```
# Відображення карти з перешкодами та шляхом
display_map(grid_size, obstacles, path)
```

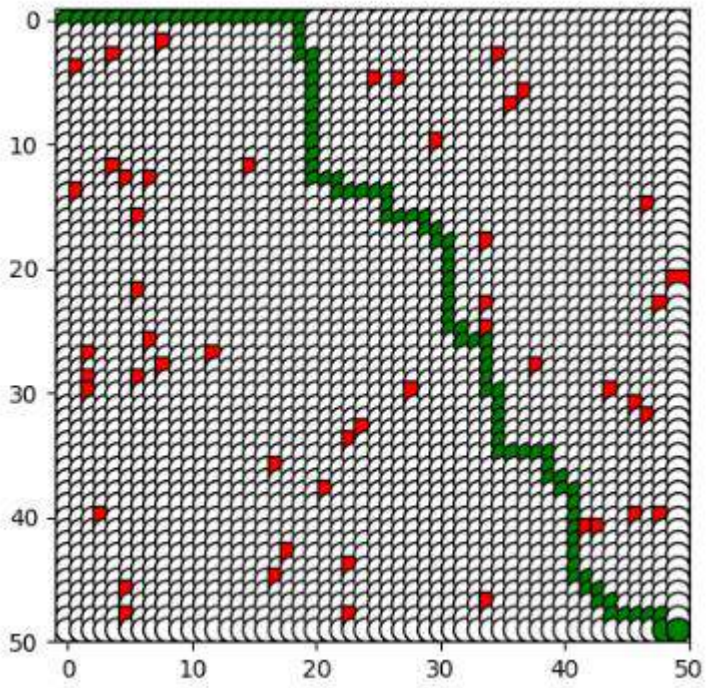
Цей фрагмент коду викликає функцію `display_map` для відображення карти з перешкодами та знайденим шляхом. Функції передаються параметри `grid_size` для розміру сітки, `obstacles` для перешкод та `path` для шляху. Карта буде відображена з перешкодами у вигляді червоних квадратів та шляхом у вигляді зелених крапок.

Повний код програми реалізацій алгоритму D* наведено у додатку Ї.

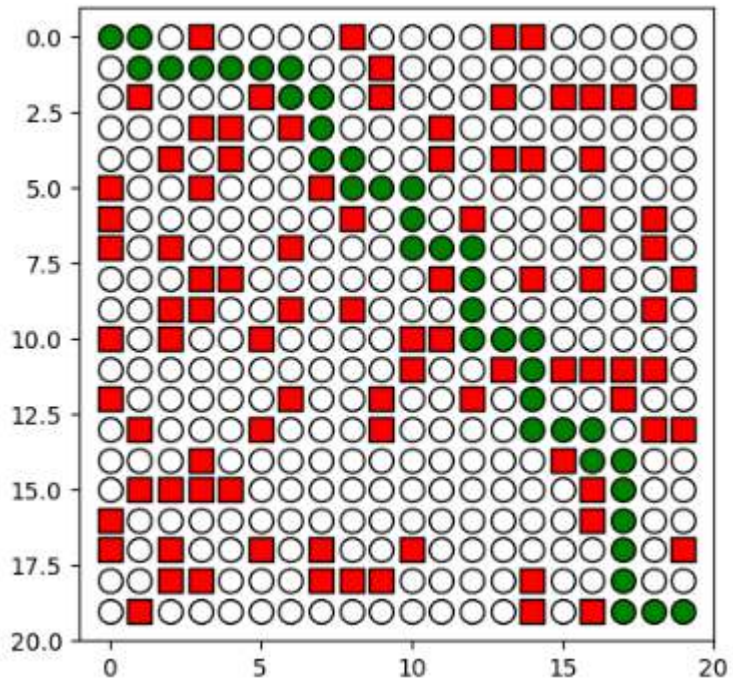
Дослідження будуть проводитися на базі наступної апаратної складової з такими параметрами: CPU Intel (R) Core (TM) i5-9300H CPU @ 2.40GHz, RAM DDR 16,0 GB, GPU NVideo GeForce GTX1660TI та Intel (R) UHD Graphics 630, HDD SSD NVM MQ04ABF100. OS Windows 10 Pro 64-розрядна. Результати роботи програми побудови маршруту на базі алгоритму D*, представлені на малюнку 4.7.



а) (grid_size = 20); (num_obstacles = 50)



в) (grid_size = 50); (num_obstacles = 50)



г) (grid_size = 20); (num_obstacles = 100)

Рисунок 4.7 - Результати роботи програми побудови маршруту на базі алгоритму D*

Алгоритм D* (D-зірка) – це ефективний метод пошуку оптимального шляху для мобільних роботів у середовищі, що змінюється [291-294]. Його застосування знаходить широке застосування у робототехніці та автономних системах для обходу перешкод та досягнення цілей. Головна перевага D* полягає в тому, що він може перераховувати шлях при змінах у середовищі, що робить його ідеальним вибором для динамічних середовищ. Однак, як і у випадку з будь-яким алгоритмом, D* має свої особливості та обмеження. Перш за все, D* має гарну масштабованість, що дозволяє застосовувати його для пошуку шляхів у великих та складних середовищах. Однак, при збільшенні розміру карти або кількості перешкод, алгоритм може споживати більше обчислювальних ресурсів, що може вплинути на продуктивність системи. Також слід враховувати, що D* не завжди гарантує знаходження абсолютно оптимального шляху у всіх випадках через свою природу алгоритму, що базується на евристиках [295-300].

Для успішного застосування алгоритму D* необхідно ретельно налаштовувати параметри, такі як вартість переміщення між вузлами і точність евристичної функції. Це допоможе досягти балансу між точністю знаходження оптимального шляху та витратами обчислювальних ресурсів. Крім того, при роботі з динамічними середовищами важливо регулярно оновлювати інформацію про перешкоди та зміни на карті, щоб алгоритм міг адаптуватися до нових умов та знаходити оптимальні шляхи [301-303].

Важливо розуміти, що алгоритм D* не універсальне рішення для всіх сценаріїв. У деяких випадках інші алгоритми, такі як A* або RRT (Rapidly-exploring Random Tree), можуть бути ефективнішими. Тому перед вибором алгоритму для конкретного завдання необхідно провести аналіз вимог та особливостей середовища, щоб вибрати найбільш підходящий метод.

Таким чином, алгоритм D* є потужним інструментом для побудови оптимальних маршрутів у динамічних середовищах. Його гнучкість та адаптивність роблять його привабливим вибором для мобільних роботів, що працюють у мінливих умовах. Однак для ефективного використання алгоритму необхідно враховувати його особливості та проводити ретельне налаштування параметрів залежно від конкретного завдання.

ВИСНОВКИ

У результаті досліджень, проведених у монографії "Технічне та програмне забезпечення малогабаритного мобільного робота", було розроблено та проаналізовано комплексний підхід до створення ефективного засобу для дослідження зруйнованих будівель. Робота розділена на дві основні частини: розробка малогабаритного мобільного робота та розробка системи керування ним.

Перша частина монографії присвячена проектуванню робота та його апаратному забезпеченню. В результаті досліджень була проведена класифікація мобільних роботів, розроблений концепт малогабаритного робота, вибрані апаратні модулі, розраховані витрати енергії для керування рухом, розроблена схема підключення апаратних модулів, створені 3D моделі конструкцій та складання робота, а також виготовлено деталі за допомогою 3D принтера. В результаті цих досліджень було успішно складено макет малогабаритного мобільного робота, готового до подальших тестувань та використання.

Друга частина монографії присвячена розробці системи керування малогабаритним мобільним роботом. Обране середовище розробки, розроблений алгоритм керування, реалізована програмна частина системи керування, налаштована та прошита плата керування, а також розроблений інтерфейс користувача HMI для зручного керування роботом оператором.

Крім того, у монографії розглянуті методи та алгоритми обробки інформації, отриманої від датчиків робота, такі як аналіз зображень, виявлення контурів, оптичний потік, визначення об'єктів та їхнє відслідковування. Також описано алгоритми побудови маршрутів для руху робота, такі як A*, BRRT, хвильовий алгоритм, PRM, D*, які дозволяють роботу автономно прокладати оптимальний шлях у зруйнованому середовищі.

Додатковою важливою частиною монографії є програмна реалізація розроблених алгоритмів. Усі алгоритми, описані в роботі, були реалізовані у вигляді програмного забезпечення і протестовані на макеті малогабаритного мобільного робота. Приклади програмного коду для кожного алгоритму подані у додатках до монографії, що дозволяє читачам детально ознайомитися з реалізацією та використанням цих алгоритмів.

Отже, монографія висвітлює важливі аспекти розробки малогабаритного мобільного робота для дослідження зруйнованих будівель, що може бути використаною у практичних роботах з рятування та обстеження пошкоджених об'єктів.

СПИСОК ЛІТЕРАТУРИ

1. Zappa, I., Tomasoni, A., Zanchettin, A. M., & Rocco, P. (2025). Robust Segmentation and Classification of Robot Skills from Demonstrations with Detection of Unknown Skills. *Procedia Computer Science*, *253*, 1535-1544.

2. Diego, P., Herrero, S., Macho, E., Corral, J., Diez, M., Campa, F. J., & Pinto, C. (2024). Devices for Gait and Balance Rehabilitation: General Classification and a Narrative Review of End Effector-Based Manipulators. *Applied Sciences*, *14*(10), 4147.

3. Yevsieiev, V., & Uskov, S. (2024). *Development of the Layout Concept of a Small-Dimensioned Mobile Robot With Increased Accessibility* (Doctoral dissertation, International Scientific Unity).

4. NEVLIUDOV, I., YEVSIEIEV, V., MAKSYMOVA, S., & CHALA, O. (2025). A Small-Sized Robot Prototype Development Using 3D Printing. *acta mechanica et automatica*, *19*(1).

5. Samsuria, E., Mahmud, M. S. A., Wahab, N. A., Romdlony, M. Z., Abidin, M. S. Z., & Buyamin, S. (2024). Adaptive fuzzy-genetic algorithm operators for solving mobile robot scheduling problem in job-shop FMS environment. *Robotics and Autonomous Systems*, *176*, 104683.

6. Liu, Y., Wang, S., Xie, Y., Xiong, T., & Wu, M. (2024). A review of sensing technologies for indoor autonomous mobile robots. *Sensors*, *24*(4), 1222.

7. Katsampiris-Salgado, K., Haninger, K., Gkrizis, C., Dimitropoulos, N., Krüger, J., Michalos, G., & Makris, S. (2024). Collision detection for collaborative assembly operations on high-payload robots. *Robotics and Computer-Integrated Manufacturing*, *87*, 102708.

8. Gillet, S., Vázquez, M., Andrist, S., Leite, I., & Sebo, S. (2024). Interaction-shaping robotics: Robots that influence interactions between other agents. *ACM Transactions on Human-Robot Interaction*, *13*(1), 1-23.

9. Wang, X., Wu, S., Zhao, Z., Guo, H., & Chen, W. (2025). Optimization of emergency rescue routes after a violent earthquake. *Natural Hazards*, *121*(4), 4585-4613.

10. Dar, R. U. N., & Alagappan, P. (2024). Ballistic impact response of reinforced concrete panels subjected to diverse multiple projectile impact scenarios: A numerical study. *Engineering Failure Analysis*, *164*, 108697.

11. Taheri, H., Hosseini, S. R., & Nekoui, M. A. (2024). Deep reinforcement learning with enhanced ppo for safe mobile robot navigation. *arXiv preprint arXiv:2405.16266*.

12. NIFTi. URL: https://www.roboticstoday.com/projects/nifti (Доступ 22.04.2025)

13. Rassypnov, G. A. (2024, January). Design of Software and Hardware Complex of a Prototype of a Mine Rescure Robot. In *2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon)* (pp. 481-484). IEEE.

14. Li, L., & Zhao, Z. (2024). Performance Test, Index System Establishment, and Comprehensive Evaluation of Earthquake Rescue Robots. *Electronics*, *13*(7), 1401.

15. Dychko, A., Remez, N., Yeremeyev, I., & Borovyk, M. (2024, May). Forecasting the risks of man-made and ecological disasters. In *IOP Conference Series: Earth and Environmental Science* (Vol. 1348, No. 1, p. 012011). IOP Publishing.

16. Jovanović, A., Mitić, V., Sibinović, V., & Veselić, B. (2024, March). Design and Implementation of a Mobile Platform for Education and Research. In *2024 23rd International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1-5). IEEE.

17. Maksymova, S., Yevsieiev, V., Nevliudov, I., & Bahlai, O. (2024, May). Balancing System For A Zoomorphic Spot Type Mobile Robot Development Using An Accelerometer MPU 6050 (GY-521). In *2024 IEEE 19th International*

Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 39-42). IEEE.

18. Khaleel, R. Z., Khaleel, H. Z., Abdullah Al-Hareeri, A. A., Mahdi Al-Obaidi, A. S., & Humaidi, A. J. (2024). Improved Trajectory Planning of Mobile Robot Based on Pelican Optimization Algorithm. *Journal Européen des Systèmes Automatisés*, *57*(4).

19. Abu-Jassar, A. T., Attar, H., Amer, A., Lyashenko, V., Yevsieiev, V., & Solyman, A. (2025). Development and Investigation of Vision System for a Small-Sized Mobile Humanoid Robot in a Smart Environment. *International Journal of Crowd Science*, *9*(1), 29-43.

20. Kuzmenko, O., Yevsieiev, V., Maksymova, S., & Abu-Jassar, A. (2024). Robot Model for Mines Searching Development. *Multidisciplinary Journal of Science and Technology*, *4*(6), 347-355.

21. Das, B., & Halder, K. K. (2024, March). Face Recognition Using ESP32-Cam for Real-Time Tracking and Monitoring. In *2024 International Conference on Advances in Computing, Communication, Electrical, and Smart Systems (iCACCESS)* (pp. 01-06). IEEE.

22. Chen, Q., Xia, C., Shi, Y., Wang, X., Zhang, X., & He, Y. (2024). A Novel Approach for Asparagus Comprehensive Classification Based on TOPSIS Evaluation and SVM Prediction. *Agronomy*, *14*(6), 1175.

23. Tran, Q. L., Nguyen, B., Jones, G. J., & Gurrin, C. (2024, May). MemoriLens: a Low-cost Lifelog Camera Using Raspberry Pi Zero. In *Proceedings of the 2024 International Conference on Multimedia Retrieval* (pp. 1255-1259).

24. PBV, R. R., Mandapati, V. S., Pilli, S. L., Manojna, P. L., Chandana, T. H., & Hemalatha, V. (2024, April). Home Security with IOT and ESP32 Cam-AI Thinker Module. In *2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC-ROBINS)* (pp. 710-714). IEEE.

25. Okokpujie, I. P., & Akingunsoye, A. V. (2024). an ESP32-Cam, IoT, and Twilio Application Programming Interface. In *Proceedings of 3rd International*

Conference on Smart Computing and Cyber Security: Strategic Foresight, Security Challenges and Innovation (SMARTCYBER 2023) (Vol. 914, p. 109). Springer Nature.

26. Sumari, A. D. W., Annurroni, I., & Ayuningtyas, A. (2025). The Internet-of-Things-based Fishpond Security System Using NodeMCU ESP32-CAM Microcontroller. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, *9*(1), 51-61.

27. Модуль камери OV2640 2Mp/FOV120-NV для ESP-CAM. URL: https://arduino.ua/prod5194-modyl-kameri-ov2640-2mpfov160-nv-dlya-esp-cam (Доступ 22.04.2025)

28. Модуль камери 2Мп OV2640 для ESP32-CAM. URL: https://arduino.ua/prod4514-modyl-kameri-2mp-ov2640-dlya-esp32-cam-fov70 (Доступ 22.04.2025)

29. OV5640 AF Camera Module 5 Million Pixel Autofocus. URL: https://www.amazon.co.jp/-/en/OV5640-Camera-Module-Million-Autofocus/dp/B0C5N9D3TK?th=1&language=en_US (Доступ 22.04.2025)

30. Herman, Y., Hasibuan, A. Z., & Sembirimg, A. (2024). Prototype Robot Pengantar Barang Pengikut Marka Hitam Berbasis Mikrokontroller. *Explorer*, *4*(2), 87-96.

31. Чебанчик Д. О. Розробка макета мобільного робота на базі ESP32-CAM : пояснювальна записка до атестаційної роботи здобувача вищої освіти на першому (бакалаврському) рівні, спеціальність 151 – Автоматизація та комп`ютерно-інтегровані технології / Д. О. Чебанчик ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2024. – 73 с.

32. Невлюдов І. Ш. BEAM робототехніка : навч. посіб. / І. Ш. Невлюдов, В. В. Євсєєв, С. С. Максимова ; Харків. нац. ун-т радіоелектроніки, кафедра комп’ютерно-інтегрованих технологій, автоматизації та робототехніки (КІТАР). – Кривий Ріг : Видавець Чернявський Д. О., 2024. – 276 с. – ISBN 978-617-8045-79-1

33. Підвищуючий модуль заряду Li-on 18650 (2S 1A-2A-4A 8.4B) Type-C. URL: https://myproject.com.ua/pidvischujuchij-modul-zarjadu-li-on-18650-2s-1a-84v-type-c-ua.html (Доступ 22.04.2025)

34. Зарядний модуль TP4056 Type-C з функцією захисту акумулятора. URL: https://arduino.ua/prod4517-zaryadnii-modyl-tp4056-type-c-s-fynkciei-zashhiti-akkymylyatora (Доступ 22.04.2025)

35. IP 2312 Модуль для зарядки Li-ion 5V 3A Type-C 4.2В/4.35В. URL: https://myproject.com.ua/ru/ip-2312-modul-dlja-zarjadki-li-ion-5v-3a-type-c-ru.html (Доступ 22.04.2025)

36. Lin, H. I., Mandal, R., & Wibowo, F. S. (2024). BN-LSTM-based energy consumption modeling approach for an industrial robot manipulator. *Robotics and Computer-Integrated Manufacturing*, *85*, 102629.

37. Huo, F., Zhu, S., Dong, H., & Ren, W. (2024). A new approach to smooth path planning of Ackerman mobile robot based on improved ACO algorithm and B-spline curve. *Robotics and Autonomous Systems*, *175*, 104655.

38. Rybczak, M., Popowniak, N., & Lazarowska, A. (2024). A survey of machine learning approaches for mobile robot control. *Robotics*, *13*(1), 12.

39. Abu-Jassar, A. T., Attar, H., Amer, A., Lyashenko, V., Yevsieiev, V., & Solyman, A. (2025). Development and Investigation of Vision System for a Small-Sized Mobile Humanoid Robot in a Smart Environment. *International Journal of Crowd Science*, *9*(1), 29-43.

40. Starykova S. Using Free Web Applications for Designing Mobile Robots in General Secondary Education Institutions (GSEI) / S. Starykova // Комп’ютерно-інтегрованих технологій, автоматизації та робототехніки 2025 : тези доповідей II-ої Всеукраїнськой конференції, 16-17 травня 2025. - Харків, 2025. – С. 18-22.

41. Tüysüz, C., Bodur, N. C., & Ugulu, I. (2024). Tinkercad circuits platform-based learning experiences of gifted students in the emergency distance education process. *Journal of Advanced Academics*, *35*(2), 329-356.

42. Handayani, A. N., Fitrianti, N. N., Kusumawardana, A., Elmunsyah, H., Arai, K., & Asmara, R. A. (2024). Development of Autodesk-based Digital Teaching Materials as a Support for Basic Digital Engineering Learning in Vocational Schools. In *E3S Web of Conferences* (Vol. 473, p. 04007). EDP Sciences.

43. Мачуга, О. С., & Олянишен, Т. В. (2024). Using Solidworks Simulation tool for automated design of drying chambers and study of their operation parameters. *Scientific Bulletin of UNFU*, *34*(2), 109-115.

44. Kyratsis, P., Tzotzis, A., & Davim, J. P. (2025). Solid Model Geometric Objects Detection Based on CAD Software Programming. In *CAD-based Programming for Design and Manufacturing* (pp. 39-52). Cham: Springer Nature Switzerland.

45. Khairulmaini, M., Michael, Z., Hamid, M. F. A., Abidin, N. A. Z., & Roslan, A. (2024). Analyzing The Influence of Diameter and Winding on Heat Transfer Efficiency in Spiral Tube Heat Exchangers: A CAD-Integrated CFD Study Using Solidworks Flow Simulation. In *Journal of Physics: Conference Series* (Vol. 2688, No. 1, p. 012002). IOP Publishing.

46. Alqutaibi, A. Y., Alghauli, M. A., Aljohani, M. H. A., & Zafar, M. S. (2024). Advanced additive manufacturing in implant dentistry: 3D printing technologies, printable materials, current applications and future requirements. *Bioprinting*, e00356.

47. Pérez-Sevilla, M., Rivas-Navazo, F., Latorre-Carmona, P., & Fernández-Zoppino, D. (2025). Protocol for Converting DICOM Files to STL Models Using 3D Slicer and Ultimaker Cura. *Journal of Personalized Medicine*, *15*(3), 118.

48. Hasan, M. R., Davies, I. J., Pramanik, A., John, M., & Biswas, W. K. (2024). Potential of recycled PLA in 3D printing: A review. *Sustainable manufacturing and service economics*, 100020.

49. Zulfiqar, A. (2024). *Hands-on ESP32 with Arduino IDE: Unleash the power of IoT with ESP32 and build exciting projects with this practical guide*. Packt Publishing.

50. Scaffi, E., Bonneau, A., Mouël, F. L., & Mieyeville, F. (2024). Usability and Performance Analysis of Embedded Development Environment for On-device Learning. *arXiv preprint arXiv:2404.07948*.

51. Mota, A., Serôdio, C., & Valente, A. (2024). Matter Protocol Integration Using Espressif's Solutions to Achieve Smart Home Interoperability. *Electronics*, *13*(11), 2217.

52. Mishra, M., Stallone, M., Zhang, G., Shen, Y., Prasad, A., Soria, A. M., ... & Panda, R. (2024). Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

53. Jelović, M., Terzić, S. A., Kovačević, T., Zorica, S., & Klarin, V. (2024). WEB-BASED MOTOR CONTROL AND ENVIRONMENTAL MONITORING SYSTEM. *ORGANIZER*, 824.

54. Mota, A., Serôdio, C., & Valente, A. (2024). Matter Protocol Integration Using Espressif's Solutions to Achieve Smart Home Interoperability. *Electronics*, *13*(11), 2217.

55. Chen, D. (2024, October). Research and Design of a Robot Controller Based on MicroPython. In *2024 17th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)* (pp. 1-6). IEEE.

56. Mnyandu, W. T., Terzoli, A., & Kobo, H. (2024, November). Emulating LTSP Clients with Mininet and QEMU: Enabling Scalable Testing for Educational Deployments. In *2024 4th International Multidisciplinary Information Technology and Engineering Conference (IMITEC)* (pp. 30-37). IEEE.

57. Kristiyanto, Y., Sugiyono, S., Perdana, C. A., & Sarah, S. (2024). PENGEMBANGAN KEAMANAN KOMPUTER DENGAN TEKNOLOGI THIN CLIENT DAN DOMAIN CONTROLLER DENGAN METODE NETWORK DEVELOPMENT LIFE CYCLE (NDLC). *Jurnal Manajamen Informatika Jayakarta*, *4*(3), 258-265.

58. Ho-Dac, H., Nguyen, P. T., & Tran, H. D. (2024). A Practical Approach for Virtual Desktop Infrastructure Implementation in Educational. *From Smart*

City to Smart Factory for Sustainable Future: Conceptual Framework, Scenarios, and Multidiscipline Perspectives, *1062*, 237.

59. Awodeyi, A. (2024). The Development of a Self-Organizing Multipurpose Mobile Robot. *Nigerian Journal of Science and Engineering Infrastructure*, *2*(1).

60. Gagan, M., Kaushik, K. R., Fernandes, K. F., & Madhu, S. (2024). Intruder detection using surveillance CCTV camera interfaced ESP32 Cam module. In *Computer Science Engineering* (pp. 402-406). CRC Press.

61. Abu-Jassar, A. T., Attar, H., Amer, A., Lyashenko, V., Yevsieiev, V., & Solyman, A. (2024). Remote Monitoring System of Patient Status in Social IoT Environments Using Amazon Web Services (AWS) Technologies and Smart Health Care. *International Journal of Crowd Science*, *8*.

62. Yevsieiev V. Using Contouring Algorithms to Select Objects in the Robots' Workspace / V. Yevsieiev, S. Maksymova, N. Demska // Technical science research in Uzbekistan. – 2024. – Vol. 2(2). – P. 32–42.

63. Yevsieiev V. The Canny Algorithm Implementation for Obtaining the Object Contour in a Mobile Robot's Workspace in Real Time / V. Yevsieiev, S. Maksymova, Amer Abu-Jassar // Journal of Universal Science Research . – 2024. – Vol. 2(3). – P. 7–19.

64. Xiang, A., Zhang, J., Yang, Q., Wang, L., & Cheng, Y. (2024). Research on splicing image detection algorithms based on natural image statistical characteristics. *arXiv preprint arXiv:2404.16296*.

65. Lasagni, L. (2025). Low contrast detection and super-resolution in CT images: evaluation of a novel approach based on Centroidal Voronoi Tessellation. *arXiv preprint arXiv:2503.06666*.

66. Song, Y., Li, C., Xiao, S., Zhou, Q., & Xiao, H. (2024). A parallel Canny edge detection algorithm based on OpenCL acceleration. *Plos one*, *19*(1), e0292345.

67. Luo, Y., & Luo, Z. (2024). Infrared and visible image fusion algorithm based on improved residual Swin Transformer and Sobel operators. *IEEE Access*.

68. Agrawal, H., & Desai, K. (2024). CANNY EDGE DETECTION: A COMPREHENSIVE REVIEW. *International Journal of Technical Research & Science*, *9*, 27-35.

69. Shaw, R., & Jenilarani, D. (2025, March). Comparative image evaluation of lung tuberculosis for improved accuracy by executing Sobel and Robert operators. In *AIP Conference Proceedings* (Vol. 3252, No. 1). AIP Publishing.

70. Yang, Z., Zuo, S., Zhou, Y., He, J., & Shi, J. (2024). A Review of Document Binarization: Main Techniques, New Challenges, and Trends. *Electronics*, *13*(7), 1394.

71. Active Contours Method Implementation for Objects Selection in the Mobile Robot's Workspace / V. Yevsieiev, S. Maksymova, N. Starodubtsev, Amer Abu-Jassar // Journal of Universal Science Research. – 2024. –Vol. 2(2). – P. 135–145.

72. Xue, P., & Niu, S. (2024). A novel active contour model based on features for image segmentation. *Pattern Recognition*, *155*, 110673.

73. Dong, B., Weng, G., Bu, Q., Zhu, Z., & Ni, J. (2024). An active contour model based on shadow image and reflection edge for image segmentation. *Expert Systems with Applications*, *238*, 122330.

74. George, N., Ramachandran, S., & Jiji, C. V. (2024). A hybrid approach using CNN and active contour model for automated segmentation of macular edema. *Journal of Intelligent & Fuzzy Systems*, JIFS-219401.

75. Ma, P., Yuan, H., Chen, Y., Chen, H., Weng, G., & Liu, Y. (2024). A Laplace operator-based active contour model with improved image edge detection performance. *Digital Signal Processing*, *151*, 104550.

76. Ametefe, D. S., Sarnin, S. S., Ali, D. M., Ametefe, G. D., John, D., Aliu, A. A., & Zoreno, Z. (2024). Automatic classification and segmentation of blast cells using deep transfer learning and active contours. *International Journal of Laboratory Hematology*, *46*(5), 837-849.

77. Behara, K., Bhero, E., & Agee, J. T. (2024). An improved skin lesion classification using a hybrid approach with active contour snake model and lightweight attention-guided capsule networks. *Diagnostics*, *14*(6), 636.

78. Behara, K., Bhero, E., & Agee, J. T. (2024). An improved skin lesion classification using a hybrid approach with active contour snake model and lightweight attention-guided capsule networks. *Diagnostics*, *14*(6), 636.

79. Zia, H., Soomro, S., & Choi, K. N. (2024). Image segmentation using bias correction active contours. *IEEE Access*.

80. Saini, T., & Shiakolas, P. S. (2025). In Situ Active Contour-Based Segmentation and Dimensional Analysis of Part Features in Additive Manufacturing. *Journal of Manufacturing and Materials Processing*, *9*(3), 102.

81. Mei, T., Zi, Y., Cheng, X., Gao, Z., Wang, Q., & Yang, H. (2024, August). Efficiency optimization of large-scale language models based on deep learning in natural language processing tasks. In *2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE)* (pp. 1231-1237). IEEE.

82. Amer Abu-Jassar The Optical Flow Method and Graham's Algorithm Implementation Features for Searching for the Object Contour in the Mobile Robot's Workspace / Amer Abu-Jassar, V. Yevsieiev, S. Maksymova // Journal of Universal Science Research. – 2024, – Vol. 2(3). – P. 64-75.

83. Cho, S., Huang, J., Kim, S., & Lee, J. Y. (2024). Flowtrack: Revisiting optical flow for long-range dense tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 19268-19277).

84. Le Moing, G., Ponce, J., & Schmid, C. (2024). Dense optical tracking: Connecting the dots. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 19187-19197).

85. Alfarano, A., Maiano, L., Papa, L., & Amerini, I. (2024). Estimating optical flow: A comprehensive review of the state of the art. *Computer Vision and Image Understanding*, 104160.

86. Shiba, S., Klose, Y., Aoki, Y., & Gallego, G. (2024). Secrets of event-based optical flow, depth and ego-motion estimation by contrast maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

87. Sun, S., Liu, J., Li, H., Liu, G., Li, T., & Gao, W. (2024). Streamflow: streamlined multi-frame optical flow estimation for video sequences. *Advances in neural information processing systems*, *37*, 9205-9228.

88. Dong, Q., & Fu, Y. (2024). Memflow: Optical flow estimation and prediction with memory. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 19068-19078).

89. Luo, A., Li, X., Yang, F., Liu, J., Fan, H., & Liu, S. (2024). Flowdiffuser: Advancing optical flow estimation with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 19167-19176).

90. Wang, J., Huang, Z., Xu, Y., & Xie, D. (2024). Gas–Liquid Two-Phase Flow Measurement Based on Optical Flow Method with Machine Learning Optimization Model. *Applied Sciences*, *14*(9), 3717.

91. Magana, M. J. U., Villespin, J. A. A. V., & Manlises, C. O. (2024, December). Baybayin Translation Using Lucas-Kanade Optical Flow and RNN. In *TENCON 2024-2024 IEEE Region 10 Conference (TENCON)* (pp. 1716-1719). IEEE.

92. Wen, B. (2024, November). Language based object detection with human body using CLIPseg: Integrating CLIP and Optical Flow for Unsupervised Object Tracking and Keypoint Detection. In *2024 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML)* (pp. 1354-1360). IEEE.

93. Liang, F., Wu, B., Wang, J., Yu, L., Li, K., Zhao, Y., ... & Marculescu, D. (2024). Flowvid: Taming imperfect optical flows for consistent video-to-video synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8207-8216).

94. Guo, H., Wang, Y., & Guo, X. (2025). Optical Flow Estimation Method Based on Bidirectional Consistency Combined Occlusion. *Information Technology and Control*, *54*(1), 219-233.

95. Xu, Y., Tang, G., Yousefzadeh, A., de Croon, G. C., & Sifalakis, M. (2025). Event-based optical flow on neuromorphic processor: ANN vs. SNN comparison based on activation sparsification. *Neural Networks*, 107447.

96. Zhang, Z., Zhao, S., Mao, X., Liu, S., Wang, H., Xu, T., & Chen, E. (2024, October). A Multi-scale Feature Learning Network with Optical Flow Correction for Micro-and Macro-expression Spotting. In *Proceedings of the 32nd ACM International Conference on Multimedia* (pp. 11497-11502).

97. Yevsieiev V. Object Recognition and Tracking Method in the Mobile Robot's Workspace in Real Time / V. Yevsieiev, Amer Abu-Jassar, S. Maksymova // Technical science research in Uzbekistan. – 2024. – Vol. 2(2). – P. 115-124.

98. Hassan, S., Mujtaba, G., Rajput, A., & Fatima, N. (2024). Multi-object tracking: a systematic literature review. *Multimedia Tools and Applications*, *83*(14), 43439-43492.

99. Abba, S., Bizi, A. M., Lee, J. A., Bakouri, S., & Crespo, M. L. (2024). Real-time object detection, tracking, and monitoring framework for security surveillance systems. *Heliyon*, *10*(15).

100. Abba, S., Bizi, A. M., Lee, J. A., Bakouri, S., & Crespo, M. L. (2024). Real-time object detection, tracking, and monitoring framework for security surveillance systems. *Heliyon*, *10*(15).

101. Kusnandar, T., Santoso, J., & Surendro, K. (2024). Enhancing Color Selection in HSV Color Space. *Ingenierie des Systemes d'Information*, *29*(4), 1483.

102. Burambekova, A., & Shamoi, P. (2024). Comparative analysis of color models for human perception and visual color difference. *arXiv preprint arXiv:2406.19520*.

103. Riana, S. J., Joy, M. H. C., & Rasedujjaman, M. (2024, May). Comparative Analysis of White Blood Cell Segmentation: HSV Color Space Model vs Traditional Machine Learning Techniques. In *2024 3rd International*

Conference on Artificial Intelligence For Internet of Things (AIIoT) (pp. 1-6). IEEE.

104. Bingjie, Z., Qing, S., Lujian, X., Meng, L., & Juan, Z. (2024, August). HSV Color Space Based Automated Chemical Titrator. In *2024 6th International Conference on Industrial Artificial Intelligence (IAI)* (pp. 1-6). IEEE.

105. Santoso, K. A., Kamsyakawuni, A., & Izza, S. V. R. (2024). Comparison Classification Of Tomatoes Ripeness Based On RGB, HSV And CMYK Colors Based On Correlation Coefficient. *Journal of Computers and Digital Business*, *3*(3), 112-120.

106. Zhai, H., Chen, M., Yang, X., & Kang, G. (2024, October). Multi-scale hsv color feature embedding for high-fidelity nir-to-rgb spectrum translation. In *2024 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 3287-3292). IEEE.

107. Zahed, N. A. S., Sabapathy, T., Rihan, S. D. A., Tayfour, O. E., Ahmed, A. E. T., & Elobaid, M. E. (2024, September). Classification of Mango Ripeness Grades Using Machine Learning. In *International Conference on Electronic Design* (pp. 361-371). Cham: Springer Nature Switzerland.

108. Abba, S., Bizi, A. M., Lee, J. A., Bakouri, S., & Crespo, M. L. (2024). Real-time object detection, tracking, and monitoring framework for security surveillance systems. *Heliyon*, *10*(15).

109. Yadav, A., & Kumar, E. (2024). Object Detection on Real-Time Video with FPN and Modified Mask RCNN Based on Inception-ResNetV2. *Wireless Personal Communications*, *138*(4), 2065-2090.

110. Mokeddem, M. L., Belahcene, M., & Bourennane, S. (2024). Real-time social distance monitoring and face mask detection based Social-Scaled-YOLOv4, DeepSORT and DSFD&MobileNetv2 for COVID-19. *Multimedia Tools and Applications*, *83*(10), 30613-30639.

111. Kumar, K. S., & Safwan, K. M. B. (2024). Accelerating object detection with yolov4 for real-time applications. *arXiv preprint arXiv:2410.16320*.

112. Reichel, S., Frank, P., & Blankenbach, K. (2024, March). Reliable and "good" camera calibration based on machine learning inspired workflow: parameter study and experimental results. In *AI and Optical Data Sciences V* (Vol. 12903, pp. 84-95). SPIE.

113. Hu, M., Yin, W., Zhang, C., Cai, Z., Long, X., Chen, H., ... & Shen, S. (2024). Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

114. Kwiek, P., & Jakubowska, M. (2024). Color Standardization of Chemical Solution Images Using Template-Based Histogram Matching in Deep Learning Regression. *Algorithms*, *17*(8), 335.

115. Yevsieiev V. The Sobel algorithm implementation for detection an object contour in the mobile robot's workspace in real time / V. Yevsieiev, S. Maksymova, Ahmad Alkhalaileh // Technical Science Research in Uzbekistan. – 2024. – Vol. 2(3). – P. 23-33.

116. Chen, G., Wang, Y., Song, S., & Yang, W. (2024). Research on coal-rock boundary identification based on the morphological sobel algorithm. *Scientific Reports*, *14*(1), 24095.

117. An, Y., Yuan, Q., & Zhang, H. (2025). Edge Detection Using Sobel Algorithm and YCbCr Colour Space Optimized on FPGA. *Informatica*, *49*(7).

118. An, Y., Yuan, Q., & Zhang, H. (2025). Edge Detection Using Sobel Algorithm and YCbCr Colour Space Optimized on FPGA. *Informatica*, *49*(7).

119. Wang, S., Yu, L., & Li, J. (2024). Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, *37*, 54905-54931.

120. Wu, X., Yu, W., Zhang, C., & Woodland, P. (2024). An improved empirical fisher approximation for natural gradient descent. *Advances in Neural Information Processing Systems*, *37*, 134151-134194.

121. Hu, J., & Fu, M. C. (2025). On the convergence rate of stochastic approximation for gradient-based stochastic optimization. *Operations Research*, *73*(2), 1143-1150.

122. Liang, Y., Sha, Z., Shi, Z., Song, Z., & Zhou, Y. (2024). Multi-layer transformers gradient can be approximated in almost linear time. *arXiv preprint arXiv:2408.13233*.

123. Liang, Y., Liu, H., Shi, Z., Song, Z., Xu, Z., & Yin, J. (2024). Conv-basis: A new paradigm for efficient attention inference and gradient computation in transformers. *arXiv preprint arXiv:2405.05219*.

124. Khanh, P. D., Mordukhovich, B. S., & Tran, D. B. (2024). A new inexact gradient descent method with applications to nonsmooth convex optimization. *Optimization Methods and Software*, 1-29.

125. Khanh, P. D., Mordukhovich, B. S., & Tran, D. B. (2024). Inexact reduced gradient methods in nonconvex optimization. *Journal of Optimization Theory and Applications*, *203*(3), 2138-2178.

126. Luo, C., Wang, L., Xie, Y., & Chen, B. (2024). A new conjugate gradient method for moving force identification of vehicle–bridge system. *Journal of Vibration Engineering & Technologies*, *12*(1), 19-36.

127. Yadav, G. S., Sathish, T., & Suresh, G. R. (2024, May). Detection of fire in forest area using chromatic measurements by Sobel edge detection algorithm compared with Prewitt gradient edge detector. In *AIP Conference Proceedings* (Vol. 2853, No. 1). AIP publishing.

128. Μαστορέκα, M. M. (2024). *Sobel Algorithm implementation with High-Level-Synthesis (HLS) on Field Programmable Gate Arrays (FPGA)* (Doctoral dissertation, Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης).

129. Priyanka, V., Rama, Y. S., Sravani, K., & Kavya, B. (2024, July). Implementation of Sobel Edge Detection with Image Processing on FPGA. In *2024 2nd World Conference on Communication & Computing (WCONF)* (pp. 1-5). IEEE.

130. Shaw, R., & Rani, D. J. (2024, November). Comparative image assessment and enhancement of lung cancer by implementing Sobel and Robert noise reduction operators. In *AIP Conference Proceedings* (Vol. 3193, No. 1). AIP Publishing.

131. Vaishnavi, L. L., Bhavana, M., Manideep, D. C. M., Neelima, N., & Venugopal, V. (2024, July). A Fast and Accurate Object Counting using Sobel Edge Detection System for Real–Time Star Gazing. In *2024 Asia Pacific Conference on Innovation in Technology (APCIT)* (pp. 1-5). IEEE.

132. Priyanka, V., Rama, Y. S., Sravani, K., & Kavya, B. (2024, July). Implementation of Sobel Edge Detection with Image Processing on FPGA. In *2024 2nd World Conference on Communication & Computing (WCONF)* (pp. 1-5). IEEE.

133. Al Rawahi, S. (2025). A Comparison of Sobel and Prewitt Edge Detection Operators. *East Journal of Computer Science*, *1*(1), 49-58.

134. Dong, K., Pengcheng, Q., & Xiaohan, K. (2024, August). A Study of a Sobel Edge Detection and SVM Based ALPR System on an ARM Single-Board Microcomputer. In *2024 7th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)* (pp. 485-490). IEEE.

135. Xiong, Z., Yu, L., An, S., Zheng, J., Ma, Y., Micó, V., & Gao, P. (2024). Automatic identification and analysis of cells using digital holographic microscopy and Sobel segmentation. *Frontiers in Photonics*, *5*, 1359595.

136. Maksymova S. The Lucas-Kanade method implementation for estimating the objects movement in the mobile robot’s workspace / S. Maksymova, V. Yevsieiev, Ahmad Alkhalaileh // Journal of Universal Science Research. – 2024. – Vol. 2(3). – P. 187-197.

137. Winkler, J. R. (2024). Error Analysis and Condition Estimation of the Pyramidal Form of the Lucas-Kanade Method in Optical Flow. *Electronics*, *13*(5), 812.

138. Ghoul, K., Zaidi, S., & Laboudi, Z. (2024). A New Motion Estimation Method using Modified Hexagonal Search Algorithm and Lucas-Kanade Optical Flow Technique. *Advances in Electrical & Computer Engineering*, *24*(1).

139. Kong, C., Zhou, X., Cheng, W., & Sun, Z. (2024). Face Expression Recognition based on Lucas-Kanade. *International Core Journal of Engineering*, *10*(2), 212-218.

140. Zhou, Z., Luo, J., Zhu, Q., Wang, Y., Zhong, H., Feng, M., & Chen, L. (2024). Uncertainty Guided Deep Lucas-Kanade Homography for Multimodal Image Alignment. *IEEE Transactions on Geoscience and Remote Sensing*.

141. Magana, M. J. U., Villespin, J. A. A. V., & Manlises, C. O. (2024, December). Baybayin Translation Using Lucas-Kanade Optical Flow and RNN. In *TENCON 2024-2024 IEEE Region 10 Conference (TENCON)* (pp. 1716-1719). IEEE.

142. Raymundo, A. A., Penuliar, J. C. D., & Manlises, C. O. (2024, December). Pedestrian Tracking Using YOLOv8 with Lucas-Kanade Optical Flow for Traffic Light Control Application. In *TENCON 2024-2024 IEEE Region 10 Conference (TENCON)* (pp. 1650-1653). IEEE.

143. Farkhadov, M., Teplukhin, R., Abramenkov, A., Abdulov, A., & Lychkov, I. (2025). Lucas Kanade optical flow computation based on the finite dimensional sampling theories. *Upravlenie Bol'shimi Sistemami*, *113*, 129-180.

144. Rahman, M., & Taebi, A. (2024, December). Extracting Cardiovascular-Induced Chest Vibrations from Ordinary Chest Videos: A Comparative Study. In *2024 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)* (pp. 1-5). IEEE.

145. Winkler, J. R. (2024). Error Analysis and Condition Estimation of the Pyramidal Form of the Lucas-Kanade Method in Optical Flow. *Electronics*, *13*(5), 812.

146. Chen, Z., Liu, J., Lu, Z. R., & Wang, L. (2025). Robust vision-based measurement of structural vibration under highly perturbed ego motion by Lucas-

Kanade-based camera motion compensation. *Measurement Science and Technology*.

147. Elasri, A., Cherroun, L., & Nadour, M. (2024). Robotic Visual-Based Navigation Structures Using Lucas-Kanade and Horn-Schunck Algorithms of Optical Flow. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, *48*(3), 1149-1172.

148. Zhang, Y., Li, H., Yan, X., Ye, Y., Ren, Q., & Hu, S. (2025). Decoupling the factors influencing methanol oxidation performance of Pt3SnCux system based on partial least squares method. *Chemical Engineering Journal*, 162566.

149. Hariguna, T., & Ruangkanjanases, A. (2024). Assessing the impact of artificial intelligence on customer performance: A quantitative study using partial least squares methodology. *Data Science and Management*, *7*(3), 155-163.

150. Wang, S., Cheah, J. H., Wong, C. Y., & Ramayah, T. (2024). Progress in partial least squares structural equation modeling use in logistics and supply chain management in the last decade: a structured literature review. *International Journal of Physical Distribution & Logistics Management*, *54*(7/8), 673-704.

151. Bahrami, N., & Siadatmousavi, S. M. (2024). Ship voyage optimisation considering environmental forces using the iterative Dijkstra's algorithm. *Ships and Offshore Structures*, *19*(8), 1173-1180.

152. Tilanterä, A., Sorva, J., Seppälä, O., & Korhonen, A. (2024, August). Students Struggle with Concepts in Dijkstra's Algorithm. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1* (pp. 154-165).

153. Shah, M. H., Li, J., & Liu, M. (2024). On Scalable Design for User-Centric Multi-Modal Shared E-Mobility Systems using MILP and Modified Dijkstra's Algorithm. *arXiv preprint arXiv:2412.10986*.

154. Bhat, R., Rao, P. K., Kamath, C. R., Tandon, V., & Vizzapu, P. (2024). Comparative analysis of Bellman-Ford and Dijkstra’s algorithms for optimal

evacuation route planning in multi-floor buildings. *Cogent Engineering*, *11*(1), 2319394.

155. Siringoringo, Y. B. P., Sitinjak, L., & Tarigan, E. D. (2024). Determining the Location of Nenas Processing Factories in North Sumatra Using Dijkstra Algorithm. *Journal of Research in Mathematics Trends and Technology*, *6*(1), 1-7.

156. Nuritdinov, J. T., Kakharov, S. S., & Dagur, A. (2024). A new algorithm for finding the Minkowski difference of some sets. In *Artificial Intelligence and Information Technologies* (pp. 142-147). CRC Press.

157. Trinh, D. C., Mac, A. T., Dang, K. G., Nguyen, H. T., Nguyen, H. T., & Bui, T. D. (2024). Alpha-EIOU-YOLOv8: an improved algorithm for rice leaf disease detection. *AgriEngineering*, *6*(1), 302-317.

158. Rahardja, U., Sari, A., Alsalamy, A. H., Askar, S., Alawadi, A. H. R., & Abdullaeva, B. (2024). Tribological properties assessment of metallic glasses through a genetic algorithm-optimized machine learning model. *Metals and Materials International*, *30*(3), 745-755.

159. Wen, X., Yin, G., Liu, T., Ong, M. C., Wang, C., & Wang, K. (2025). An optimization framework for mooring design of floating offshore wind turbines using a genetic algorithm based on a surrogate model. *Renewable Energy*, *245*, 122807.

160. Ahmadipour, M., Othman, M. M., Bo, R., Javadi, M. S., Ridha, H. M., & Alrifaey, M. (2024). Optimal power flow using a hybridization algorithm of arithmetic optimization and aquila optimizer. *Expert Systems with Applications*, *235*, 121212.

161. Liu, H., Wang, K., Wang, Y., Zhang, M., Liu, Q., & Li, W. (2025). An Enhanced Algorithm for Detecting Small Traffic Signs Using YOLOv10. *Electronics*, *14*(5), 955.

162. Ding, X., Yao, R., & Khezri, E. (2024). An efficient algorithm for optimal route node sensing in smart tourism Urban traffic based on priority constraints. *Wireless Networks*, *30*(9), 7189-7206.

163. Yang, G., & Yu, L. (2024). A chimp algorithm based on the foraging strategy of manta rays and its application. *Plos one*, *19*(3), e0298230.

164. Almeida, A., Brás, S., Sargento, S., & Pinto, F. C. (2024). Focalize K-NN: an imputation algorithm for time series datasets. *Pattern Analysis and Applications*, *27*(2), 39.

165. Almeida, A., Brás, S., Sargento, S., & Pinto, F. C. (2024). Focalize K-NN: an imputation algorithm for time series datasets. *Pattern Analysis and Applications*, *27*(2), 39.

166. Moges, D. M., & Wordofa, B. G. (2024). An efficient algorithm for solving multilevel multi-objective linear fractional optimization problem with neutrosophic parameters. *Expert Systems with Applications*, *257*, 125122.

167. Park, H., & Lee, C. (2024). An exact algorithm for maximum electric vehicle flow coverage problem with heterogeneous chargers, nonlinear charging time and route deviations. *European Journal of Operational Research*, *315*(3), 926-951.

168. Svensson, T., Madhawa, K., Chung, U. I., & Svensson, A. K. (2024). Validity and reliability of the Oura Ring Generation 3 (Gen3) with Oura sleep staging algorithm 2.0 (OSSA 2.0) when compared to multi-night ambulatory polysomnography: A validation study of 96 participants and 421,045 epochs. *Sleep Medicine*, *115*, 251-263.

169. Zhou, F. Y., Yapp, C., Shang, Z., Daetwyler, S., Marin, Z., Islam, M. T., ... & Danuser, G. (2024). A general algorithm for consensus 3D cell segmentation from 2D segmented stacks. *bioRxiv*, 2024-05.

170. Building a Route for a Mobile Robot Based on the BRRT and A*(H-BRRT) Algorithms for the Effective Development of Technological Innovations / Amer Abu-Jassar, Hassan Al-Sukhni, Yasser Al-Sharo, S. Maksymova, V. Yevsieiev, V. Lyashenko // International Journal of Engineering Trends and Technology. – 2024. – V. 72(11). – P. 294-306.

171. Yevsieiev V. Comparative Analysis of Modifications of RRT Algorithms for Route Planning of a Mobile Robot/ V. Yevsieiev // Computer-

integrated technologies, automation and robotics 2024 : proceedings of the I st All-Ukrainian Conference, Kharkiv, May 16-17, 2024. – Kharkiv, 2024. – P. 25-28.

172. Ayalew, W., Menebo, M., Merga, C., & Negash, L. (2024). Optimal path planning using bidirectional rapidly-exploring random tree star-dynamic window approach (BRRT*-DWA) with adaptive Monte Carlo localization (AMCL) for mobile robot. *Engineering Research Express*, *6*(3), 035212.

173. Li, M., Shan, L., Wu, Z., & Wang, W. (2024, October). An Adaptive Multimodal Enhanced RRT* Path Planning Algorithm Based on Node Collision Feature Calculation Model. In *2024 IEEE International Conference on Unmanned Systems (ICUS)* (pp. 982-988). IEEE.

174. Ni, X., Hu, W., Fan, Q., Cui, Y., & Qi, C. (2024). A Q-learning based multi-strategy integrated artificial bee colony algorithm with application in unmanned vehicle path planning. *Expert Systems with Applications*, *236*, 121303.

175. Zhong, W., Zheng, S., Huang, J., Zhou, Z., Wu, M., & Yu, R. (2025). Integrating Global Path Information with Advanced Risk Assessment: An Enhanced Potential Field Method for Intelligent Connected Vehicles Local Path Planning. *IEEE Sensors Journal*.

176. Gargano, I. E., von Ellenrieder[1], K. D., & Vivolo, M. (2025). A Survey of Trajectory Planning Algorithms for Off-Road Uncrewed. In *Modelling and Simulation for Autonomous Systems: 10th International Conference, MESAS 2023, Palermo, Italy, October 17–19, 2023, Revised Selected Papers* (Vol. 14615, p. 120). Springer Nature.

177. Gargano, I. E., von Ellenrieder, K. D., & Vivolo, M. (2025). A Survey of Trajectory Planning Algorithms for Off-Road Uncrewed Ground Vehicles. In *International Conference on Modelling and Simulation for Autonomous Systems* (pp. 120-148). Springer, Cham.

178. Aung, Z., Myo, A. K., War, N. N., & Mikhailovna, G. O. (2024, March). Experimental Study of Algorithms for Planning the Trajectory of a Warehouse Mobile Robot Based on Reinforcement Learning. In *2024 IEEE Conference on Computer Applications (ICCA)* (pp. 1-6). IEEE.

179. Huang, Z., Ye, G., Yang, P., & Yu, W. (2025). Application of multi-sensor fusion localization algorithm based on recurrent neural networks. *Scientific Reports*, *15*(1), 8195.

180. Jiang, Z., Liu, Q., Zheng, D., Shi, R., & Wang, Y. (2024, December). Research on Obstacle Avoidance Path Planning for UGV in Complex Environments Based on Fusion Algorithm. In *2024 2nd International Conference on Artificial Intelligence and Automation Control (AIAC)* (pp. 84-87). IEEE.

181. Otomo, K., & Ishikawa, K. (2024). Ground vehicle path planning on Uneven terrain Using UAV Measurement point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *48*, 321-326.

182. De Carvalho, G. P., Sawanobori, T., & Horii, T. (2025). Data-driven Motion Planning: A Survey on Deep Neural Networks, Reinforcement Learning, and Large Language Model Approaches. *IEEE Access*.

183. Zhang, Z., Chen, Y., Han, F., Fan, J., Yu, H., Zhang, H., & Wang, Y. (2024). Wgit*: Workspace-guided informed tree for motion planning in restricted environments. *IEEE/ASME Transactions on Mechatronics*.

184. Zhou, R., Zhang, C., Zhao, R., & Zhang, T. (2024). RSP-UV: real-time sampling-based path planning method for unmanned vehicles. *Physica Scripta*, *100*(1), 016011.

185. Kaklis, D., Kontopoulos, I., Varlamis, I., Emiris, I. Z., & Varelas, T. (2024). Trajectory mining and routing: A cross-sectoral approach. *Journal of Marine Science and Engineering*, *12*(1), 157.

186. Chen, X., Liu, S., Zhao, J., Wu, H., Xian, J., & Montewka, J. (2024). Autonomous port management based AGV path planning and optimization via an ensemble reinforcement learning framework. *Ocean & Coastal Management*, *251*, 107087.

187. Grujic, Z., & Grujic, B. (2025). Optimal Routing in Urban Road Networks: A Graph-Based Approach Using Dijkstra's Algorithm. *Applied Sciences*, *15*(8), 4162.

188. Sampath, M., Duraisamy, A. K., Samuel, A. M. R., & Malu, Y. D. M. (2024). Unmanned Aerial Vehicle Path Planning Using Water Strider Algorithm. *International Journal of Industrial Engineering: Theory, Applications and Practice*, *31*(3).

189. Zhao, T., Yan, Z., Zhang, B., Zhang, P., Pan, R., Yuan, T., ... & Chen, S. (2024). A comprehensive review of process planning and trajectory optimization in arc-based directed energy deposition. *Journal of Manufacturing Processes*, *119*, 235-254.

190. Wu, C., Pan, H., Luo, Z., Liu, C., & Huang, H. (2024). Multi-objective optimization of residential building energy consumption, daylighting, and thermal comfort based on BO-XGBoost-NSGA-II. *Building and Environment*, *254*, 111386.

191. Fan, Y., Xu, H., Liu, M., Zhuo, Q., & Zhang, T. (2025). JPDS-NN: Reinforcement Learning-Based Dynamic Task Allocation for Agricultural Vehicle Routing Optimization. *arXiv preprint arXiv:2503.02369*.

192. Ping, G., Zhu, M., Ling, Z., & Niu, K. (2024). Research on optimizing logistics transportation routes using AI large models. *Spectrum of Research*, *4*(1).

193. Smith, J., Hall, S., Coombs, G., Abbot, H., Fekry, A., Thorne, M., ... & Fox, M. (2025). Path-Planning on a Spherical Surface with Disturbances and Exclusion Zones. *Journal of Artificial Intelligence Research*, *82*, 1845-1907.

194. Singh, R. K., Modgil, S., & Shore, A. (2024). Building artificial intelligence enabled resilient supply chain: a multi-method approach. *Journal of Enterprise Information Management*, *37*(2), 414-436.

195. Dat, P. V. T., Doan, L., & Binh, H. T. T. (2025, April). Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 39, No. 25, pp. 26931-26938).

196. Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., ... & Zhang, Q. (2024). Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*.

197. Ghiaskar, A., Amiri, A., & Mirjalili, S. (2024). Polar fox optimization algorithm: a novel meta-heuristic algorithm. *Neural Computing and Applications*, *36*(33), 20983-21022.

198. Ghiaskar, A., Amiri, A., & Mirjalili, S. (2024). Polar fox optimization algorithm: a novel meta-heuristic algorithm. *Neural Computing and Applications*, *36*(33), 20983-21022.

199. Campos, D. G., Fütterer, T., Gfrörer, T., Lavelle-Hill, R., Murayama, K., König, L., ... & Scherer, R. (2024). Screening smarter, not harder: A comparative analysis of machine learning screening algorithms and heuristic stopping criteria for systematic reviews in educational research. *Educational Psychology Review*, *36*(1), 19.

200. Prokop, K., & Połap, D. (2024). Heuristic-based image stitching algorithm with automation of parameters for smart solutions. *Expert Systems with Applications*, *241*, 122792.

201. Lewis, L., Gilboa, D., & McClean, J. R. (2025). Quantum advantage for learning shallow neural networks with natural data distributions. *arXiv preprint arXiv:2503.20879*.

202. Salman, H. A., Kalakech, A., & Steiti, A. (2024). Random forest algorithm overview. *Babylonian Journal of Machine Learning*, *2024*, 69-79.

203. Kashyap, V., Styliaris, G., Mouradian, S., Cirac, J. I., & Trivedi, R. (2025). Accuracy Guarantees and Quantum Advantage in Analog Open Quantum Simulation with and without Noise. *Physical Review X*, *15*(2), 021017.

204. Wang, W. C., Tian, W. C., Xu, D. M., & Zang, H. F. (2024). Arctic puffin optimization: A bio-inspired metaheuristic algorithm for solving engineering design optimization. *Advances in Engineering Software*, *195*, 103694.

205. Berry, D. W., Su, Y., Gyurik, C., King, R., Basso, J., Barba, A. D. T., ... & Babbush, R. (2024). Analyzing prospects for quantum advantage in topological data analysis. *PRX Quantum*, *5*(1), 010319.

206. Delgado-Granados, L. H., Krogmeier, T. J., Sager-Smith, L. M., Avdic, I., Hu, Z., Sajjan, M., ... & Mazziotti, D. A. (2025). Quantum algorithms and applications for open quantum systems. *Chemical Reviews*, *125*(4), 1823-1839.

207. Bo, Y., Lu, H., Li, H., Gao, D., Pan, Z., & Jiang, L. Optimal Hybrid Pv-Teg Systems Reconfigurations for Effective Mitigation of Partial Shading Conditions Via Cooperative Q-Learning and Advantage Actor-Critic Algorithm. *Available at SSRN 5115130*.

208. Andersen, J. E., & Shan, S. (2025). Estimating the Percentage of GBS Advantage in Gaussian Expectation Problems. *arXiv preprint arXiv:2502.19362*.

209. Roussel, R., Edelen, A. L., Boltz, T., Kennedy, D., Zhang, Z., Ji, F., ... & Neiswanger, W. (2024). Bayesian optimization algorithms for accelerator physics. *Physical review accelerators and beams*, *27*(8), 084801.

210. Lan, G., Han, D. J., Hashemi, A., Aggarwal, V., & Brinton, C. G. (2024). Asynchronous federated reinforcement learning with policy gradient updates: Algorithm design and convergence analysis. *arXiv preprint arXiv:2404.08003*.

211. Wang, J., Wang, W. C., Hu, X. X., Qiu, L., & Zang, H. F. (2024). Black-winged kite algorithm: a nature-inspired meta-heuristic for solving benchmark functions and engineering problems. *Artificial Intelligence Review*, *57*(4), 98.

212. Wang, S., Cao, L., Chen, Y., Chen, C., Yue, Y., & Zhu, W. (2024). Gorilla optimization algorithm combining sine cosine and cauchy variations and its engineering applications. *Scientific Reports*, *14*(1), 7578.

213. Yuan, C., Zhao, D., Heidari, A. A., Liu, L., Chen, Y., & Chen, H. (2024). Polar lights optimizer: Algorithm and applications in image segmentation and feature selection. *Neurocomputing*, *607*, 128427.

214. Li, X., Yin, X., Wiebe, N., Chun, J., Schenter, G. K., Cheung, M. S., & Mülmenstädt, J. (2025). Potential quantum advantage for simulation of fluid dynamics. *Physical Review Research*, *7*(1), 013036.

215. Kooshari, A., Fartash, M., Mihannezhad, P., Chahardoli, M., AkbariTorkestani, J., & Nazari, S. (2024). An optimization method in wireless sensor network routing and IoT with water strider algorithm and ant colony optimization algorithm. *Evolutionary Intelligence*, *17*(3), 1527-1545.

216. Rashidovna, M. Z., & Ochilovna, M. M. (2025). PYTHON DASTURLASH TILINING ASOSIY KUTUBXONALAR YORDAMIDA 3D GRAFIKLAR YARATISH. *PEDAGOGIK TADQIQOTLAR JURNALI*, *2*(2), 289-295.

217. Kaur, K., Kaur, P., Kaur, J., & Channi, H. K. (2024, November). Comprehensive Analysis of Domestic Energy Consumption Using Python and Matplotlib. In *2024 2nd International Conference on Advancements and Key Challenges in Green Energy and Computing (AKGEC)* (pp. 1-6). IEEE.

218. Manuhara, N. F., & Nugroho, Y. S. (2024, June). Python Library Dynamics: A Study of Patterns and Project Complexity on GitHub. In *2024 International Conference on Smart Computing, IoT and Machine Learning (SIML)* (pp. 279-285). IEEE.

219. Baxridtdinovich, H. B. (2024). PYTHON DASTURLASH TILI VA UNING DASTURIY TA'MINOT SOHASIDAGI O'RNI. *MASTERS*, *2*(12), 41-48.

220. Hazrat, R. (2024). The matplotlib Library and Projects. In *A Course in Python: The Core of the Language* (pp. 211-246). Cham: Springer Nature Switzerland.

221. Savvidis, L. S., Mavromoustakis, C. X., Mastorakis, G. N., & Markakis, E. K. (2025). On the Implementation of a Transverse Wave-Based Routing Technique for a TSP Circular Instance. *IEEE Access*.

222. Chen, X., Hu, R., Luo, K., Wu, H., Biancardo, S. A., Zheng, Y., & Xian, J. (2025). Intelligent ship route planning via an A∗ search model enhanced double-deep Q-network. *Ocean Engineering*, *327*, 120956.

223. Hashali, S. D., Yang, S., & Xiang, X. (2024). Route planning algorithms for unmanned surface vehicles (usvs): a comprehensive analysis. *Journal of Marine Science and Engineering*, *12*(3), 382.

224. Rosendo, B., Fortunato, D., & Abreu, R. (2025). Quantum Approaches for Vehicle Routing Optimization on NISQ Platforms. *IEEE Software*.

225. Yevsieiev V. Building a traffic route taking into account obstacles based on the A-star algorithm using the python language / V. Yevsieiev V., Amer Abu-Jassar, S. Maksymova // Technical Science Research In Uzbekistan. – 2024. – Vol. 2(3). – P. 103-112.

226. Kang, J., Zhu, X., Shen, L., & Li, M. (2024). Fault diagnosis of a wave energy converter gearbox based on an Adam optimized CNN-LSTM algorithm. *Renewable Energy*, *231*, 121022.

227. Zhang, H., Jiang, X., Wang, F., & Yang, X. (2024). The time two-grid algorithm combined with difference scheme for 2D nonlocal nonlinear wave equation. *Journal of Applied Mathematics and Computing*, *70*(2), 1127-1151.

228. Huseyn, R., Hashimov, A., Shokri, A., & Mukalazi, H. (2024). Calculation algorithm for electromagnetic wave processes in multi-wire intersystem ETL. *Discover Electronics*, *1*(1), 5.

229. Liu, Y., Tao, M., Shi, T., Wang, J., Wang, J., & Mao, X. (2024). Sub-aperture polar format algorithm for curved trajectory millimeter wave radar imaging. *IEEE Transactions on Radar Systems*, *2*, 67-83.

230. Khurshid, H., Mohammed, B. S., Al-Yacoubya, A. M., Liew, M. S., & Zawawi, N. A. W. A. (2024). Analysis of hybrid offshore renewable energy sources for power generation: a literature review of hybrid solar, wind, and waves energy systems. *Developments in the Built Environment*, 100497.

231. Graml, M., Zollner, K., Hernangómez-Pérez, D., Faria Junior, P. E., & Wilhelm, J. (2024). Low-scaling GW algorithm applied to twisted transition-metal dichalcogenide heterobilayers. *Journal of Chemical Theory and Computation*, *20*(5), 2202-2208.

232. Shao, W., Hu, Y., Jiang, X., & Zhang, Y. (2024). Wave retrieval from quad-polarized Chinese Gaofen-3 SAR image using an improved tilt modulation transfer function. *Geo-Spatial Information Science*, *27*(5), 1405-1423.

233. Cao, C., Bao, L., Gao, G., Liu, G., & Zhang, X. (2024). A novel method for ocean wave spectra retrieval using deep learning from sentinel-1 wave mode data. *IEEE Transactions on Geoscience and Remote Sensing*.

234. Liu, Y., & Adams, I. S. (2025). Tomographic reconstruction algorithms for retrieving two-dimensional ice cloud microphysical parameters using along-track (sub) millimeter-wave radiometer observations. *Atmospheric Measurement Techniques*, *18*(7), 1659-1674.

235. Constructing an Optimal Route for a Mobile Robot Using a Wave Algorithm / I. Nevliudov, S. Maksymova, V. Yevsieiev, Nur Uluhan // Journal of natural sciences and technologies. – 2024. – Vol. 3 (1). – P. 282-289.

236. Route constructing for a mobile robot based on the D-star algorithm / V. Yevsieiev, Amer Abu-Jassar, S. Maksymova, Ahmad Alkhalaileh // Technical Science Research in Uzbekistan. – 2024. – Vol. 2(4). – P. 55-66.

237. Katona, K., Neamah, H. A., & Korondi, P. (2024). Obstacle avoidance and path planning methods for autonomous navigation of mobile robot. *Sensors*, *24*(11), 3573.

238. Saad, A., Abo El-Lail, A. S., & Zidane, I. (2025). Development of Path Planning Approach for Mobile Robot In Static Environment. *JES. Journal of Engineering Sciences*, 45-62.

239. Saad, A., Abo El-Lail, A. S., & Zidane, I. (2025). Development of Path Planning Approach for Mobile Robot In Static Environment. *JES. Journal of Engineering Sciences*, 45-62.

240. Min Lin, X., Xia, L., Ge, Z., Hong, X., Zhao, R., & Jalalnezhad, M. (2024). Designing the optimal path curve based on spline functions for mobile robot using the combination of bee colony algorithm and genetic algorithm. *Journal of Vibration and Control*, 10775463241240802.

241. Ušinskis, V., Nowicki, M., Dzedzickis, A., & Bučinskas, V. (2025). Sensor-fusion based navigation for autonomous mobile robot. *Sensors*, *25*(4), 1248.

242. Shoeib, M. A., Lewandowski, J., & Omara, A. M. (2024). A novel methodology for vision-based path planning and obstacle avoidance in mobile robot applications. *Advanced Robotics*, *38*(12), 802-817.

243. Li, S., Huang, J., Mao, W., Yang, Y., & Nie, J. (2025). TOACA-DWA: A path planning algorithm for mobile robot in dynamic environments. *Journal of Intelligent & Fuzzy Systems*, *48*(3), 245-261.

244. Chen, Y., Fan, Y., & Jin, M. (2024). Research on Sensor Technology in Mobile Robot Navigation. *Applied and Computational Engineering*, *93*, 50-55.

245. Numbi, J., Zioui, N., & Tadjine, M. (2025). Quantum Particle Swarm Optimisation Proportional–Derivative Control for Trajectory Tracking of a Car-like Mobile Robot. *Electronics*, *14*(5), 832.

246. Zhang, W., Cui, Z., Wang, N., & Lan, Y. (2024). Research on Path Planning Algorithm Based on Fast Target Detection. *Journal of Artificial Intelligence Practice (2)*.

247. Chambouleyron, V., Sengupta, A., Salama, M., van Kooten, M., Gerard, B. L., Haffert, S. Y., ... & Macintosh, B. (2024). Using the Gerchberg–Saxton algorithm to reconstruct nonmodulated pyramid wavefront sensor measurements. *Astronomy & Astrophysics*, *681*, A48.

248. Zhang, Q., Jing, X., Sun, L., Wei, X., Wan, X., & Zheng, J. (2025). Using adaptive genetic algorithm optimize to realize the extended depth-of-field wavefront coding mask for a large field-of-view microscopy system. *Optics and Lasers in Engineering*, *185*, 108733.

249. Coyle, C., Kanella, I., Mann, I., Qureshi, N., Linton, N. W., & Kanagaratnam, P. (2024). RETRO-mapping: A novel algorithm automating wavefront categorization using activation mapping during persistent atrial fibrillation demonstrates a reduction in wavefront collisions following pulmonary vein isolation. *Journal of Cardiovascular Electrophysiology*, *35*(3), 557-568.

250. Guo, Y., Tang, T., Wang, Q., Yang, C., Xia, Z., Peng, C., ... & Cui, Y. (2024, December). WFA-vect: a SIMD wavefront algorithm for gap-affine

pairwise alignment. In *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (pp. 528-533). IEEE.

251. Ren, H., Lu, Z., Li, G., Zhang, Y., Yang, X., Guo, Y., ... & Du, Y. (2024). A High-Resolution Spotlight Imaging Algorithm via Modified Second-Order Space-Variant Wavefront Curvature Correction for MEO/HM-BiSAR. *Remote Sensing*, *16*(24).

252. Liu, S., Zhong, H., Li, Y., & Liu, K. (2024). Fast and Highly Accurate Zonal Wavefront Reconstruction from Multi-Directional Slope and Curvature Information Using Subregion Cancelation. *Applied Sciences*, *14*(8), 3476.

253. Wen, L., Mei, X., Tan, Y., Zhang, Z., Chai, F., Wu, J., ... & Yang, P. (2024, September). Cross-Correlation Algorithm Based on Speeded-Up Robust Features Parallel Acceleration for Shack–Hartmann Wavefront Sensing. In *Photonics* (Vol. 11, No. 9, p. 844). MDPI.

254. Guan, H., Zhao, W., Wang, S., Yang, K., Zhao, M., Liu, S., ... & Yang, P. (2024). Higher-resolution wavefront sensing based on sub-wavefront information extraction. *Frontiers in Physics*, *11*, 1336651.

255. Fan, X., Yang, Y., Zhang, Y., & Wang, X. (2024). Design of wavefront detection system for eyeglasses based on transmissive phase measuring deflectometry using gamma error correction algorithm. *Physica Scripta*, *99*(8), 085522.

256. Ren, H., Lu, Z., Li, G., Zhang, Y., Yang, X., Guo, Y., ... & Du, Y. (2024). A High-Resolution Spotlight Imaging Algorithm via Modified Second-Order Space-Variant Wavefront Curvature Correction for MEO/HM-BiSAR. *Remote Sensing*, *16*(24).

257. Yevsieiev V. Route planning for a mobile robot in 3d space based on an algorithm probabilistic roadmap / V. Yevsieiev, S. Maksymova, Amer Abu-Jassar // Journal of Universal Science Research. – 2024. – Vol. 2(4). – P. 22-33.

258. Rajasa Pohan, M. A., & Utama, J. (2024). SMART PROBABILISTIC ROAD MAP (SMART-PRM): FAST ASYMPTOTICALLY OPTIMAL PATH

PLANNING USING SMART SAMPLING STRATEGIES. *Jordanian Journal of Computers & Information Technology*, *10*(2).

259. Jathunga, T., & Rajapaksha, S. (2025). Improved Path Planning for Multi-Robot Systems Using a Hybrid Probabilistic Roadmap and Genetic Algorithm Approach. *Journal of Robotics and Control (JRC)*, *6*(2), 715-733.

260. Xiong, Y., Wang, C., Li, B., & Wu, S. (2024, May). Improved probabilistic roadmap method based on obstacle distance sampling. In *2024 36th Chinese Control and Decision Conference (CCDC)* (pp. 5554-5559). IEEE.

261. Aviram, S., & Levner, E. (2025). Enhancing Path Planning for Autonomous Robots in Large, Obstacle-Crowded Environments: A Practical Improvement to the PRM Algorithm. *Journal of Robotics*, *2025*(1), 9569965.

262. Bao, J., & Yonetani, R. (2025). Path Planning using Instruction-Guided Probabilistic Roadmaps. *arXiv preprint arXiv:2502.16515*.

263. Schlapbach, J., & Schopferer, S. (2024, December). Time-Aware Probabilistic Roadmaps for Multi-Query Path Planning in Dynamic Environments. In *2024 Eighth IEEE International Conference on Robotic Computing (IRC)* (pp. 9-16). IEEE.

264. Zhou, Y., Lu, Y., & Lv, L. (2024). Grid-based non-uniform probabilistic roadmap-based agv path planning in narrow passages and complex environments. *Electronics*, *13*(1), 225.

265. Bana, F. R., Alsayed, A., Krajník, T., & Arvin, F. (2024, February). Iterative Risk Aware PRM Path Planning Algorithm for Autonomous Unknown Environments Exploration. In *2024 10th International Conference on Automation, Robotics and Applications (ICARA)* (pp. 133-138). IEEE.

266. Zhou, X., Wang, X., Xie, Z., Gao, J., Li, F., & Gu, X. (2024). A Collision-free path planning approach based on rule guided lazy-PRM with repulsion field for gantry welding robots. *Robotics and Autonomous Systems*, *174*, 104633.

267. Zheng, D., Ridderhof, J., Zhang, Z., Tsiotras, P., & Agha-Mohammadi, A. A. (2024). CS-BRM: A probabilistic roadmap for consistent belief space

planning with reachability guarantees. *IEEE Transactions on Robotics*, *40*, 1630-1649.

268. Stone, R., Wang, J., & Sha, Z. (2025). Risk-Bounded and Probabilistic Roadmap-Based Motion Planner for Arbitrarily-Shaped Robots with Uncertainty. *Journal of Computing and Information Science in Engineering*, 1-11.

269. Wu, G., Guo, L., Shi, D., Han, B., & Yang, F. (2025). Hybrid Probabilistic Road Map Path Planning for Maritime Autonomous Surface Ships Based on Historical AIS Information and Improved DP Compression. *Journal of Marine Science and Engineering*, *13*(1), 184.

270. Jamshidi, A., & Khanmirza, E. (2024, December). Improved Probabilistic Roadmap Path Planning Algorithm. In *2024 12th RSI International Conference on Robotics and Mechatronics (ICRoM)* (pp. 362-367). IEEE.

271. Rajasa Pohan, M. A., & Utama, J. (2024). SMART PROBABILISTIC ROAD MAP (SMART-PRM): FAST ASYMPTOTICALLY OPTIMAL PATH PLANNING USING SMART SAMPLING STRATEGIES. *Jordanian Journal of Computers & Information Technology*, *10*(2).

272. Jathunga, T., & Rajapaksha, S. (2025). Improved Path Planning for Multi-Robot Systems Using a Hybrid Probabilistic Roadmap and Genetic Algorithm Approach. *Journal of Robotics and Control (JRC)*, *6*(2), 715-733.

273. Noh, G., Park, J., Kim, J., Kim, J., & Lee, D. (2024). Real-time Path Re-planning to Deal with Dynamic Obstacles Using a Parallel Probabilistic Roadmap. *International Journal of Aeronautical and Space Sciences*, 1-12.

274. Aviram, S., & Levner, E. (2025). Enhancing Path Planning for Autonomous Robots in Large, Obstacle-Crowded Environments: A Practical Improvement to the PRM Algorithm. *Journal of Robotics*, *2025*(1), 9569965.

275. Singh, R. (2024). Trajectory optimization with hybrid probabilistic roadmap approach to achieve time efficient navigation of unmanned vehicles in unstructured environment. *Robotic Intelligence and Automation*, *44*(1), 164-189.

276. Bao, J., & Yonetani, R. (2025). Path Planning using Instruction-Guided Probabilistic Roadmaps. *arXiv preprint arXiv:2502.16515*.

278. Wullt, B., Norrlöf, M., Mattsson, P., & Schön, T. B. (2025). Probabilistic Bubble Roadmap. *arXiv preprint arXiv:2502.16205*.

279. Bana, F. R., Alsayed, A., Krajník, T., & Arvin, F. (2024, February). Iterative Risk Aware PRM Path Planning Algorithm for Autonomous Unknown Environments Exploration. In *2024 10th International Conference on Automation, Robotics and Applications (ICARA)* (pp. 133-138). IEEE.

280. Abujabal, N. A., Baziyad, M., Fareh, R., Khadraoui, S., & Bettayeb, M. (2024, June). Path Planning Technique based on Artificial Potential Field and Probabilistic Roadmap for Narrow Passages. In *2024 Advances in Science and Engineering Technology International Conferences (ASET)* (pp. 1-6). IEEE.

281. Sarbini, R. N., Ahmad, I. R. D. A. M., Bura, R. O., & Simbolon, L. U. H. U. T. (2024). Development of pathfinding using a-star and d-star lite algorithms in video game. *Journal of Theoretical and Applied Information Technology*, *102*(3), 832-841.

282. Suprapto, B. Y., Dwijayanti, S., Windisari, D., & Pratama, G. A. (2025). Optimizing Route Planning for Autonomous Electric Vehicles Using the D-Star Lite Algorithm. *International Journal of Advanced Computer Science & Applications*, *16*(1).

283. Wu, M., Jiang, Y., & Wang, S. (2024, May). Optimized Search Direction Based A-Star and DWA Fusion Algorithm for Path Planning of Automated Guided Vehicle. In *2024 IEEE 13th Data Driven Control and Learning Systems Conference (DDCLS)* (pp. 1597-1602). IEEE.

284. Shen, L. H., Wu, P. C., Ku, C. J., Li, Y. T., Feng, K. T., Liu, Y., & Hanzo, L. (2024). D-STAR: Dual simultaneously transmitting and reflecting reconfigurable intelligent surfaces for joint uplink/downlink transmission. *IEEE Transactions on Communications*, *72*(6), 3305-3322.

285. Sridhar, S., Shendre, S. P., Kamble, S. S., Priya, M. Y., & Apurvanand, S. (2024, November). Performance Assessment of Algorithms for Ship Routing through AnyLogic Simulations. In *2024 First International Conference for Women in Computing (InCoWoCo)* (pp. 1-8). IEEE.

286. Su, Y. (2024, November). Improved A* algorithm applied in dynamic environment for pathfinding. In *2nd International Conference on Mechatronic Automation and Electrical Engineering (ICMAEE 2024)* (Vol. 2024, pp. 294-299). IET.

287. Alferov, G., Korolev, V., Fedorov, V., & Khokhriakova, A. (2024). Correction of the movement of the mobile robot using the modified algorithm. In *E3S Web of Conferences* (Vol. 549, p. 08005). EDP Sciences.

288. Huang, J. Improved A-Star Algorithm to Implement a Path Planning Algorithm Under Multiple Constraints in 3d Space. *Available at SSRN 4891138*.

289. Wu, Y., Wei, Y., & Zhang, H. (2025). NanoDet Model-Based Tracking and Inspection of Net Cage Using ROV. *Aquaculture Research*, *2025*(1), 7715838.

290. Lou, C., Lu, H., Li, Z., & Ni, C. (2025, April). Optimized design of floor sweeping and mopping robot path. In *International Conference on Mechatronic Engineering and Artificial Intelligence (MEAI 2024)* (Vol. 13555, pp. 334-340). SPIE.

291. Xie, T., Yao, X., Jiang, Z., & Meng, J. (2025). AGV Path Planning with Dynamic Obstacles Based on Deep Q-Network and Distributed Training. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 1-17.

292. Banh, T. T., Nguyen, X. T., Le, T. N., Tran, T. B., & Vuong, B. T. (2024, February). Efficient Algorithms on Dynamic Obstacle Avoidance for Multi-Robot Agents In Automated Warehouse System. In *Proceedings of the 2024 9th International Conference on Intelligent Information Technology* (pp. 474-481).

293. Hou, R., & Zhang, P. (2024, December). A-star Ant Colony Fusion Optimisation for Unmanned Ground Vehicle Route Planning. In *Journal of Physics: Conference Series* (Vol. 2891, No. 11, p. 112001). IOP Publishing.

294. Suanpang, P., & Jamjuntr, P. (2024). Optimizing autonomous UAV navigation with d* algorithm for sustainable development. *Sustainability*, *16*(17), 7867.

295. Yan, X. Z., Zhou, X. Y., Ding, R. C., Luo, Q. H., & Ju, C. Y. (2025). A Path Planning Algorithm Based on A Heuristic Method. *Journal of Internet Technology*, *26*(2), 183-198.

296. Aman, A., Debnath, A., Kumar, S., Sarkar, R. D., Chaudhuri, R., & Deb, S. (2024, November). Adaptive Heuristic Pathfinding Algorithm (AHPA): A Dynamic Path Adjuster for Mobile Robots. In *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)* (pp. 588-594). IEEE.

297. Hsieh, S. Y., Le, H. O., Le, V. B., & Peng, S. L. (2024). On the d-Claw Vertex Deletion Problem. *Algorithmica*, *86*(2), 505-525.

298. Lu, S., & Shi, W. (2024). Testing for Vehicle Computing. In *Vehicle Computing: From Traditional Transportation to Computing on Wheels* (pp. 211-240). Cham: Springer Nature Switzerland.

299. Diyachuk, A. (2024, November). Comparative Analysis of Optimal Path Planning Algorithms Using Deep Learning Technique. In *2024 International Conference on Cyber-Physical Social Intelligence (ICCSI)* (pp. 1-6). IEEE.

300. Karamitsos, G., Bechtsis, D., Tsolakis, N., & Vlachos, D. (2024). Assessing Path Planning Algorithms of Mobile Robots: A ROS-Based Simulation Framework. In *Disruptive Technologies and Optimization Towards Industry 4.0 Logistics* (pp. 139-160). Cham: Springer International Publishing.

301. He, Y., Chen, H., & Shi, W. (2024). An Advanced Framework for Ultra-Realistic Simulation and Digital Twinning for Autonomous Vehicles. *arXiv preprint arXiv:2405.01328*.

302. Zhang, X. (2024, July). Lightweight Road Vehicle Detection Algorithm Based on Improved YOLOv8. In *2024 7th International Conference on Computer Information Science and Application Technology (CISAT)* (pp. 52-55). IEEE.

303. Qian, J., Wang, Z., & Yan, B. (2024, July). Research on dynamic local path planning for robots based on centerline extraction algorithm with improved DWA. In *Proceedings of the 2024 International Conference on Image Processing, Intelligent Control and Computer Engineering* (pp. 248-260).

Додаток А

Код реалізацій алгоритму Canny

```
import cv2
import time

def main():
    # Відкриття відеопотоку з камери (зазвичай 0 для вбудованої камери)
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Помилка при відкритті камери.")
        return
    # Создание окна для отображения исходного видео
    cv2.namedWindow('Original Video', cv2.WINDOW_NORMAL)
    # Створення вікна для відображення результату роботи алгоритму Canny
    cv2.namedWindow('Canny Edge Detection', cv2.WINDOW_NORMAL)
    while True:
        # Захват кадра из видеопотока
        ret, frame = cap.read()
        if not ret:
            print("Не вдалося отримати кадр.")
            break
        # Вимірювання часу початку обробки кадру
        start_time = time.time()
        # Перетворення зображення на відтінки сірого
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Применение алгоритма Canny
        edges = cv2.Canny(gray, 50, 150)
        # Відображення вихідного відео
        cv2.imshow('Original Video', frame)
        # Відображення результату роботи алгоритму Canny
```

```
        cv2.imshow('Canny Edge Detection', edges)
        # Замер времени окончания обработки кадра
        end_time = time.time()
        # Расчет скорости обработки кадров и построения контура
        processing_speed = 1 / (end_time - start_time)
        # Виведення результатів у термінал
        print(f" Швидкість обробки: {processing_speed:.2f} кадрів за секунду ")
        # Очікування натискання клавіші ESC для завершення програми
        key = cv2.waitKey(1)
        if key == 27:  # ESC
            break
    # Звільнення ресурсів та закриття вікон
    cap.release()
    cv2.destroyAllWindows()
if __name__ == "__main__":
    main()
```

Додаток Б

Код реалізації методу активних контурів

```
import cv2
import numpy as np

def active_contour(image_path, alpha=0.015, beta=10, gamma=0.001):
    # Завантаження зображення
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Вибір прямокутної області для ініціалізації контуру
    rect = cv2.selectROI("Select ROI", img)
    cv2.destroyAllWindows()
    # Отримання контуру на основі обраної області
    points = cv2.convexHull(np.array([[rect[0], rect[1]], [rect[0], rect[1] + rect[3]],
[rect[0] + rect[2], rect[1] + rect[3]], [rect[0] + rect[2], rect[1]]]))
    # Ініціалізація активного контуру
    snake = points.reshape((-1, 1, 2))
    # Процесс эволюции контура
    epsilon = 0.1  # Параметр для апроксимації
    snake = cv2.approxPolyDP(snake, epsilon, closed=True)
    # Відображення контуру на зображенні
    img_contour = img.copy()
    cv2.polylines(img_contour, [np.int32(snake)], isClosed=True, color=(0, 255, 0),
thickness=2)
    # Відображення вихідного зображення та зображення з контуром
    cv2.imshow('Original Image', img)
    cv2.imshow('Active Contour', img_contour)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
# Приклад використання
image_path = 'Pic/12.jpg'
active_contour(image_path)
```

Додаток В

Код реалізацій методу оптичного потоку та алгоритму Грехема

```
import cv2
import time
import numpy as np

def main():
    # Відкриття відеопотоку з камери
    cap = cv2.VideoCapture(0)
    # Перевірка успішного відкриття відеопотоку
    if not cap.isOpened():
        print("Помилка: Неможливо відкрити відеопотік.")
        return
    # Створення вікон для відображення відеопотоку та контуру
    cv2.namedWindow('Video', cv2.WINDOW_NORMAL)
    cv2.namedWindow('Graham Scan Contour', cv2.WINDOW_NORMAL)
    start_time = time.time()
    frame_count = 0
    # Читання першого кадру для ініціалізації методу оптичного потоку
    ret, prev_frame = cap.read()
    if not ret:
        print("Помилка: Неможливо отримати перший кадр.")
        return
    # Конвертація першого кадру у відтінки сірого
    prev_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)
    # Читання та обробка кадрів з відеопотоку
    while True:
        ret, frame = cap.read()
        if not ret:
```

```
            print("Помилка: Неможливо отримати кадр.")
            break
        # Конвертація поточного кадру у відтінки сірого
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Застосування оптичного потоку для отримання точок, що рухаються
        flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)
        # Преобразование оптического потока в список точек
        points = np.argwhere(np.abs(flow) > 2)
        # Застосування алгоритму Грехем для отримання контуру об'єкта з візуалізацією точок
        contour_image = apply_graham_scan(frame.copy(), points)
        # Відображення оригінального відео у вікні "Video"
        cv2.imshow('Video', frame)
        # Відображення контуру з візуалізацією крапок у вікні "Graham Scan Contour"
        cv2.imshow('Graham Scan Contour', contour_image)
        frame_count += 1
        # Вихід із програми при натисканні клавіші ESC
        if cv2.waitKey(1) & 0xFF == 27:
            break
        # Оновлення попереднього кадру та відтінків сірого
        prev_gray = gray.copy()
    end_time = time.time()
    elapsed_time = end_time - start_time
    # Обчислення швидкості обробки кадрів
    processing_speed = frame_count / elapsed_time
    # Виведення результатів розрахунків
    print("Швидкість обробки кадрів: {:.2f} кадрів за секунду".format(processing_speed))
```

```
    # Звільнення ресурсів та закриття вікон
    cap.release()
    cv2.destroyAllWindows()
def apply_graham_scan(frame, points):
    # Тут має бути ваш код для застосування алгоритму Грехема до точок об'єкта
    # У цьому прикладі відображаються точки об'єкта
    for point in points:
        cv2.circle(frame, (point[1], point[0]), 5, (0, 0, 255), -1)
    return frame
if __name__ == "__main__":
    main()
```

Додаток Г

Код реалізацій методу розпізнавання та відстеження об'єкта

```
import cv2
import numpy as np
import time

def process_frame(frame):
    # Засікаємо час початку обробки кадру
    start_time = time.time()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_yellow = np.array([20, 100, 100])
    upper_yellow = np.array([30, 255, 255])
    mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) > 0:
        largest_contour = max(contours, key=cv2.contourArea)
        M = cv2.moments(largest_contour)
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
            cv2.drawMarker(frame, (cx, cy), (0, 0, 255), cv2.MARKER_CROSS,
markerSize=40, thickness=3)
    # Засікаємо час закінчення обробки кадру
    end_time = time.time()
    # Розраховуємо часові інтервали
    processing_time = end_time - start_time
    detection_speed = 1 / processing_time if processing_time > 0 else 0
    # Виводимо результати
```

```
    print(f"Processing Time: {processing_time:.4f} seconds")
    print(f"Detection Speed: {detection_speed:.2f} FPS")
    # Відображення маски у вікні
    cv2.imshow('Color Mask', mask)
    # Відображення кадру з прицілом
    cv2.imshow('Video Stream', frame)
# Запуск відеопотоку з камери
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    process_frame(frame)
    # Вихід із циклу при натисканні клавіші 'q' або 'Esc'
    key = cv2.waitKey(1)
    if key == ord('q') or key == 27:  # 27 - код клавіші 'Esc'
        break
# Визволення ресурсів
cap.release()
cv2.destroyAllWindows()
```

Додаток Д

Код реалізацій алгоритму Sobel

```
import cv2
import time

# Функція для обробки відеопотоку
def process_video():
    # Відкриття відеопотоку з камери (зазвичай 0 для вбудованої камери)
    cap = cv2.VideoCapture(0)
    # Перевірка успішності відкриття відеопотоку
    if not cap.isOpened():
        print("Помилка: Неможливо відкрити камеру.")
        return
    # Створення вікна для вихідного відеопотоку
    cv2.namedWindow("Original Video", cv2.WINDOW_NORMAL)
    # Створення вікна для виведення результату роботи оператора Sobel
    cv2.namedWindow("Sobel Edge Detection", cv2.WINDOW_NORMAL)
    start_time = time.time()  # Час початку обробки відеопотоку
    while True:
        ret, frame = cap.read()  # Отримання кадру з відеопотоку
        if not ret:
            print("Помилка: Неможливо отримати кадр.")
            break
        # Перетворення зображення на відтінки сірого
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Застосування оператора Sobel
        sobel_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
        sobel_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
        edges = cv2.magnitude(sobel_x, sobel_y)
```

```
        # Відображення вихідного відеопотоку
        cv2.imshow("Original Video", frame)
        # Відображення результату роботи оператора Sobel
        cv2.imshow("Sobel Edge Detection", edges)
        # Обробка події натискання клавіші ESC для завершення програми
        key = cv2.waitKey(1) & 0xFF
        if key == 27:  # 27 - код клавіші ESC
            break
    end_time = time.time()  # Час закінчення обробки відеопотоку
    # Обчислення швидкості обробки кадрів
    frame_rate = cap.get(cv2.CAP_PROP_FPS)
    processing_speed = 1 / ((end_time - start_time) / frame_rate)
    # Виведення результатів розрахунків у термінал
    print("Швидкість обробки кадрів: {:.2f} кадрів за
секунду".format(frame_rate))
    print("Швидкість побудови контуру: {:.2f} кадрів за
секунду".format(processing_speed))
    # Звільнення ресурсів та закриття вікон
    cap.release()
    cv2.destroyAllWindows()
# Виклик функції обробки відеопотоку
process_video()
```

Додаток Е

Код реалізацій оцінки руху об'єкта в потоковому відео з камери, використовуючи метод Лукаса-Канаде

```
import cv2
import numpy as np
import time

# Функція обчислення швидкості кадрів
def calculate_fps(prev_time, current_time, frame_count):
    elapsed_time = current_time - prev_time
    fps = frame_count / elapsed_time
    return fps
# Функція для обчислення швидкості побудови контуру
def calculate_contour_speed(start_time, end_time, contour_count):
    elapsed_time = end_time - start_time
    contour_speed = contour_count / elapsed_time
    return contour_speed
# Відкриття відеопотоку з камери
cap = cv2.VideoCapture(0)
# Створення вікна для виведення відео
cv2.namedWindow("Video Stream")
# Змінні методу Лукаса-Канаде
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))
# Змінні алгоритму Грехема
corners = np.array([], dtype=np.float32)
while True:
```

```
    ret, frame = cap.read()
    if not ret:
        break
    # Перетворення зображення на відтінки сірого
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Якщо є кути, використовуємо їх у методі Лукаса-Канаді
    if len(corners):
        new_corners, status, _ = cv2.calcOpticalFlowPyrLK(old_gray, gray_frame,
corners, None, **lk_params)
        good_new = new_corners[status == 1]
        good_old = corners[status == 1]
        # Відображення ліній між старими та новими точками
        for i, (new, old) in enumerate(zip(good_new, good_old)):
            a, b = new.ravel()
            c, d = old.ravel()
            frame = cv2.line(frame, (int(a), int(b)), (int(c), int(d)), (0, 255, 0), 2)
    # Пошук кутів за допомогою алгоритму Грехема
    corners = cv2.goodFeaturesToTrack(gray_frame, 100, 0.01, 10)
    if corners is not None:
        corners = np.float32(corners)
        for corner in corners:
            x, y = corner.ravel()
            cv2.circle(frame, (int(x), int(y)), 3, (0, 0, 255), -1)
    cv2.imshow("Video Stream", frame)
    # Оновлення попереднього кадру
    old_gray = gray_frame.copy()
    # Перевірка натискання кнопки ESC
    key = cv2.waitKey(1)
    if key == 27:
        break
```

```
# Обчислення та виведення в термінал швидкості обробки кадрів
total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
frame_rate = cap.get(cv2.CAP_PROP_FPS)
processing_speed = total_frames / (time.time() - start_time)
print(f" Швидкість обробки кадрів: {processing_speed} кадрів/с ")
# Обчислення та виведення в термінал швидкості побудови контуру
contour_speed = calculate_contour_speed(start_time, time.time(), len(corners))
print(f" Швидкість побудови контуру: {contour_speed} точок/с ")
cap.release()
cv2.destroyAllWindows()
```

Додаток Є

Код реалізацій BRRT+A*

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import time

# Параметри середовища
width = 100
height = 100
# Початкова та кінцева точки
start_point = (10, 10)
goal_point = (5000, 5000)
# Параметри алгоритму BRRT и A*
num_iterations = 100
step_size = 30.0
# Генерація перешкод у вигляді блоків розміром 10x10
np.random.seed(42)  # Для відтворюваності
num_obstacles = 20
obstacles = [(np.random.randint(0, width - 10), np.random.randint(0, height - 10))
for _ in range(num_obstacles)]
# Функція для перевірки колізій із перешкодами
def is_collision_free(point):
    for obstacle in obstacles:
        if obstacle[0] <= point[0] < obstacle[0] + 10 and obstacle[1] <= point[1] <
obstacle[1] + 10:
            return False
    return True
# Функція для побудови маршруту за алгоритмом BRRT
```

```
def build_rrt(start, goal, num_iterations, step_size):
    tree = [start]
    for _ in range(num_iterations):
        random_point = np.random.rand(2) * np.array([width, height])
        nearest_point_index = np.argmin([np.linalg.norm(np.array(random_point) -
np.array(point)) for point in tree])
        nearest_point = tree[nearest_point_index]
        new_point = nearest_point + step_size * (random_point - nearest_point)
        new_point = tuple(new_point)
        if is_collision_free(new_point):
            tree.append(new_point)
    return tree
# Функція оптимізації маршруту з використанням A*
def optimize_path(rrt_tree, start, goal):
    graph = nx.Graph()
    for point in rrt_tree:
        graph.add_node(point)
    for i in range(len(rrt_tree) - 1):
        graph.add_edge(rrt_tree[i], rrt_tree[i + 1],
                       weight=np.linalg.norm(np.array(rrt_tree[i]) - np.array(rrt_tree[i +
1])))
    # Додамо початкову та кінцеву точки до графа
    graph.add_node(start)
    graph.add_node(goal)
    # Перевіримо наявність зв'язків з початковою та кінцевою точками
    if not nx.has_path(graph, start, rrt_tree[0]):
        graph.add_edge(start, rrt_tree[0], weight=np.linalg.norm(np.array(start) -
np.array(rrt_tree[0])))
    if not nx.has_path(graph, rrt_tree[-1], goal):
```

```
        graph.add_edge(rrt_tree[-1], goal, weight=np.linalg.norm(np.array(rrt_tree[-
1]) - np.array(goal)))
    # Виміряємо час виконання алгоритму
    start_time = time.time()
    result = nx.astar_path(graph, start, goal, heuristic=lambda n, goal:
np.linalg.norm(np.array(n) - np.array(goal)))
    end_time = time.time()
    execution_time = end_time - start_time
    return result, execution_time
# Функція для вимірювання довжини колії
def calculate_path_length(path):
    length = 0
    for i in range(len(path) - 1):
        length += np.linalg.norm(np.array(path[i]) - np.array(path[i + 1]))
    return length
# Функція для підрахунку кількості поворотів
def count_turns(path):
    turns = 0
    for i in range(1, len(path) - 1):
        vector1 = np.array(path[i - 1]) - np.array(path[i])
        vector2 = np.array(path[i + 1]) - np.array(path[i])
        cross_product = np.cross(vector1, vector2)
        if cross_product != 0:
            turns += 1
    return turns
# Функція для оцінки складності довкілля
def evaluate_environment_complexity(obstacles):
    return len(obstacles)
# Функція для оцінки загальної надійності та стабільності
def evaluate_reliability_and_stability(rrt_tree):
```

```
    return len(rrt_tree)
# Функція для оцінки вирішення проблем із виродженими випадками
def evaluate_handling_degenerate_cases(rrt_tree, optimal_path):
    # Приклад оцінки: чим менше поворотів в оптимальному шляху, тим
краще
    optimal_turns = count_turns(optimal_path)
    return optimal_turns
# Візуалізація початкової та кінцевої точок, перешкод
plt.scatter(*start_point, color='yellow', marker='o', label='Start')
plt.scatter(*goal_point, color='red', marker='o', label='Goal')
for obstacle in obstacles:
    plt.Rectangle((obstacle[0], obstacle[1]), 10, 10, color='black', alpha=0.5)
# Побудова маршруту
rrt_tree = build_rrt(start_point, goal_point, num_iterations, step_size)
# Оптимізація маршруту
optimal_path, execution_time = optimize_path(rrt_tree, start_point, goal_point)
# Візуалізація маршруту
for i in range(len(rrt_tree) - 1):
    plt.plot([rrt_tree[i][0], rrt_tree[i + 1][0]], [rrt_tree[i][1], rrt_tree[i + 1][1]],
color='blue', alpha=0.5)
# Візуалізація оптимального маршруту
for i in range(len(optimal_path) - 1):
    plt.plot([optimal_path[i][0], optimal_path[i + 1][0]], [optimal_path[i][1],
optimal_path[i + 1][1]], color='green',linewidth=2)
# Відображення графіки
plt.title('Optimized BRRT Path Planning+A*')
plt.legend()
plt.grid(True)
plt.show()
# Вимірювання довжини колії та кількості поворотів
```

```
path_length = calculate_path_length(optimal_path)
turns_count = count_turns(optimal_path)
# Оцінка критеріїв
environment_complexity = evaluate_environment_complexity(obstacles)
reliability_and_stability = evaluate_reliability_and_stability(rrt_tree)
degenerate_cases_evaluation = evaluate_handling_degenerate_cases(rrt_tree,
optimal_path)
# Вывод параметров
print(f"Час виконання: {execution_time:.6f} сек.")
print(f"Довжина отриманого маршруту: {path_length:.6f} умовних одиниць")
print(f"Плавність маршруту (Кількість поворотів): {turns_count}")
print(f"Складність навколишнього середовища: {environment_complexity}")
print(f"Загальна надійність та стабільність: {reliability_and_stability}")
print(f"Оцінка вирішення проблем з виродженими випадками:
{degenerate_cases_evaluation}")
```

Додаток Ж

Код реалізацій побудови оптимального маршруту між початковою та кінцевою точками з огляду на перешкоди на карті

```
import numpy as np
import matplotlib.pyplot as plt
from heapq import heappop, heappush

# Клас для комірки карти
class Cell:
    def __init__(self, x, y, obstacle=False):
        self.x = x
        self.y = y
        self.obstacle = obstacle
        self.parent = None
        self.g = float('inf')
        self.h = float('inf')
        self.f = float('inf')
    # Переопределение оператора сравнения "<" для сравнения объектов Cell
    def __lt__(self, other):
        return self.f < other.f
    # Перевизначення оператора порівняння "==" для порівняння об'єктів Cell
    def __eq__(self, other):
        return self.x == other.x and self.y == other.y
    # Перевизначення хеш-функції для використання об'єктів Cell як ключів у словниках та множинах
    def __hash__(self):
        return hash((self.x, self.y))
# Решта програми залишається без змін
```

```
# Функція для обчислення евристичної відстані (для A*)
def heuristic(cell, goal):
    return abs(cell.x - goal.x) + abs(cell.y - goal.y)
# Функція для побудови оптимального маршруту за відстанню
def build_path(goal):
    path = []
    current = goal
    while current is not None:
        path.append((current.x, current.y))
        current = current.parent
    return path[::-1]
# Функція пошуку оптимального маршруту з використанням A*
def find_path(start, goal, grid):
    open_list = []
    closed_set = set()
    start.g = 0
    start.h = heuristic(start, goal)
    start.f = start.g + start.h
    heappush(open_list, start)
    while open_list:
        current = heappop(open_list)
        if current == goal:
            return build_path(goal)
        closed_set.add(current)
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                if dx == 0 and dy == 0:
                    continue
                new_x, new_y = current.x + dx, current.y + dy
                if 0 <= new_x < len(grid) and 0 <= new_y < len(grid[0]):
```

```
                neighbor = grid[new_x][new_y]
                if neighbor.obstacle or neighbor in closed_set:
                    continue
                tentative_g = current.g + 1
                if tentative_g < neighbor.g:
                    neighbor.parent = current
                    neighbor.g = tentative_g
                    neighbor.h = heuristic(neighbor, goal)
                    neighbor.f = neighbor.g + neighbor.h
                    heappush(open_list, neighbor)
# Генерація карти місцевості
def generate_map(width, height, obstacles):
    grid = [[Cell(x, y, (x, y) in obstacles) for y in range(height)] for x in
range(width)]
    return grid
# Візуалізація карти місцевості
def visualize_map(grid, start, goal, path):
    fig, ax = plt.subplots()
    for row in grid:
        for cell in row:
            color = 'black' if cell.obstacle else 'white'
            ax.add_patch(plt.Rectangle((cell.x, cell.y), 1, 1, color=color))
    ax.plot(start[0], start[1], 'go', markersize=10)
    ax.plot(goal[0], goal[1], 'ro', markersize=10)
    if path:
        path_x, path_y = zip(*path)
        ax.plot(path_x, path_y, 'b-', linewidth=2)
    ax.set_aspect('equal', 'box')
    ax.grid(True)
    plt.show()
```

```
# Задаємо параметри карти місцевості
width = 10
height = 10
start = (1, 1)
goal = (8, 8)
obstacles = [(3, 3), (4, 4), (5, 5), (6, 6)]
# Генеруємо карту місцевості
grid = generate_map(width, height, obstacles)
# Знаходимо оптимальний маршрут
path = find_path(grid[start[0]][start[1]], grid[goal[0]][goal[1]], grid)
# Візуалізуємо карту місцевості з маршрутом
visualize_map(grid, start, goal, path)
```

Додаток З

Код реалізацій побудови оптимального маршруту з використанням хвильового алгоритму (зліва направо)

```
import numpy as np
import matplotlib.pyplot as plt

# Функція для візуалізації карти місцевості з перешкодами та результатами розрахунку хвильового алгоритму
def visualize_map(grid, distances, start, goal, path):
    plt.imshow(grid, cmap='binary', origin='lower')  # Білий фон, чорні перешкоди
    plt.imshow(distances, cmap='gray', origin='lower', alpha=0.5, vmin=0, vmax=np.max(distances))  # Результати хвильового алгоритму
    plt.colorbar(label='Distance')  # Шкала значень відстаней
    # Анотація значень у клітинах
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            plt.text(j, i, f'{distances[i, j]:.1f}', ha='center', va='center', color='gray')
    plt.plot(start[1], start[0], 'go', markersize=10)  # Початкова точка
    plt.plot(goal[1], goal[0], 'ro', markersize=10)    # Кінцева точка
    if path:
        path_x, path_y = zip(*path)
        plt.plot(path_y, path_x, 'b-', linewidth=2)    # Оптимальний шлях
    plt.title('Map with Path and Wavefront Algorithm')
    plt.xlabel('Columns')
    plt.ylabel('Rows')
    plt.show()
# Функція створення карти місцевості з перешкодами
def create_map(rows, cols, obstacles):
```

```
    grid = np.zeros((rows, cols), dtype=int)  # Створюємо картку із чорним фоном
    for obstacle in obstacles:
        grid[obstacle[0], obstacle[1]] = 1  # Перешкоди робимо білими
    return grid
# Функція для побудови оптимального шляху
def build_path(distances, start, goal):
    path = [goal]
    current = goal
    while current != start:
        neighbors = get_neighbors(current)
        for neighbor in neighbors:
            if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:
                current = neighbor
                path.append(current)
                break
    return path[::-1]
# Функція для одержання сусідніх комірок
def get_neighbors(cell):
    neighbors = []
    row, col = cell
    for i in range(-1, 2):
        for j in range(-1, 2):
            if i == 0 and j == 0:
                continue
            new_row, new_col = row + i, col + j
            if 0 <= new_row < rows and 0 <= new_col < cols:
                neighbors.append((new_row, new_col))
    return neighbors
# Функція виконання хвильового алгоритму
```

```
def wavefront_algorithm(grid, start, goal):
    rows, cols = grid.shape
    distances = np.full((rows, cols), np.inf)
    distances[start[0], start[1]] = 0
    queue = [start]
    while queue:
        current = queue.pop(0)
        neighbors = get_neighbors(current)
        for neighbor in neighbors:
            if grid[neighbor[0], neighbor[1]] == 0 and distances[neighbor[0],
neighbor[1]] == np.inf:
                distances[neighbor[0], neighbor[1]] = distances[current[0], current[1]] +
1
                queue.append(neighbor)
    return distances
# Параметри карти місцевості
rows = 10
cols = 10
start = (0, 0)
goal = (9, 9)
obstacles = [(3, 3), (4, 5), (6, 6), (6, 7), (6, 8), (6, 9)]
# Створення карти місцевості
grid = create_map(rows, cols, obstacles)
# Виконання хвильового алгоритму
distances = wavefront_algorithm(grid, start, goal)
# Побудова оптимального шляху
path = build_path(distances, start, goal)
# Візуалізація карти місцевості з перешкодами та оптимальним шляхом
visualize_map(grid, distances, start, goal, path)
```

Додаток И

Код реалізацій побудови маршруту на базі хвильового алгоритму в динамічному просторі

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import random

# Функція візуалізації карти місцевості
def visualize_map(grid, distances, start, goal, path):
    cmap = ListedColormap(['white', 'black', 'gray'])  # Білий фон, чорні перешкоди, сірі значення хвильового алгоритму
    plt.figure(figsize=(8, 8))
    plt.imshow(grid, cmap=cmap, origin='lower')
    for i in range(grid.shape[0]):
        for j in range(grid.shape[1]):
            plt.text(j, i, f'{distances[i, j]:.0f}', ha='center', va='center', color='black' if grid[i, j] == 1 else 'gray')
    plt.plot(start[1], start[0], 'go', markersize=10)  # Початкова точка
    plt.plot(goal[1], goal[0], 'ro', markersize=10)    # Кінцева точка
    if path:
        path_x, path_y = zip(*path)
        plt.plot(path_y, path_x, 'b-', linewidth=2)    # Оптимальний шлях
    plt.title('Map with obstacles, wavefront algorithm, and path')
    plt.xlabel('Columns')
    plt.ylabel('Rows')
    plt.show()
# Функція виконання хвильового алгоритму
def wavefront_algorithm(grid, start, goal):
```

```
    rows, cols = grid.shape
    distances = np.full((rows, cols), np.inf)
    distances[start[0], start[1]] = 0
    queue = [start]
    while queue:
        current = queue.pop(0)
        neighbors = get_neighbors(current, rows, cols)
        for neighbor in neighbors:
            if grid[neighbor[0], neighbor[1]] == 0 and distances[neighbor[0],
neighbor[1]] == np.inf:
                distances[neighbor[0], neighbor[1]] = distances[current[0], current[1]] + 1
                queue.append(neighbor)
    return distances
# Функція для одержання сусідніх комірок
def get_neighbors(cell, rows, cols):
    neighbors = []
    row, col = cell
    for i in range(-1, 2):
        for j in range(-1, 2):
            if i == 0 and j == 0:
                continue
            new_row, new_col = row + i, col + j
            if 0 <= new_row < rows and 0 <= new_col < cols:
                neighbors.append((new_row, new_col))
    return neighbors
# Параметри карти місцевості
rows = 20
cols = 20
start = (0, 0)
goal = (rows - 1, cols - 1)
```

```
# Створення порожньої карти місцевості
grid = np.zeros((rows, cols), dtype=int)
# Рандомне додавання перешкод на карту місцевості
for _ in range(rows * cols // 5):
    row, col = random.randint(0, rows - 1), random.randint(0, cols - 1)
    grid[row, col] = 1
# Виконання хвильового алгоритму
distances = wavefront_algorithm(grid, start, goal)
# Побудова оптимального шляху
path = [(goal[0], goal[1])]
current = goal
while current != start:
    neighbors = get_neighbors(current, rows, cols)
    for neighbor in neighbors:
        if distances[neighbor[0], neighbor[1]] < distances[current[0], current[1]]:
            current = neighbor
            path.append(current)
            break
path = path[::-1]
# Візуалізація карти місцевості з перешкодами, значеннями хвильового
алгоритму та маршрутом
visualize_map(grid, distances, start, goal, path)
```

Додаток І

Код реалізацій побудови маршруту у 3-вимірному просторі на базі алгоритму PRM

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random

# Клас для представлення точки у 3D просторі
class Point:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
    def __repr__(self):
        return f"({self.x}, {self.y}, {self.z})"
# Функція перевірки перетину сфер (перешкод)
def check_collision(point, obstacles, radius):
    for obstacle in obstacles:
        if np.linalg.norm([point.x - obstacle.x, point.y - obstacle.y, point.z -
obstacle.z]) < radius:
            return True
    return False
# Генерація PRM шляху
def generate_prm_path(start, end, obstacles, num_samples, num_neighbors,
radius):
    nodes = [start, end]
    for _ in range(num_samples):
```

```
        sample = Point(random.uniform(0, 10), random.uniform(0, 10),
random.uniform(0, 10))
        if not check_collision(sample, obstacles, radius):
            nodes.append(sample)
    prm_path = [start]
    for node in nodes:
        if node != start and not check_collision(node, obstacles, radius):
            nearest_neighbors = sorted(nodes, key=lambda x: np.linalg.norm([x.x -
node.x, x.y - node.y, x.z - node.z]))[:num_neighbors]
            prm_path.extend(nearest_neighbors)
    prm_path.append(end)
    return prm_path
# Створення простору
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Початкова та кінцева точки
start = Point(0, 0, 0)
end = Point(10, 10, 10)
# Генерація перешкод
num_obstacles = 20
obstacles = []
radius = 1
for _ in range(num_obstacles):
    obstacle = Point(random.uniform(0, 10), random.uniform(0, 10),
random.uniform(0, 10))
    while check_collision(obstacle, obstacles, 2 * radius):
```

```
        obstacle = Point(random.uniform(0, 10), random.uniform(0, 10),
random.uniform(0, 10))
    obstacles.append(obstacle)
# Генерація PRM шляху
num_samples = 100
num_neighbors = 5
prm_path = generate_prm_path(start, end, obstacles, num_samples,
num_neighbors, radius)
# Малювання простору з перешкодами
for obstacle in obstacles:
    ax.plot([obstacle.x], [obstacle.y], [obstacle.z], color='black', marker='s')
# Відображення початкової та кінцевої точок
ax.scatter(start.x, start.y, start.z, color='g', marker='o')
ax.scatter(end.x, end.y, end.z, color='b', marker='o')
# Відображення маршруту
prm_path_x = [point.x for point in prm_path]
prm_path_y = [point.y for point in prm_path]
prm_path_z = [point.z for point in prm_path]
ax.plot(prm_path_x, prm_path_y, prm_path_z, color='g')
# Показати графік
plt.show()
```

Додаток Ї

Код реалізацій алгоритму D*

```
import random
import matplotlib.pyplot as plt

# Генерація картки з перешкодами
def generate_map(grid_size, num_obstacles):
    obstacles = set()
    for _ in range(num_obstacles):
        obstacle = (random.randint(0, grid_size-1), random.randint(0, grid_size-1))
        obstacles.add(obstacle)
    return obstacles
# Функція для відображення карти з перешкодами та шляхом
def display_map(grid_size, obstacles, path):
    for i in range(grid_size):
        for j in range(grid_size):
            if (i, j) in obstacles:
                plt.scatter(i, j, color='red', s=100, edgecolors='black', marker='s')  #
червоний квадрат для перешкод
            elif (i, j) in path:
                plt.scatter(i, j, color='green', s=100, edgecolors='black')  # зелена точка
для шляху
            else:
                plt.scatter(i, j, color='white', s=100, edgecolors='black')
    plt.xlim(-1, grid_size)
    plt.ylim(-1, grid_size)
    plt.gca().invert_yaxis()
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()
```

```
# Функція для пошуку шляху методом D*
def d_star(start, goal, obstacles):
    grid = {}
    for i in range(grid_size):
        for j in range(grid_size):
            grid[(i, j)] = float('inf')
    grid[start] = 0
    open_set = {start}
    came_from = {}
    while open_set:
        current = min(open_set, key=lambda x: grid[x])
        open_set.remove(current)
        if current == goal:
            break
        neighbors = [(current[0]+1, current[1]), (current[0]-1, current[1]), (current[0], current[1]+1), (current[0], current[1]-1)]
        for neighbor in neighbors:
            if 0 <= neighbor[0] < grid_size and 0 <= neighbor[1] < grid_size and neighbor not in obstacles:
                tentative_g_score = grid[current] + 1
                if tentative_g_score < grid[neighbor]:
                    came_from[neighbor] = current
                    grid[neighbor] = tentative_g_score
                    open_set.add(neighbor)
    # Відновлення шляху
    path = []
    current = goal
    while current != start:
        path.append(current)
        if current not in came_from:  # Перевірка наявності ключа у словнику
```

```
            break
        current = came_from[current]
    path.append(start)
    path.reverse()
    return path
# Розмір сітки та кількість перешкод
grid_size = 20
num_obstacles = 50
# Генерація карти та відображення її
obstacles = generate_map(grid_size, num_obstacles)
# Завдання початкової та кінцевої точок
start = (0, 0)
goal = (grid_size-1, grid_size-1)
# Пошук шляху методом D*
path = d_star(start, goal, obstacles)
# Відображення карти з перешкодами та шляхом
display_map(grid_size, obstacles, path)
```

Додаток Й

Код реалізації програми керування

(фрагмент)

```
// Визначення пінів для моторів
#define MOTOR_A1 12
#define MOTOR_A2 13
#define MOTOR_B1 14
#define MOTOR_B2 15

void setup() {
  Serial.begin(115200);

  // Ініціалізація пінів
  pinMode(MOTOR_A1, OUTPUT);
  pinMode(MOTOR_A2, OUTPUT);
  pinMode(MOTOR_B1, OUTPUT);
  pinMode(MOTOR_B2, OUTPUT);

  // Підключення до Wi-Fi
  WiFi.begin("YOUR_SSID", "YOUR_PASSWORD");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Запуск веб-сервера
  server.begin();
  Serial.println("Сервер запущено. IP:");
  Serial.println(WiFi.localIP());
```

```
}

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

    if (request.indexOf("/forward") != -1) moveForward();
    else if (request.indexOf("/back") != -1) moveBackward();
    else if (request.indexOf("/left") != -1) turnLeft();
    else if (request.indexOf("/right") != -1) turnRight();
    else if (request.indexOf("/stop") != -1) stopMotors();

    client.print("HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n");
    client.print("<h1>Керування ESP32-CAM Роботом</h1>");
    client.print("<a href='/forward'>Вперед</a><br>");
    client.print("<a href='/back'>Назад</a><br>");
    client.print("<a href='/left'>Ліво</a><br>");
    client.print("<a href='/right'>Право</a><br>");
    client.print("<a href='/stop'>Стоп</a><br>");
  }
}

void moveForward() {
  digitalWrite(MOTOR_A1, HIGH); digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, HIGH); digitalWrite(MOTOR_B2, LOW);
}

void moveBackward() {
```

```
  digitalWrite(MOTOR_A1, LOW); digitalWrite(MOTOR_A2, HIGH);
  digitalWrite(MOTOR_B1, LOW); digitalWrite(MOTOR_B2, HIGH);
}

void turnLeft() {
  digitalWrite(MOTOR_A1, LOW); digitalWrite(MOTOR_A2, HIGH);
  digitalWrite(MOTOR_B1, HIGH); digitalWrite(MOTOR_B2, LOW);
}

void turnRight() {
  digitalWrite(MOTOR_A1, HIGH); digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, LOW); digitalWrite(MOTOR_B2, HIGH);
}

void stopMotors() {
  digitalWrite(MOTOR_A1, LOW);
  digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, LOW);
  digitalWrite(MOTOR_B2, LOW);
}
```

Додаток К

Фрагмент коду реалізацій системи керування мобільним роботом на базі ESP32-Cam

```
#include <WiFi.h>
#include "esp_camera.h"

// Налаштування Wi-Fi
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Піни для керування моторами
#define MOTOR_A1 12
#define MOTOR_A2 13
#define MOTOR_B1 14
#define MOTOR_B2 15

// Налаштування камери (AI Thinker)
#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
```

```
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

WiFiServer server(80);

void startCameraServer() {
  server.begin();
  Serial.println("HTTP server started");
}

void setup() {
  Serial.begin(115200);

  pinMode(MOTOR_A1, OUTPUT);
  pinMode(MOTOR_A2, OUTPUT);
  pinMode(MOTOR_B1, OUTPUT);
  pinMode(MOTOR_B2, OUTPUT);

  // Підключення до Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println(WiFi.localIP());
```

```
// Ініціалізація камери
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size = FRAMESIZE_QVGA;
config.jpeg_quality = 10;
config.fb_count = 1;

// Запуск камери
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
```

```
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  startCameraServer();
}

void loop() {
  WiFiClient client = server.available();
  if (client) {
    Serial.println("Client connected");
    String currentLine = "";

    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        currentLine += c;

        if (c == '\n') {
          if (currentLine.indexOf("GET /forward") >= 0) {
            moveForward();
          } else if (currentLine.indexOf("GET /back") >= 0) {
            moveBackward();
          } else if (currentLine.indexOf("GET /left") >= 0) {
            turnLeft();
          } else if (currentLine.indexOf("GET /right") >= 0) {
            turnRight();
          } else if (currentLine.indexOf("GET /stop") >= 0) {
            stopMotors();
          }
```

```
          // Відповідь HTML
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println();
          client.println("<html><body>");
          client.println("<h1>ESP32-CAM Robot</h1>");
          client.println("<a href=\"/forward\">Вперед</a><br>");
          client.println("<a href=\"/back\">Назад</a><br>");
          client.println("<a href=\"/left\">Ліво</a><br>");
          client.println("<a href=\"/right\">Право</a><br>");
          client.println("<a href=\"/stop\">Стоп</a><br>");
          client.println("</body></html>");
          break;
        }
      }
    }
    client.stop();
    Serial.println("Client disconnected");
  }
}

void moveForward() {
  digitalWrite(MOTOR_A1, HIGH);
  digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, HIGH);
  digitalWrite(MOTOR_B2, LOW);
}

void moveBackward() {
```

```
  digitalWrite(MOTOR_A1, LOW);
  digitalWrite(MOTOR_A2, HIGH);
  digitalWrite(MOTOR_B1, LOW);
  digitalWrite(MOTOR_B2, HIGH);
}

void turnLeft() {
  digitalWrite(MOTOR_A1, LOW);
  digitalWrite(MOTOR_A2, HIGH);
  digitalWrite(MOTOR_B1, HIGH);
  digitalWrite(MOTOR_B2, LOW);
}

void turnRight() {
  digitalWrite(MOTOR_A1, HIGH);
  digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, LOW);
  digitalWrite(MOTOR_B2, HIGH);
}

void stopMotors() {
  digitalWrite(MOTOR_A1, LOW);
  digitalWrite(MOTOR_A2, LOW);
  digitalWrite(MOTOR_B1, LOW);
  digitalWrite(MOTOR_B2, LOW);
}
```

Додаток Л

Код програмної реалізації зображень побудованого на виявленні змін яскравості на зображенні

```
import cv2
import numpy as np

# Завантаження двох зображень (до та після змін)
image1 = cv2.imread('before.jpg')  # перше зображення
image2 = cv2.imread('after.jpg')   # друге зображення

# Переведення в відтінки сірого
gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

# Визначення різниці яскравості
diff = cv2.absdiff(gray1, gray2)

# Порогове значення для виявлення змін
_, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)

# Відображення результатів
cv2.imshow("Зміни яскравості", thresh)
cv2.imshow("Різниця", diff)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Додаток М

Код програмної реалізації алгоритму A*

```
import matplotlib.pyplot as plt
import heapq
import numpy as np

# Рух: вгору, вниз, ліво, право
moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]

def heuristic(a, b):
    # Евристика - манхеттенська відстань
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_set = []
    heapq.heappush(open_set, (0 + heuristic(start, goal), 0, start))
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current_cost, current = heapq.heappop(open_set)

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
```

```
        for move in moves:
            neighbor = (current[0] + move[0], current[1] + move[1])
            if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols:
                if grid[neighbor[0]][neighbor[1]] == 1:
                    continue  # перешкода
                tentative_g_score = g_score[current] + 1
                if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                    g_score[neighbor] = tentative_g_score
                    priority = tentative_g_score + heuristic(neighbor, goal)
                    heapq.heappush(open_set, (priority, tentative_g_score, neighbor))
                    came_from[neighbor] = current
    return None  # шлях не знайдено

def draw(grid, path=None, start=None, goal=None):
    grid = np.array(grid)
    plt.imshow(grid, cmap='gray_r')
    if path:
        px, py = zip(*path)
        plt.plot(py, px, color='blue', linewidth=2, label="Path")
    if start:
        plt.plot(start[1], start[0], "go", label="Start")
    if goal:
        plt.plot(goal[1], goal[0], "ro", label="Goal")
    plt.legend()
    plt.grid(True)
    plt.show()

# Карта: 0 - вільно, 1 - перешкода
grid = [
```

```
    [0, 0, 0, 0, 0, 0],
    [0, 1, 1, 0, 1, 0],
    [0, 1, 0, 0, 1, 0],
    [0, 0, 0, 1, 0, 0],
    [1, 1, 0, 0, 0, 1],
    [0, 0, 0, 1, 0, 0]
]

start = (0, 0)
goal = (5, 5)

path = astar(grid, start, goal)
if path:
    print("Path found:")
    print(path)
else:
    print("No path found.")

draw(grid, path, start, goal)
```

НАУКОВЕ ВИДАННЯ

І.Ш. Невлюдов, В.В. Євсєєв, Д.В. Гурін

ТЕХНІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ РОЗРОБКИ МАЛОГАБАРИТНОГО МОБІЛЬНОГО РОБОТА

[Монографія]

Рекомендовано Вченою радою
Харківського національного університету радіоелектроніки
(протокол № 4 від 22.05.2025 року)

Комп'ютерна верстка Н.К. Ляшова

Підп. до друку 02.06.2025 р Формат 60х84 1/16.
Папір офсетний. Друк ротаційний та цифровий лазерний.
Тираж 300 прим.Зам № 598
Видано на замовлення ХНУРЕ

Ігор Шакирович Невлюдов

Завідувач кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки, доктор технічних наук, професор, Заслужений діяч науки і техніки України, Лауреат Державної премії в галузі науки і техніки України; Лауреат Державної премії України в галузі освіти; член вченої ради, член Президії НМР, голова секції 4 НМР, член Президії НТР, заступник голови секції 3 НТР, член редакційної колегії журналу «Сучасний стан наукових досліджень і технологій в промисловості», член Асоціації випускників ХНУРЕ

Владислав В'ячеславович Євсєєв

Доктор технічних наук, професор, професор кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки, учений секретар спеціалізованої вченої ради, член Асоціації випускників ХНУРЕ

Дмитро Валерійович Гурін

Старший викладач кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки

Монографія "Технічне та програмне забезпечення розробки малогабаритного мобільного робота" є комплексним дослідженням, що вивчає різні напрямки розробки технічного та програмного забезпечення для малогабаритних мобільних роботів. Монографія охоплює такі теми, як класифікація мобільних роботів, проектування концепції та структурної схеми робота, вибір апаратних модулів, розрахунок витрат енергії для керування рухом робота, розробка системи керування з вибором мови програмування, розробка алгоритмів керування та програмної реалізації, а також методи і алгоритми обробки зображень та побудови маршрутів руху. Монографія містить важливі теоретичні роздуми та практичні висновки, що робить її цінним джерелом знань для вчених, інженерів та здобувачів, що цікавляться робототехнікою.

У першому розділі розглядається розробка самого робота, включаючи класифікацію мобільних роботів, проектування концепції, структурну схему, вибір апаратних модулів, розрахунок витрат енергії та розробку 3D моделей.

Другий розділ присвячений розробці системи керування роботом, включаючи вибір мови програмування, розробку алгоритмів керування, програмну реалізацію, налаштування та прошивку плати керування та створення HMI інтерфейсу.

Третій розділ концентрується на методах та алгоритмах обробки зображень для виявлення та відстеження об'єктів у реальному часі. Він описує методи аналізу зображень, алгоритми Canny, активних контурів, оптичного потоку, алгоритм Грейгема, розпізнавання та відстеження об'єктів, алгоритм Sobel та метод Лукаса-Канаде.

У четвертому розділі розглядається побудова маршрутів руху робота, включаючи алгоритми находження короткого маршруту A*, BRRT та A*, побудову оптимального маршруту з урахуванням перешкод, побудову маршруту на базі волнового алгоритму, побудову 3D маршруту на базі алгоритму PRM та алгоритм D*.

Монографія може бути корисна науковцям, аспірантам, здобувачам, які займаються питаннями розробки інтелектуальних робототехнічних систем та фахівцям в галузі знань 17 - Електроніка, автоматизація та електронні комунікації за спеціальністю 174 - Автоматизація, комп'ютерно-інтегровані технології та робототехніка та освітньо-професійних програм: «Робототехніка та кіберфізичні системи», «Автоматизація та комп'ютерно-інтегровані технології».