НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ
імені адмірала Макарова

Навчально-науковий інститут автоматики та електротехніки
(повне найменування інституту, факультету)

Кафедра комп'ютеризованих систем управління
(повна назва кафедри)

«Допущений до захисту»
Завідувач кафедри
_____ Черно О.О.
«_18__» _____12_____ 2025 р.

Пояснювальна записка

до кваліфікаційної роботи

на здобуття другого (магістерського) рівня вищої освіти

за спеціальністю 174 − «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»
ОПП − «Комп'ютеризовані системи управління та автоматика»

на тему: Система комп'ютерного зору робототехнічного комплексу

Виконав студент **6** курсу, **6341м** групи
Сіриченко С.О.
(прізвище та ініціали)
Керівник **Герасін О. С.**
(прізвище та ініціали)

Миколаїв – 2025

Національний університет кораблебудування імені адмірала Макарова

НН Інститут автоматики і електротехніки
Кафедра комп'ютеризованих систем управління
Освітньо–кваліфікаційний рівень **«магістр»**
Спеціальність **174 «Автоматизація, комп'ютерно–інтегровані технології та робототехніка»**
Освітня програма **«Комп'ютеризовані системи управління та автоматика»**

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
_____ **Черно О.О.**
«___» _____ 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНА РОБОТУ СТУДЕНТУ

С і р и ч е н к у С т а н і с л а в у О л е г о в и ч у
(прізвище, ім'я, по батькові)

1. Тема роботи: *Система комп'ютерного зору робототехнічного комплексу.*

керівник роботи *Герасін Олександр Сергійович, к.т.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "14" жовтня 2025 р. № 1217-уч

2. Строк подання роботи *08.12.2025*

3. Вихідні дані до роботи *забезпечення можливості стабільно приймати та обробляти відеопотік з камери у реальному часі, підтримки передачі керуючих команд через UDP-протокол, а також реалізація режиму виявлення та відстеження об'єктів. Система має забезпечувати стабільну роботу на рівні не менше 10 кадрів на секунду.*

4. Мета дослідження
полягає в розробці системи комп'ютерного зору в інтеграції з робототехнічним комплексом, що забезпечує автономний рух мобільної платформи в залежності від поставленої задачі, що підвищує рівень автоматизації керування

5. Зміст розрахунково-пояснювальної записки (основні задачі дослідження)
Аналіз систем комп'ютерного зору в робототехніці.
Проєктування системи комп'ютерного зору робототехнічного комплексу.
Реалізація системи комп'ютерного зору робототехнічного комплексу.
Охорона праці та навколишнього середовища

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Актуальність, мета та задачі кваліфікаційної роботи.

Розробка архітектури системи комп'ютерного зору робототехнічного комплексу.

Налаштування плати ESP32-CAM

Реалізація інтерфейсу користувача

Мережеве підключення

Трекери

Рекомендації щодо покращення точності та надійності системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Охорона праці та навколишнього середовища	*Герасін О.С., доцент каф. КСУ*	*25.11.2025*	*01.12.2025*

Дата видачі завдання: «01» 09 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційна проектування	Строк виконання	Примітка
1	Аналіз систем комп'ютерного зору в робототехніці	15.10.2025	*Виконано*
2	Проєктування системи комп'ютерного зору робототехнічного комплексу	31.10.2025	*Виконано*
3	Реалізація системи комп'ютерного зору робототехнічного комплексу	18.11.2025	*Виконано*
4	Охорона праці та навколишнього середовища	01.12.2025	*Виконано*
5	Оформлення пояснювальної записки	04.12.2025	*Виконано*

Студент _____ (підпис) Сіриченко С.О. (прізвище та ініціали)

Керівник роботи _____ (підпис) Герасін О.С. (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена розробці системи комп’ютерного зору робототехнічного комплексу для автоматизації керування рухом мобільної платформи.

Робота має послідовну структуру. У першому розділі проводиться аналіз сфер застосування робототехнічних систем, інтегрованих з технологіями комп’ютерного зору, а також розглядаються сучасні методи та підходи до їх реалізації. У другому розділі формується архітектура розроблюваної системи та здійснюється вибір апаратних і програмних засобів. У третьому розділі виконується налаштування апаратної частини комплексу, розробка програмного забезпечення для керування рухом мобільної платформи, а також тестування системи з подальшим аналізом її ефективності та виявленням можливих шляхів покращення. У четвертому розділі наведено питання охорони праці та впливу робототехнічних систем на навколишнє середовище.

Обсяг кваліфікаційної роботи становить 88 сторінок, 49 рисунків, 2 таблиці та 15 використаних джерел.

Ключові слова: комп’ютерний зір, машинний зір, виявлення та відстеження об’єктів, робототехніка, автономне керування.

ABSTRACT

The master's thesis is dedicated to the development of a computer vision system for a robotic complex aimed at automating the control of a mobile platform’s movement.

The work is organized in a sequential structure. The first chapter analyzes the application areas of robotic systems integrated with computer vision technologies, as well as modern methods and approaches used for their implementation. The second chapter defines the architecture of the developed system and presents the selection of hardware and software tools. The third chapter focuses on configuring the hardware components of the complex, developing the software for controlling the mobile platform, and testing the system followed by an analysis of its performance and identification of potential improvements. The fourth chapter addresses occupational safety considerations and the environmental impact associated with the development and operation of robotic systems.

The total volume of the qualification paper is 88 pages, 49 figures, 2 tables, and 15 referenced sources.

Keywords: computer vision, machine vision, object detection and tracking, robotics, autonomous control.

ЗМІСТ

ВСТУП

Робототехніка є однією з найдинамічніших і найперспективніших галузей сучасної науки й техніки. Уявлення про створення механічних систем, що імітують діяльність людини, згадуються ще в античні часи, і відтоді концепція роботів постійно привертала увагу винахідників, дослідників і науковців, формуючи основу для подальшого розвитку автоматизованих систем.

З розвитком науково-технічного прогресу людство постійно вдосконалювало способи автоматизації процесів, що стали важливим чинником соціального та технологічного розвитку. Робототехніка, як складова галузі автоматизації, спрямована на створення систем, здатних полегшувати людську працю та виконувати завдання, які раніше вважалися недосяжними. Завдяки цьому з’явилися роботи-пилососи, роботи-газонокосарки, роботи-марсоходи та інші пристрої, що поступово інтегруються в повсякденне життя, змінюючи підходи до виконання побутових і професійних процесів.

Одним із напрямів розвитку сучасної робототехніки є біологічно натхнений підхід – створення роботів, що імітують рухи та поведінку живих організмів. Прикладом є розробки компанії Boston Dynamics, чиї роботи здатні пересуватися, відтворюючи людську або тваринну ходу.

Ключовою перевагою робототехніки є здатність систем виконувати завдання в умовах, які є небезпечними або недосяжними для людини. Так, роботи-марсоходи досліджують поверхню інших планет, системи, здатні функціонувати при високих температурах і тиску, застосовуються для вивчення вулканів, а безпілотні розвідувальні системи забезпечують безпеку в складних умовах. У цих сферах роботи підвищують ефективність виконання завдань та зменшують ризики для життя людей.

Метою робототехніки є створення машин, здатних відтворювати функції людини та, за потреби, перевищувати їх продуктивність. Для досягнення цієї мети робототехнічні системи потребують не лише функціональних механізмів, а й сенсорних модулів, аналогічних до органів чуття людини. Сучасні технології дозволяють робототехнічним системам сприймати та аналізувати

звуки, розрізняти їх за частотними характеристиками, відчувати дотики та оцінювати їхню інтенсивність, а також спостерігати за об’єктами та аналізувати їхні властивості. Розробляються також системи, здатні моделювати сприйняття запахів і смаків. Усі ці функції реалізуються через інтегровані електронні сенсорні системи, спрямовані на автоматизацію людської діяльності та підвищення ефективності технічних пристроїв.

Серед електронних сенсорів найбільш поширеною та практичною є технологія комп’ютерного зору. Комп’ютерний зір застосовується для аналізу зображень, ідентифікації та класифікації об’єктів, а також для виконання інших завдань обробки візуальної інформації. У промисловості ця технологія дозволяє виявляти дефекти на виробах, а в медицині діагностувати злоякісні пухлини на ранніх стадіях.

У повсякденному житті технології комп’ютерного зору знайшли широке застосування: від смартфонів, де вони використовуються для покращення якості фотографій, автоматичного перекладу тексту та пошуку зображень, до складних систем автономного керування транспортом, які здійснюють аналіз оточення в реальному часі.

Для людини зір є ключовим органом сприйняття, через який надходить основна частина інформації про навколишній світ. Це робить задачу передачі цієї здатності робототехнічним системам одним із ключових напрямів сучасних досліджень у науці та техніці.

РОЗДІЛ 1
АНАЛІЗ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ В РОБОТОТЕХНІЦІ

1.1 Сфери застосування та загальні принципи побудови систем комп'ютерного зору

Сфера розробки систем на базі штучного інтелекту (ШІ) розглядається як один із ключових чинників технологічних змін. Це міждисциплінарна галузь, у межах якої створюються інтелектуальні системи, які виконують завдання, раніше доступні лише людині. До таких завдань належать машинне навчання (аналіз даних та прийняття рішень на їх основі), обробка природної мови (розпізнавання текстів та мовлення), робототехніка, експертні системи (використання баз знань і правил для формування рішень або рекомендацій) та комп'ютерний зір.

Комп'ютерний зір є окремим напрямком штучного інтелекту, що зосереджується на створенні автономних систем, здатних відтворювати функції людського зору. Він охоплює збір, обробку та інтерпретацію візуальної інформації з різних джерел (цифрових зображень, відео, камер або систем відеоспостереження) для формування обґрунтованих висновків та прийняття рішень на їх основі.

Упродовж останніх років галузь комп'ютерного зору демонструє динамічний розвиток, і прогнози свідчать, що цей тренд лише посилюватиметься. Популярність технологій комп'ютерного зору пояснюється їхньою універсальністю: вони здатні виконувати широкий спектр завдань – від виявлення та класифікації об'єктів, відстеження руху, розпізнавання дій та оцінювання пози людини, до семантичної сегментації (semantic segmentation) та екземплярної сегментації (instance segmentation), оптичного розпізнавання символів (OCR – Optical Character Recognition), ідентифікації облич, а також реконструкції тривимірних сцен. [1]

Сьогодні комп’ютерний зір знаходить застосування у великій кількості реальних та повсякденних завдань. Одним із перших напрямів його використання є розпізнавання символів, що дозволяє зчитувати рукописні поштові коди на листах або автоматично розпізнавати номерні знаки автомобілів. У промисловості комп’ютерний зір застосовується для оперативного контролю деталей та забезпечення їх якості. За допомогою стереокамер і спеціального освітлення можна вимірювати допуски на деталях літаків та автомобілів, а також виявляти дефекти у литих сталевих виробах. У сфері торгівлі технології комп’ютерного зору дозволяють автоматично ідентифікувати товари на касах самообслуговування.

Ще одним напрямом є створення тривимірних моделей (фотограмметрія), коли автоматизоване відтворення 3D-моделей об’єктів здійснюється на основі знімків предметів, сцен, фотографій зі супутників та літаків. Це корисно у системах картографування, при фіксації сцени злочину, а також для відтворення 3D-об’єктів у комп’ютерних іграх та дизайні інтер’єрів.

У медицині комп’ютерний зір сприяє підвищенню точності діагностики та виявленню захворювань на основі отриманих знімків, зокрема рентгенівських, комп’ютерної томографії (КТ) та магнітно-резонансної томографії (МРТ).

У сфері безпеки та транспорту комп’ютерний зір дозволяє виявляти перешкоди, зокрема пішоходів на дорозі. Технології візуального відстеження також застосовуються у кінематографі для інтеграції комп’ютерної графіки з реальним відео, оцінки руху камери та побудови просторових моделей оточення. Крім того, комп’ютерний зір використовується для захоплення руху (motion capture) за допомогою світловідбиваючих маркерів і камер при створенні анімацій персонажів, у системах спостереження для моніторингу порушників або дорожнього руху, а також у технологіях безпеки, наприклад для виявлення потопаючих людей у басейнах. Технології біометричної ідентифікації, зокрема розпізнавання відбитків пальців, застосовуються як для автоматичної аутентифікації доступу, так і у судово-експертній практиці.

У повсякденному житті комп’ютерний зір також активно застосовується. До прикладів належать об’єднання фотографій у панорами та злиття знімків з різною експозицією для отримання оптимального освітлення зображення. Крім того, можуть застосовуватись методи морфінгу, що дозволяють плавно трансформувати одне зображення в інше. Алгоритми розпізнавання облич та об’єктів сприяють покращенню фокусування камери, виконанню пошуку за зображенням і автоматичному сортуванню фотографій за тематикою.

Комп’ютерний зір має також ключове значення для розвитку робототехніки, оскільки дозволяє роботам сприймати та аналізувати навколишнє середовище, розпізнавати об’єкти, оцінювати їх положення та форму, а також приймати рішення на основі отриманої інформації. Завдяки цьому роботизовані системи можуть виконувати більш складні та точні завдання, адаптуватися до змін середовища та взаємодіяти з оточенням без втручання людини. Такий функціонал значно розширює можливості роботів, сприяє автоматизації рутинних процесів і відкриває нові напрями розвитку робототехніки в промисловості, медицині, логістиці та побуті.

У промисловій автоматизації зорові системи дозволяють роботам точно ідентифікувати та позиціонувати об’єкти на виробничих лініях, що забезпечує ефективне виконання операцій зі складання, контролю якості та пакування продукції. Використання таких технологій підвищує продуктивність і забезпечує стабільну якість виробництва.

У складській логістиці комп’ютерний зір дає змогу роботам автономно переміщуватися приміщенням, визначати та обробляти вантажі, що сприяє зниженню витрат на працю, зменшенню кількості помилок і підвищенню ефективності операцій.

Медицина також отримує значні переваги від впровадження зорових систем. Сучасні технології дозволяють хірургам виконувати точні операції, зменшуючи ризик травмування пацієнта та скорочуючи час проведення процедур.

Інтелектуальні роботи для побуту та взаємодії з людьми також широко застосовують комп’ютерний зір. Завдяки цьому вони здатні розпізнавати предмети в приміщенні, виконувати прибирання, ідентифікувати користувачів за обличчям та надавати персоналізовані послуги. Крім того, роботизовані системи можуть виконувати сортування сміття, автоматично визначаючи типи матеріалів.

Вкрай актуальною залишається сфера застосування роботизованих систем із комп’ютерним зором у військовому контексті, зокрема у поєднанні з безпілотними літальними апаратами. Машинний зір підвищує ситуаційну обізнаність, забезпечує виконання завдань автономної навігації та полегшує реалізацію складних операцій у динамічних і небезпечних умовах, що робить ці технології перспективними для оборонних і розвідувальних застосувань.

Однією з ключових проблем, яку вирішує комп’ютерний зір, є складна фінальна фаза виконання місії за динамічних умов і ураженості засобами радіоелектронної боротьби. Втрата каналу зв’язку або порушення джерел навігації ускладнюють керування та реалізацію завдань. Машинний зір здатний зменшувати частину цих проблем, підвищуючи автономність платформи та надаючи оператору допоміжну інформацію.

Впровадження сучасних алгоритмів комп’ютерного зору у безпілотні системи впливає й на економічний аспект: вартість платформи зростає залежно від складності вбудованих сенсорів і обчислювальних модулів. Це формує компроміс між доступністю системи та її функціональністю: більш складні рішення підвищують ефективність, але роблять платформи дорожчими у виробництві та обслуговуванні.

Ключові переваги інтеграції машинного зору у безпілотні платформи включають зниження навантаження на оператора, скорочення часу навчання персоналу, підвищення стійкості до часткових відмов сенсорних каналів і поліпшення точності розпізнавання об’єктів у складних сценах. Водночас широке застосування таких технологій у військовій сфері потребує ретельної

оцінки нормативно-правових аспектів, контролю та заходів із зменшення ризиків ненавмисного або зловмисного використання. [2]

Таким чином, машинний зір у поєднанні з безпілотними платформами відкриває значні можливості для підвищення ефективності операцій в оборонній сфері, проте його впровадження вимагає балансу між технологічним прогресом, економічними реаліями та етичними й правовими обмеженнями.

Подібні принципи автономного слідування за об’єктом, що застосовуються у військових безпілотних системах, знаходять використання і в побутових роботах. Так, розробляються «роботизовані валізи», які здатні самостійно пересуватися за власником без ручного керування. Комп’ютерний зір дозволяє роботам визначати положення людини, оцінювати її траєкторію руху та адаптуватися до змін навколишнього середовища, уникати перешкод і підтримувати безпечну відстань. Завдяки цьому такі системи підвищують комфорт користувачів і демонструють потенціал технологій автономного пересування у повсякденному житті.

Інтеграція комп’ютерного зору з технологіями штучного інтелекту дозволяє створювати більш «розумних» роботів, здатних не лише виконувати фізичні дії, а й здійснювати елементарне логічне мислення, взаємодіяти з людьми та приймати рішення на основі аналізу даних. Такі системи демонструють потенціал покращення зручності та ефективності у різних аспектах життя людини, одночасно сприяючи розвитку та інноваціям у сфері взаємодії людини з машиною.

Людське око здатне миттєво та без зусиль сприймати тривимірні структури навколишнього середовища. При спостереженні за об’єктами ми оцінюємо їх об’єм, форму та текстуру, незалежно від змін умов освітлення, а також ефективно відокремлюємо їх від фону. Мозок здатний швидко визначати кількість об’єктів на статичних зображеннях, розрізняти їх між собою, а також розпізнавати обличчя людей та оцінювати емоційний стан на основі виразів обличчя.

Протягом десятиліть психологи та нейробіологи досліджують принципи функціонування зорової системи та механізми сприйняття інформації. Незважаючи на значні наукові досягнення, повне розуміння цих процесів залишається недосяжним, зокрема через специфічні обмеження людського сприйняття, що підтверджується великою кількістю візуальних ілюзій, які демонструють, як наш мозок інтерпретує або спотворює сенсорну інформацію.

Паралельно з дослідженнями людського зору у сфері комп'ютерного зору та штучного інтелекту ведуться роботи над створенням алгоритмів, здатних відтворювати тривимірну форму об'єктів та виділяти їх на зображеннях. Сучасні методи дозволяють із високою точністю аналізувати сцени, виявляти предмети та людей, що значно наближає можливості комп'ютерного бачення до механізмів, властивих людському сприйняттю.

Незважаючи на значні успіхи у розвитку алгоритмів комп'ютерного зору, створення системи, здатної інтерпретувати зображення на рівні людського сприйняття, залишається відкритою задачею. Сучасні методи дозволяють аналізувати сцени та виділяти об'єкти з високою точністю, проте повноцінне розуміння контексту та взаємозв'язків у сцені досі недосяжне.

Комп'ютерний зір залишається складною задачею через багатозначність та обмеженість візуальних даних. Система має відновлювати невідомі параметри сцени (форму об'єктів, матеріали поверхонь, освітлення та просторове розташування) на основі обмеженої інформації. Оскільки одне й те саме зображення може відповідати різним комбінаціям цих параметрів, задача є оберненою й містить численні невизначеності. Для її розв'язання застосовуються комплексні підходи: фізичні моделі освітлення та форми, ймовірнісні методи для оцінки невизначеності, а також алгоритми машинного навчання, натреновані на великих наборах прикладів. Така комбінація методів дозволяє зменшити невизначеність і підвищити точність реконструкції, наближаючи результати комп'ютерного бачення до висновків, зроблених людським сприйняттям. [3]

Починаючи з 1960-х років, із розвитком цифрових технологій постало питання створення систем, здатних виконувати завдання на рівні людини. Проте обмежені обчислювальні ресурси того часу не дозволяли здійснювати обробку великих обсягів даних у реальному часі. Попри це, вже тоді робилися перші спроби автоматичної обробки зображень. Зокрема, у 1964 році в межах космічної програми NASA Ranger застосовувались методи цифрової корекції спотворень на фотографіях, отриманих із супутників [4]. Приблизно в той самий період з'явилися алгоритми для порівняння зображень, підвищення їх якості, усунення розмиття та виділення форм, що суттєво спрощувало подальший аналіз даних людиною. Значний поштовх розвитку цих технологій забезпечувала військова сфера – алгоритми цифрової обробки зображень використовувалися для виявлення бункерів, техніки та інших об'єктів на розвідувальних знімках.

Важкою задачею залишалося автоматичне виявлення тривимірних об'єктів у просторі. У 1965 році було запропоновано перший алгоритм для розв'язання такого типу задач, який базувався на використанні простих каркасних моделей фігур – паралелепіпедів, кубів, трикутних і шестикутних призм – та їх порівнянні з лініями контурів об'єктів на зображенні. Це дозволяло системі визначати, яка з відомих моделей найбільше відповідає спостережуваному зображенню. [5]

Паралельно, із розвитком нейрофізіології та дослідженнями роботи мозку, почали з'являтися перші алгоритми машинного навчання, що швидко знайшли застосування у сфері комп'ютерного зору. Такі підходи дали змогу системам «навчатися» на прикладах і поступово підвищувати точність розпізнавання, переходячи від простих геометричних форм до складніших об'єктів.

Після періоду активного розвитку у 1980-х роках інтерес до комп'ютерного зору тимчасово знизився, але вже наприкінці 1990-х – на початку XXI століття – завдяки зростанню обчислювальних потужностей і появі перших ефективних алгоритмів розпізнавання облич відбулося

відродження цієї галузі. Саме ці досягнення стали основою для подальшого прориву у використанні комп’ютерного зору в практичних застосуваннях.

З цього моменту алгоритми глибокого навчання почали активно впроваджуватись у сфері комп’ютерного зору, що дозволило роботизованим системам навчатися на основі великих масивів даних і формувати багаторівневі представлення інформації. Поява згорткових нейронних мереж (Convolutional Neural Networks, CNN) та інших моделей глибинного навчання забезпечила суттєве підвищення точності розпізнавання об’єктів, відстеження руху та виконання інших складних задач аналізу зображень.

Сьогодні згорткові нейронні мережі є одним із найефективніших і найпоширеніших методів у комп’ютерному зорі. Їхня основна перевага полягає у здатності автоматично виявляти значущі ознаки у вхідних даних без необхідності ручного визначення характеристик людиною. Такий підхід значно підвищує узагальнюючу здатність систем і спрощує процес побудови моделей.

Завдяки цьому CNN широко застосовуються не лише для розпізнавання зображень, але й у суміжних сферах – обробці відеопотоків, розпізнаванні облич, аналізі мовлення та інших завданнях, де необхідна автоматична обробка великих обсягів складно структурованої інформації.

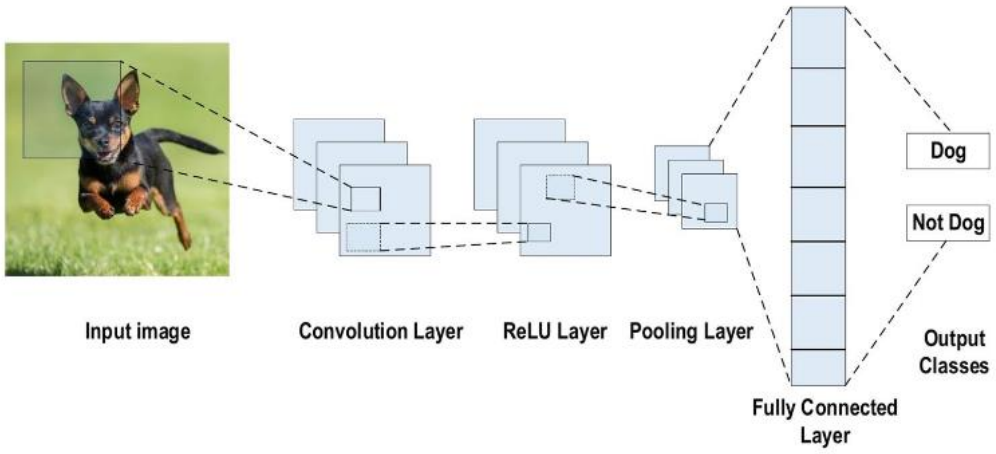

Рис. 1.1. Приклад архітектури згорткової нейронної мережі для класифікації зображення [6]

Однією з ключових переваг згорткових нейронних мереж є спільне використання ваг – повторне застосування одного й того самого набору параметрів (фільтра) для аналізу різних ділянок зображення. На відміну від традиційних нейронних мереж, де кожен нейрон має власні унікальні ваги для кожного вхідного пікселя, CNN використовує один фільтр, який «ковзає» по зображенню (операція згортки) і виявляє повторювані закономірності, такі як краї, кути або текстури, незалежно від їхнього розташування. Таким чином, мережа не запам’ятовує конкретні пікселі, а навчається розпізнавати характерні ознаки, що дозволяє суттєво зменшити кількість параметрів і, відповідно, прискорює навчання, підвищуючи його ефективність.

Типова архітектура CNN має структуру, подібну до багатошарового перцептрона (MLP), однак відрізняється способом обробки просторових даних. Вона зазвичай включає кілька згорткових шарів (convolution layers), що відповідають за виділення ознак, шари підвибірки (sub-sampling або pooling layers), які зменшують розмірність та виділяють найсуттєвіші елементи, а також повнозв’язні шари (fully connected layers, FC), що здійснюють фінальну класифікацію або регресію на основі отриманих ознак (рис. 1.1).

Вхідні дані x кожного шару згорткової нейронної мережі мають тривимірну структуру: висоту, ширину та глибину, тобто $m \times m \times r$, де m – висота та ширина, а r – кількість каналів (глибина). Наприклад, для кольорового RGB-зображення $r = 3$.

Кожен згортковий шар містить кілька ядер (фільтрів), що позначаються через k. Вони також мають три виміри – $n \times n \times q$, подібно до вхідного зображення, однак розмір ядра менший ($n < m$), а кількість каналів q дорівнює або є меншою за r. Ці ядра формують локальні зв’язки та спільно використовують одні й ті самі параметри – ваги W^k та зсув b^k – для створення k карт ознак (feature maps) h^k розміру приблизно $(m - n + 1)$.

У згортковому шарі виконується операція згортки, тобто обчислюється скалярний добуток між вхідним фрагментом вхідного зображення та ядром згортки. Математично цей процес можна подати як:

$$\mathrm{h}^k = f(W^k * x + b^k) \quad (1)$$

де f – функція активації, яка додає нелінійності у модель.

Після операції згортки для кожної карти ознак виконується підвибірка (sub-sampling або pooling), яка зменшує розмірність даних, знижує кількість параметрів і прискорює процес навчання. Така операція також допомагає зменшити ризик перенавчання, роблячи модель стійкішою до незначних зсувів або спотворень у зображенні. Для кожної карти ознак застосовується pooling-функція (наприклад, max або average), що обчислює узагальнене представлення у межах локальної області розміру $p \times p$, де p – розмір вікна.

На заключних етапах повнозв'язні шари (fully connected layers) отримують результати попередніх рівнів – середньо- та низькорівневі ознаки – і на їх основі формують високорівневі абстракції. Ці шари працюють аналогічно класичній нейронній мережі, перетворюючи витягнуті ознаки на вихідні класифікаційні оцінки. Кінцевий шар формує ймовірності належності зображення до певного класу, зазвичай за допомогою функції Softmax або методу опорних векторів (SVM).

Таким чином, архітектура згорткової нейронної мережі складається з послідовності згорткових, підвибіркових та повнозв'язних шарів, які поступово перетворюють вхідне зображення у структуроване представлення, придатне для розпізнавання об'єктів.

Ядро згортки являє собою невелику матрицю (сітку) числових значень, кожне з яких відповідає певній вазі. На початку навчання CNN ці ваги ініціалізуються випадковим чином або за допомогою спеціальних методів ініціалізації. У процесі навчання значення ваг поступово змінюються на кожній ітерації, що дозволяє ядру «навчитися» виділяти найбільш інформативні ознаки зображення – такі як краї, кути чи текстурні переходи.

Вхідні дані для CNN мають вигляд багатоканального зображення. На відміну від традиційних нейронних мереж, які працюють із векторами, згорткові мережі оперують тривимірними масивами пікселів, що описують

висоту, ширину та кількість каналів. Наприклад, зображення у відтінках сірого містить один канал, а RGB-зображення – три.

Для кращого розуміння операції згортки розглянемо приклад: маємо вхідне зображення розміром 4×4 пікселі у відтінках сірого та ядро (фільтр) розміром 2×2 з певними вагами. Ядро послідовно «ковзає» по зображенню в горизонтальному та вертикальному напрямках. У кожній позиції обчислюється скалярний добуток між елементами фрагмента зображення та вагами ядра: відповідні значення перемножуються, а результати підсумовуються. Отримане число формує елемент карти ознак (feature map) (рис. 1.2). Цей процес повторюється, поки ядро не пройде всі допустимі позиції.

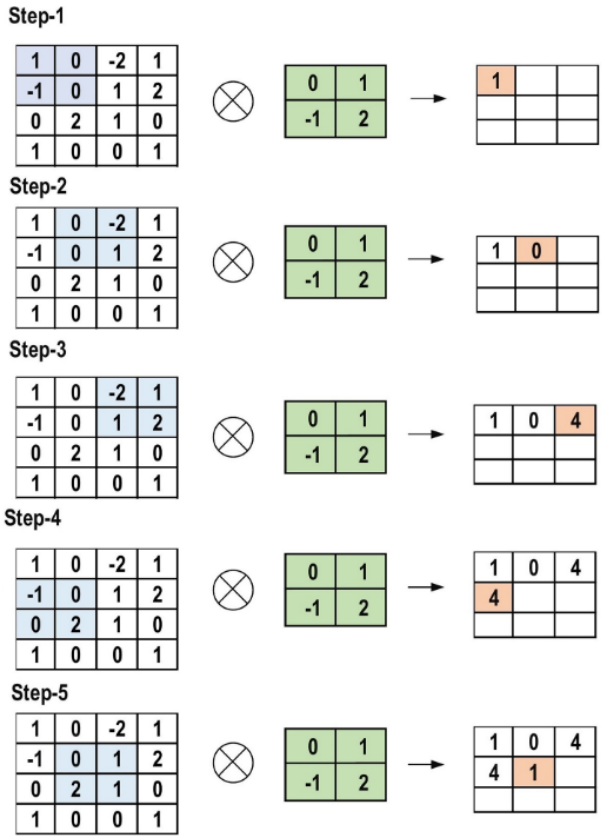

Рис. 1.2. Основні обчислення, що виконуються на кожному кроці згорткового шару [6]

У наведеному прикладі доповнення (padding) не використовується, а крок переміщення (stride) дорівнює 1. Якщо збільшити stride, карта ознак стане меншою, оскільки ядро пропускає частину пікселів. Застосування padding

дозволяє зберегти інформацію з країв зображення – без нього ознаки, розташовані на межах, можуть бути втрачені. Доповнення збільшує розмір вхідного зображення, що відповідно впливає на розмір карти ознак на виході.

Головним завданням шару підвибірки (pooling layer) є зменшення розмірів карт ознак (feature maps), сформованих на попередньому згортковому шарі. Це дозволяє скоротити обчислювальні витрати та зменшити ризик перенавчання, зберігаючи при цьому найважливішу інформацію про ознаки на кожному кроці підвибірки.

Перед виконанням підвибірки визначаються розмір ядра (kernel) та крок переміщення (stride), подібно до операції згортки. Існують різні методи pooling, які можуть застосовуватися на різних шарах, зокрема: max/min pooling (вибір максимальних/мінімальних значень з локальної області), average pooling (обчислення середнього значення), global average pooling (усереднення всіх значень карти ознак, зменшуючи її до одного числа).

Іноді застосування шару підвибірки може зменшувати загальну продуктивність CNN. Це відбувається тому, що підвибірка зосереджується на виявленні наявності ознаки, але частково втрачає інформацію про її точне розташування. В результаті мережа може втрачати частину релевантних деталей, що впливає на точність класифікації чи розпізнавання складних об’єктів.

Головна роль функцій активації в нейронних мережах полягає у відображенні вхідного сигналу на вихідний. Вхідне значення нейрона обчислюється як зважена сума його входів разом із bias (зсувом), якщо він присутній. Функція активації визначає, чи буде нейрон активований для даного входу, і формує відповідний вихід.

Нелінійні функції активації застосовуються після всіх шарів із вагами (тобто шарів, які навчаються, таких як повнозв’язні та згорткові шари) у CNN. Завдяки цьому відображення вхідного сигналу на вихід стає нелінійним, що дозволяє мережі навчатися складнішим закономірностям у даних. Функція активації також повинна бути диференційованою, оскільки це необхідно для

використання алгоритму зворотного поширення помилки (backpropagation) під час навчання мережі. Суть зворотного поширення полягає в обчисленні похідної помилки по відношенню до ваг мережі: диференційована функція активації дозволяє визначити, як зміни у вході нейрона впливають на його вихід, і відповідно коригувати ваги для зменшення помилки.

Найпоширеніші функції активації в CNN та інших глибоких мережах:

- Sigmoid: приймає будь-які дійсні числа на вході та видає значення між 0 і 1.

$$f(x)_{sigm} = \frac{1}{1 + e^{-x}}. \quad (2)$$

- Tanh: подібна до Sigmoid, але вихід обмежується від −1 до 1.

$$f(x)_{tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \quad (3)$$

- ReLU: найпопулярніша функція у CNN. Всі негативні значення на вході перетворюються на 0, а позитивні залишаються без змін. Основна перевага – низька обчислювальна складність.

$$f(x)_{ReLU} = \max(0, x). \quad (4)$$

Повнозв’язний (fully connected, FC) шар зазвичай розташовується в кінці архітектури CNN. У ньому кожен нейрон з’єднаний з усіма нейронами попереднього шару, що і визначає принцип повного з’єднання. FC-шар виконує функцію класифікатора, працюючи за принципом багатошарового перцептрону (MLP), тобто є типом прямої (feed-forward) штучної нейронної мережі.

Вхід у повнозв’язний шар надходить із останнього згорткового або pooling-шару і формується шляхом розгортання (flattening) карт ознак (feature maps) у вектор. Вихід FC-шару представляє кінцевий результат роботи CNN. Остаточна класифікація здійснюється через вихідний шар мережі, де застосовуються функції втрат для оцінки похибки прогнозу на навчальних прикладах.

Функція втрат обчислює різницю між прогнозованим значенням мережі (prediction) та фактичним результатом (label). Вибір конкретної функції втрат залежить від типу завдання: класифікаційного, регресійного або іншого.

Для багатокласової класифікації використовується крос-ентропія (Cross-Entropy або Softmax Loss), яку також називають логарифмічною втратою (log loss). На вихідному шарі застосовується softmax-активація для формування ймовірнісного розподілу класів:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a},$$
$$H(p, y) = -\sum_i y_i \log(p_i), i \in [1, N]. \quad (5)$$

Для задач регресії переважно застосовується євклідова функція втрат (Euclidean Loss), яку ще називають середньоквадратичною помилкою (Mean Square Error):

$$H(p, y) = \frac{1}{2N} \sum_{i=1}^{N} (p_i - y_i)^2. \quad (6)$$

Для бінарної класифікації часто використовують функцію Hinge (Hinge Loss), особливо в методах опорних векторів (SVM). Вона спрямована на максимізацію зазору (margin) між двома класами, де margin зазвичай дорівнює 1, прогнозоване значення позначається p_i, а бажане – y_i:

$$H(p, y) = \sum_{i=1}^{N} \max(0, m - (2y_i - 1)p_i). \quad (7)\ [6]$$

Таким чином, сучасні методи комп’ютерного зору, від класичної обробки зображень до алгоритмів глибинного навчання та CNN, дозволяють створювати роботизовані системи, здатні точно розпізнавати об’єкти, аналізувати сцени та приймати рішення на основі отриманих даних. Комбінація ефективних архітектур, функцій активації та функцій втрат забезпечує поступове навчання мережі та підвищення її точності. Завдяки цьому комп’ютерний зір стає

універсальним інструментом для автоматизації різноманітних задач у промисловості, медицині, побуті та інших сферах життя.

1.2. Апаратні засоби реалізації робототехнічних комплексів з системами комп'ютерного зору.

Сучасні робототехнічні комплекси з системами комп'ютерного зору зазвичай організовані у вигляді трирівневої архітектури (рис. 1.3), що забезпечує чіткий розподіл функцій і взаємодію між компонентами. Такий підхід дозволяє ефективно поєднувати сприйняття середовища, аналітичну обробку даних і керування виконавчими механізмами, що особливо важливо для автономних і мобільних роботів.

Верхній рівень (high-level) відповідає за стратегічне планування дій, прийняття рішень і взаємодію з користувачем або іншими системами. На цьому рівні здійснюється інтерпретація даних, отриманих від сенсорів, формулюються цілі та стратегії роботи робота, визначаються пріоритети завдань і маршрути дій.

Проміжний рівень (interface layer) виконує роль посередника між високорівневим плануванням і фізичними виконавчими механізмами. Він інтегрує сигнали з камер, лідарів, IMU та інших сенсорів, формує модель навколишнього середовища та обчислює траєкторії руху. Цей рівень також відповідає за адаптивне планування та своєчасне реагування на зміни у середовищі.

Нижній рівень (low-level) реалізує безпосереднє керування апаратними компонентами: двигунами, серводвигунами, актуаторами, системами приводу і переміщення. Він забезпечує стабільне виконання команд, коригування положення та уникання перешкод у реальному часі. Завдяки цьому рівню, рішення, що сформовані на верхніх рівнях, втілюються у фізичну дію робота.

Завдяки чіткій ієрархії рівнів, така архітектура забезпечує модульність, масштабованість і гнучкість системи, дозволяючи інтегрувати нові сенсори,

алгоритми обробки даних і типи виконавчих механізмів без повного перепроектування робота. Вона також сприяє зменшенню затримок у прийнятті рішень, оскільки сенсорна інформація та керуючі команди обробляються на відповідному рівні, що підвищує ефективність і автономність робототехнічного комплексу. [7]

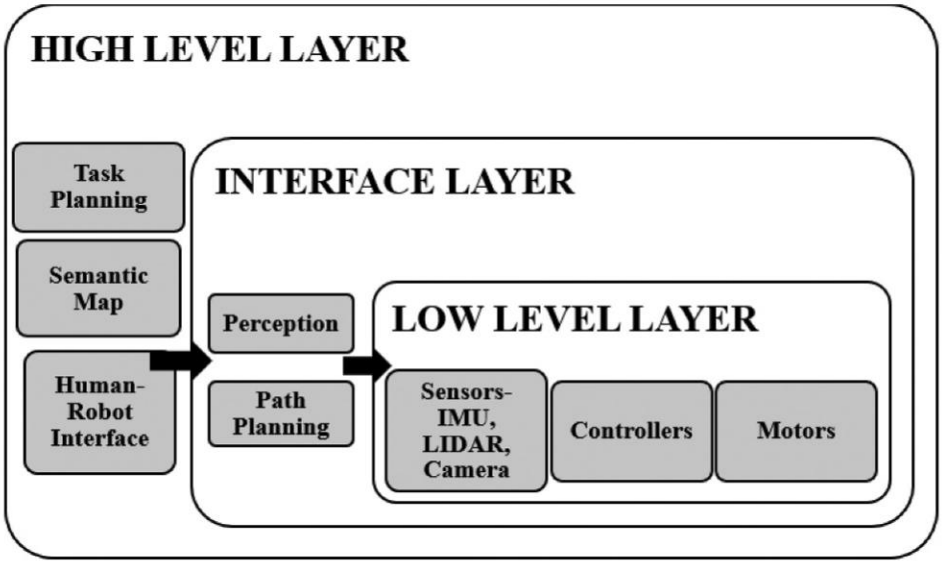

Рис. 1.3. Рівні архітектури автономних роботизованих систем з комп’ютерним зором [7]

Головним органом сприйняття в системах комп’ютерного зору є сенсорна система, яка збирає інформацію про навколишнє середовище для подальшої обробки. Найбільш поширеними є звичайні RGB-камери, що забезпечують кольорове зображення сцени в трьох каналах – червоний, зелений та синій. Такі камери дозволяють аналізувати форму об’єктів, кольори та текстури, що є базою для багатьох алгоритмів комп’ютерного зору.

Для оцінки глибини та тривимірної структури сцени широко використовують стереокамери, що складаються з двох або більше оптичних сенсорів, розташованих на певній відстані один від одного. Порівняння зображень з різних камер дозволяє обчислювати відстань до об’єктів у сцені, що необхідно для навігації мобільних роботів, уникання перешкод та побудови тривимірних моделей оточення (рис. 1.4).

Інфрачервоні камери та тепловізори застосовуються для виявлення об’єктів у темряві або при поганій освітленості, а також для контролю температури поверхні об’єктів (рис. 1.4). Такі сенсори корисні, наприклад, у промислових роботах для перевірки стану обладнання або у системах безпеки.

Камери ToF (Time-of-Flight) дозволяють безпосередньо вимірювати відстань до кожної точки сцени, формуючи глибокі карти та тривимірні моделі об’єктів (рис. 1.4). Спектральні сенсори використовуються для аналізу властивостей матеріалів та освітленості сцени, що може бути важливо, наприклад, для аграрних роботів, які оцінюють стан рослин, або для виробничих ліній, що контролюють якість продукції.

Крім основних камер, до систем комп’ютерного зору часто включають додаткові сенсори, які підвищують точність сприйняття середовища та безпеку роботи робототехнічних комплексів. Одним із найпоширеніших є лідар (Light Detection and Ranging), який вимірює відстань до об’єктів за допомогою лазерного променя (рис. 1.4). Лідари дають змогу будувати точні карти оточення, формувати 3D-моделі сцени та уникати зіткнень, що особливо важливо для мобільних роботів і автономних транспортних засобів.

Ультразвукові сенсори використовують звукові хвилі високої частоти для виявлення перешкод на коротких відстанях. Вони є дешевшими та простішими у використанні, ніж лідари, і добре підходять для базових задач контролю близького оточення, наприклад, під час уникання зіткнень у мобільних робототехнічних системах.

Інерційні вимірювальні блоки (IMU) складаються з акселерометрів, гіроскопів та іноді магнітометрів. Вони дозволяють відстежувати прискорення, кутову швидкість та орієнтацію платформи у просторі. Завдяки цьому робот може підтримувати стабільність руху, виконувати точні маневри та коригувати траєкторію незалежно від зовнішніх факторів.

GPS-модулі забезпечують визначення позиції робота у відкритих просторах із глобальною прив’язкою до координат. Це дає змогу планувати

маршрути, синхронізувати роботу кількох роботів в одному середовищі та інтегрувати систему з картографічними сервісами.

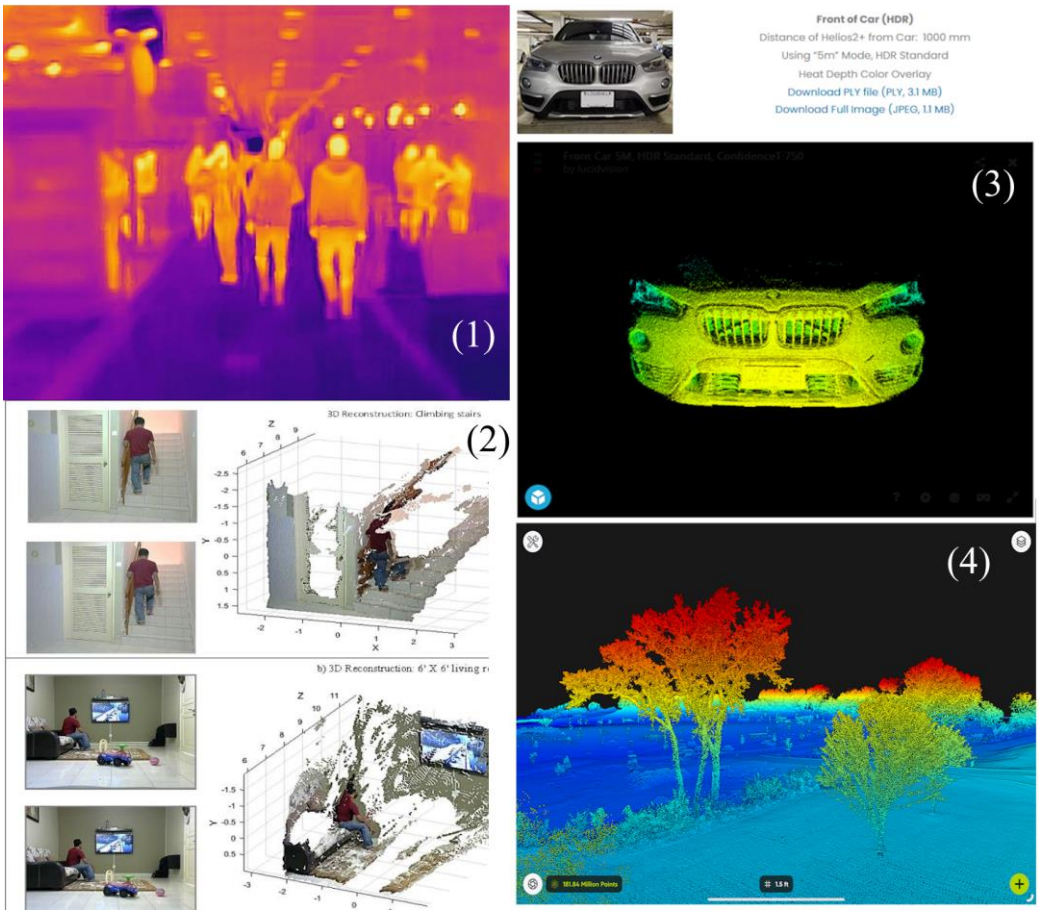

Рис. 1.4. Інфрачервона камера (1) [8], Стерео-камера (2) [9], ToF-камера (3) [10], LiDAR (4) [11]

Крім того, можуть застосовуватися спеціалізовані сенсори для вимірювання додаткових фізичних параметрів – температури, вологості, тиску, концентрації газів та інших характеристик середовища. Такі дані є важливими для промислових роботів, сільськогосподарських платформ і автономних систем моніторингу, де стан навколишнього середовища безпосередньо впливає на ефективність роботи пристрою або якість виконуваних завдань.

Обчислювальні модулі є ключовим компонентом робототехнічних систем із комп’ютерним зором, оскільки вони забезпечують обробку інформації, отриманої від сенсорів. Такі модулі можуть бути інтегровані безпосередньо в конструкцію робота, що дає змогу виконувати локальну обробку даних та

мінімізувати затримку при прийнятті рішень. До вбудованих обчислювальних рішень належать одноплатні комп’ютери, такі як Raspberry Pi або NVIDIA Jetson (Nano, Xavier), а також системи на базі ARM-процесорів із вбудованими графічними прискорювачами. Завдяки цьому роботи здатні виконувати складні алгоритми комп’ютерного зору в реальному часі, обробляти відеопотоки та відстежувати об’єкти без необхідності передавання даних на зовнішні сервери.

У випадках, коли обчислювальні ресурси вбудованої системи є недостатніми або потрібна підвищена продуктивність, обробка даних може здійснюватися на віддалених серверах або за допомогою спеціалізованих прискорювачів. У таких випадках сенсорна інформація передається через мережу, де виконується обчислення, а результати повертаються у вигляді керувальних команд. Для задач глибокого навчання, що вимагають опрацювання великих обсягів даних і виконання паралельних операцій, широко застосовуються графічні процесори (GPU) або спеціалізовані прискорювачі: TPU (Tensor Processing Unit) та NPU (Neural Processing Unit).

GPU відзначаються високим рівнем паралельності обчислень, великою пропускною здатністю пам’яті та ефективністю при роботі з матричними операціями, які є основою згорткових нейронних мереж. Їх архітектура дозволяє виконувати одночасно тисячі обчислень, що суттєво прискорює процес навчання й інференсу моделей. Найпоширенішими середовищами розробки для GPU є CUDA та OpenCL, а також бібліотеки cuDNN, TensorFlow і Caffe, які оптимізують виконання алгоритмів глибокого навчання.

Для компактних робототехнічних систем застосовуються вбудовані модулі, такі як NVIDIA Jetson, що забезпечують виконання моделей безпосередньо на пристрої (edge computing). У масштабних комплексах може використовуватися розподілена обробка або мульти-GPU системи, які дозволяють суттєво скоротити час навчання моделей за рахунок паралельного виконання обчислень на кількох процесорах. [12]

Для обміну даними між компонентами робототехнічного комплексу застосовуються різноманітні інтерфейси, що забезпечують як локальну, так і

віддалену комунікацію. Локальні дротові з’єднання, такі як USB, Ethernet або послідовні інтерфейси (UART, SPI, I²C), забезпечують швидкий та надійний обмін інформацією між обчислювальними модулями та периферійними пристроями, включно з сенсорами та приводами. Такі інтерфейси дозволяють організовувати високошвидкісну передачу даних і забезпечують стабільність роботи системи, що критично для робототехнічних комплексів.

Для віддаленої комунікації використовуються бездротові технології, такі як Wi-Fi, Bluetooth, ZigBee, а також стільникові модулі LTE і 5G. Вони дають можливість передавати дані на віддалені сервери або в хмару в реальному часі, що дозволяє здійснювати моніторинг, обробку та управління робототехнічними системами на відстані. Це особливо важливо для IoT-систем і автономних комплексів, де обробка даних може відбуватися безпосередньо на пристрої або централізовано на сервері. [13]

Завдяки поєднанню дротових і бездротових інтерфейсів сучасні робототехнічні системи можуть одночасно обмінюватися даними з великою кількістю пристроїв, забезпечуючи надійний і масштабований зв’язок між обчислювальними елементами та периферією. Це дозволяє інтегрувати складні системи управління і автоматизації, забезпечуючи ефективну роботу автономних комплексів у реальному часі.

Залежно від типу переданих даних можуть застосовуватись різні мережеві протоколи. Для потокової передачі відео й телеметрії перевага надається UDP, який забезпечує мінімальну затримку при обміні, тоді як TCP використовується для передавання команд і службових даних, де важлива гарантована доставка. У більш складних системах додатково можуть застосовуватись промислові протоколи, наприклад ROS, MQTT або OPC UA, які підтримують одночасну передачу команд і потокових даних між кількома вузлами системи.

Системи приводу та переміщення є кінцевими виконавчими елементами робототехнічних комплексів із комп’ютерним зором, оскільки саме вони реалізують рішення, сформовані обчислювальними модулями. До складу таких систем можуть входити електродвигуни різних типів, серводвигуни, лінійні

актуатори, а також колісні, гусеничні або комбіновані платформи – залежно від функціонального призначення робота.

Конфігурація приводів і систем переміщення визначається специфікою завдань, які виконує робот. Для маніпуляторів пріоритетними є точність і плавність рухів, тому використовуються серводвигуни з високою роздільною здатністю та системами зворотного зв’язку. Для мобільних роботів основним є забезпечення швидкості та прохідності, що досягається за рахунок колісних або гусеничних ходових частин. У деяких конструкціях застосовується незалежне керування окремими приводами, що підвищує маневровість і точність руху.

У випадках, коли робот діє у складному або динамічному середовищі, системи приводу інтегруються з датчиками положення, інерційними вимірювальними блоками та алгоритмами контролю траєкторії. Це дозволяє здійснювати адаптивне керування рухом у реальному часі, своєчасно коригувати положення та уникати перешкод.

Таким чином, апаратні засоби робототехнічних комплексів із системами комп’ютерного зору формують основу їхньої функціональності. Сенсорні системи забезпечують сприйняття навколишнього середовища, обчислювальні модулі здійснюють аналіз і прийняття рішень, а виконавчі механізми реалізують ці рішення у фізичній дії. Узгоджена взаємодія всіх цих компонентів дозволяє створювати автономні, адаптивні та високотехнологічні робототехнічні системи, здатні ефективно працювати у складних і змінних умовах.

1.3. Програмні засоби реалізації алгоритмів обробки зображень і відео

Сучасні системи комп’ютерного зору широко використовують програмні бібліотеки, які реалізують як базові, так і складні алгоритми обробки зображень. Такі бібліотеки доступні для різних мов програмування, зокрема Python, C++, Java та MATLAB. Найпоширенішою мовою нині є Python завдяки

наявності великої кількості фреймворків для глибокого навчання. Використання цих інструментів значно спрощує розробку систем комп’ютерного зору – від етапу підготовки даних і попередньої обробки зображень до реалізації та навчання нейронних мереж, усуваючи потребу створювати базові алгоритми вручну.

Бібліотеки обробки зображень надають широкий набір інструментів для роботи з візуальною інформацією – від простих операцій над пікселями до комплексного аналізу сцени. До базових функцій належать зміна розмірів зображення, обертання, нормалізація, корекція освітленості та перетворення кольорових просторів. Такі операції використовуються на етапі попередньої обробки, забезпечуючи підготовку даних до подальшого навчання нейронних мереж або застосування алгоритмів класичного комп’ютерного зору.

Серед ключових алгоритмів, реалізованих у бібліотеках комп’ютерного зору, варто відзначити фільтрацію та згладжування зображень, які дозволяють зменшити рівень шуму, покращити якість і виділити основні структури. Для цього застосовуються гаусові, середні, медіанні та адаптивні фільтри. Важливим напрямом є також виявлення контурів і градієнтів, що забезпечує визначення меж і форми об’єктів на зображенні. Типовими прикладами таких алгоритмів є Sobel, Canny та Laplacian.

Фреймворки для глибокого навчання є ключовими інструментами у розробці сучасних систем комп’ютерного зору. Вони надають програмні інтерфейси та готові модулі для створення, навчання й тестування нейронних мереж, що дає змогу ефективно обробляти великі масиви даних і виконувати складні обчислення. На відміну від бібліотек обробки зображень, фреймворки не обмежуються окремими функціями чи алгоритмами – вони формують повноцінне середовище для побудови моделей, управління процесом навчання, інтеграції з апаратними прискорювачами (GPU, TPU, NPU) та оптимізації параметрів мережі.

Одними з найважливіших критеріїв відбору бібліотек і фреймворків для обробки зображень є їх функціональність, популярність і підтримка спільноти,

простота використання, продуктивність, сумісність із різними форматами зображень, можливість інтеграції з іншими програмними продуктами та гнучкість для розширення під конкретні задачі.

OpenCV (Open Source Computer Vision Library) є потужною бібліотекою з відкритим вихідним кодом для задач комп'ютерного зору та обробки зображень у реальному часі. Вона надає широкий набір алгоритмів для маніпуляцій із зображеннями, виявлення об'єктів, виокремлення ознак, обробки відео та інтеграції з моделями глибокого навчання, що робить її популярною як у дослідженнях, так і в промислових проєктах.

Pillow (Python Imaging Library, PIL) пропонує прості та зручні інструменти для базових операцій з цифровими зображеннями: обрізання, зміна розміру та збереження у різних форматах. Scikit-Image забезпечує ефективні методи фільтрації, сегментації та виділення ознак, а Mahotas оптимізована для швидкої обробки великих наборів даних і забезпечує базові алгоритми комп'ютерного зору.

TensorFlow та Keras дозволяють створювати та навчати моделі глибокого навчання, інтегруючи їх із класичними методами обробки зображень для виконання складних завдань комп'ютерного зору.

Усі ці бібліотеки та фреймворки можна порівняти за функціональністю, продуктивністю та іншими характеристиками, що дозволяє обрати оптимальний інструмент для конкретних задач (табл. 1). [14]

Моделі комп'ютерного зору визначають структуру та принцип роботи нейронних мереж або алгоритмів, що виконують аналіз і інтерпретацію візуальної інформації. Вони задають послідовність обробки даних від сенсорів – від первинного зчитування пікселів до формування числових ознак, які використовуються для прийняття рішень або формування керувальних сигналів у робототехнічних системах.

Таблиця 1.

Порівняння бібліотек Python для обробки зображень [14]

	OpenCV	**Pillow**	**Scikit-Image**	**TensorFlow**	**Mahotas**
Підтримувані формати	Широкий спектр	Поширені формати	Поширені формати	Поширені формати	Поширені формати
Основні функції	Фільтрація, перетворення, виявлення країв, виявлення об’єктів тощо.	Базові маніпуляції зі зображеннями (обрізання, зміна розміру тощо).	Фільтрація, сегментація, визначення особливостей тощо.	Попередня обробка зображень для машинного навчання.	Швидкі операції обробки зображень.
Додаткові можливості	Інтеграція машинного навчання, обробка в режимі реального часу.	Обмежені.	Науковий аналіз зображень.	Глибоке навчання, аналіз зображень на основі ШІ.	Операції з бінарними зображеннями, виділення ознак.
Простота інтеграції	Помірна (вимагає розуміння концепцій комп’ютерного зору).	Легка (простий API, добре задокументована).	Помірна (добре задокументована, але бажаний науковий досвід).	Помірна (вимагає розуміння концепцій машинного навчання).	Помірна (вимагає розуміння обробки зображень).
Показники ефективності	Висока продуктивність, оптимізована для швидкодії.	Хороша продуктивність для базових завдань.	Хороша продуктивність, але не така швидка як OpenCV.	Висока продуктивність для завдань машинного навчання.	Висока продуктивність для конкретних завдань.
Якість документації	Чудова якість, детальна документація.	Чудова якість, зрозуміла для користувачів.	Чудова якість, широка документація.	Чудова якість, детальна документація.	Хороша сфокусована на конкретних завданнях.

Залежно від завдання, моделі адаптуються під різні типи операцій. Згорткові нейронні мережі (CNN) застосовуються для класифікації об’єктів на зображеннях, U-Net або Mask R-CNN – для сегментації сцен, а YOLO, SSD або

Faster R-CNN – для детекції та визначення положення об’єктів. Алгоритми відстеження руху аналізують послідовності кадрів і формують траєкторії переміщення, що забезпечує навігацію або маніпуляцію робототехнічного комплексу.

Використання готових або попередньо навчених моделей скорочує час розробки, дозволяючи одразу інтегрувати їх у робототехнічну платформу. При цьому можливе до- або перенавчання (fine-tuning) враховує специфіку середовища, типи об’єктів або сценарії роботи, що підвищує точність та стабільність роботи системи у реальних умовах.

На низькому рівні прошивки вбудованих контролерів і плат виконують базову обробку відеопотоків, попереднє розпізнавання об’єктів та формування команд для приводів у реальному часі, що знижує навантаження на центральний обчислювальний модуль і мінімізує затримку реакції.

Крім того, у робототехнічних системах використовуються користувацькі інтерфейси та спеціалізовані програми для моніторингу, налаштування параметрів моделей і запуску сценаріїв. Вони дозволяють змінювати порогові значення для визначення об’єктів, чутливість до освітлення, частоту оновлення кадрів, роздільну здатність відеопотоку або режим фільтрації шумів, оптимізуючи роботу системи під конкретні умови середовища.

Комплексна взаємодія прошивок, алгоритмічних модулів, моделей глибокого навчання та користувацьких інтерфейсів формує єдине програмне середовище, що забезпечує автономний аналіз візуальної інформації та виконання складних навігаційних і технологічних завдань без постійного втручання людини. [15]

Висновок до першого розділу

У першому розділі здійснено огляд сфер застосування та загальних принципів побудови систем комп’ютерного зору. Визначено сутність комп’ютерного зору та коротко розглянуто його історичний розвиток, що

дозволяє оцінити еволюцію технологій і підходів до обробки візуальної інформації.

Детально проаналізовано апаратні засоби реалізації робототехнічних комплексів із системами комп’ютерного зору: сенсорні пристрої, обчислювальні модулі та приводи, які забезпечують збір даних, їх обробку та перетворення рішень у фізичні дії робота.

Також розглянуто програмні засоби, що реалізовують алгоритми обробки зображень і відео, включно з бібліотеками комп’ютерного зору, фреймворками глибокого навчання та готовими моделями для різних задач. Ця інтеграція апаратних і програмних компонентів дозволяє створювати автономні робототехнічні системи, здатні приймати рішення у реальному часі й адаптуватися до динамічних умов навколишнього середовища.

Метою роботи є розробка системи комп’ютерного зору робототехнічного комплексу для забезпечення автономного руху шляхом ідентифікації та відстеження об’єктів.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ РОБОТОТЕХНІЧНОГО КОМПЛЕКСУ

2.1 Постановка задачі та загальна архітектура системи

Розробка націлена на створення Android-додатку, що забезпечуватиме можливість як ручного, так і автономного керування рухом робототехнічної платформи на основі різних методів відстеження об'єктів. Система також передбачатиме функції ідентифікації та класифікації об'єктів, що розширює її функціональність та дозволяє реалізувати інтелектуальну поведінку робота під час автономної навігації.

Відеодані з камери робота передаватимуться до Android-додатку в режимі реального часу за допомогою WebSocket-з'єднання. Це забезпечить низьку затримку та можливість одночасного виконання декількох операцій: відображення відеопотоку, проведення аналізу зображень, а також вмикання та перемикання режимів роботи. Користувач зможе ініціювати ручне керування, запускати алгоритми ідентифікації або активувати режим слідування за об'єктом безпосередньо через інтерфейс додатку.

При взаємодії користувача з інтерфейсом, відповідні команди передаватимуться на робототехнічну платформу через протокол UDP, що забезпечує мінімальну затримку управління. Сигнали надходитимуть на керуючу плату камери, звідки вони будуть передаватися на драйвери приводів, які формуватимуть команди для двигунів. Такий підхід дозволяє реалізувати синхронізовану роботу програмного та апаратного забезпечення, гарантує швидку реакцію платформи на дії користувача та забезпечує високу точність виконання рухових команд (рис. 2.1).

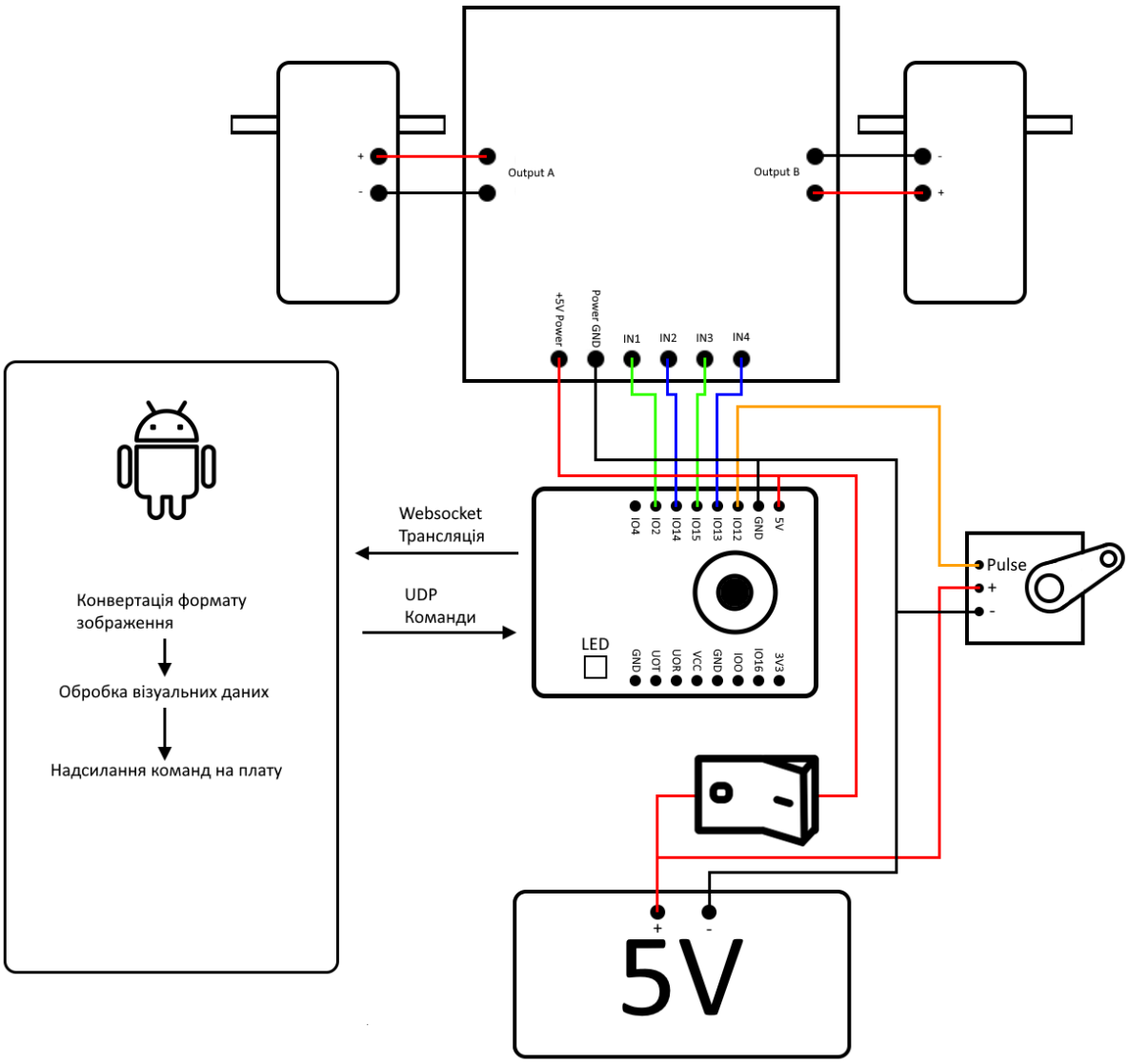

Рис. 2.1. Схема взаємодії елементів робототехнічного комплексу

В Android-додаток буде інтегровано модулі для обробки вхідного відеопотоку, застосування алгоритмів відстеження та розпізнавання, а також механізми прийняття рішень щодо руху робота в автономному режимі.

2.2 Вибір апаратної частини

В якості камери для мобільного робота було обрано монокулярну камеру, оскільки вона є простою у використанні, доступною за вартістю та універсальною для виконання більшості завдань комп'ютерного зору. Для цього застосовується сенсор OV2640, інтегрований у модуль ESP32-CAM,

який широко використовується в робототехнічних проєктах. Камера OV2640 поєднує компактність, низьке енергоспоживання та прийнятну якість зображення. Сенсор забезпечує максимальну роздільну здатність 1600×1200 пікселів (UXGA) та підтримує вивід у форматах JPEG, YUV і RGB, що дозволяє адаптувати його для різних задач, таких як потокова передача відео або попередня обробка зображень. Завдяки вбудованому модулю обробки зображень (ISP) камера забезпечує автоматичну експозицію, баланс білого, шумозниження та корекцію кольорів, що знижує навантаження на основний контролер. Підтримується швидкість до 30 кадрів на секунду у роздільності SVGA, а об'єктив з форматом сенсора 1/4" і розміром пікселя 2,2 мкм забезпечує достатню якість для роботи в умовах нормального освітлення. Популярність сенсора, сумісність з ESP32-CAM та доступність програмних бібліотек роблять його оптимальним вибором для компактного мобільного робота.

Модуль ESP32-CAM, на базі мікроконтролера ESP32-S, є компактним обчислювальним вузлом з інтегрованим Wi-Fi та Bluetooth, що дозволяє організовувати бездротовий зв'язок у робототехнічних системах. Плата має габарити 27×40,5 мм, містить 32-Мбітну SPI-Flash пам'ять, 520 КБ SRAM та 4 МБ PSRAM для обробки даних, а також 9 доступних GPIO і підтримку інтерфейсів UART, SPI, I²C та PWM для підключення датчиків, сервоприводів та іншої периферії. Вбудована PCB-антена з підсиленням 2 dBi у поєднанні з передавальною потужністю до 17 dBm забезпечує стабільний Wi-Fi-зв'язок, а чутливість приймача від −90 до −67 dBm визначає максимальну дальність та надійність підключення. Модуль підтримує безпечне підключення до мереж через протоколи WPA/WPA2/WPA2-Enterprise та WPS, що забезпечує захист переданої інформації.

Для руху мобільної платформи використовується двоканальний H-мостовий драйвер двигунів постійного струму L298N, який дозволяє незалежно керувати двома двигунами у двох напрямках. Драйвер здатний працювати з двигунами до 2 А кожен, має вхідну напругу від 3,2 до 40 В та

вбудований стабілізатор +5 В, який може живити контролер. Для керування двигунами достатньо кількох цифрових виходів мікроконтролера, а сигнали управління відповідають стандартним TTL-рівням. Плата оснащена світлодіодним індикатором живлення та захисними діодами, що підвищує надійність її роботи. Розмір плати складає 3,4×4,3×2,7 см, а максимальне споживання потужності досягає 20 Вт при температурі 75 °C, що робить її зручною для використання в автономних роботах.

В якості сервоприводу для керування положенням камери застосовується Micro Servo SG90. Цей мікросервомотор є компактним і легким (вага 9 г), забезпечує крутний момент 1,2–1,5 кг·см при напрузі 4,8–6 В і може обертатися приблизно на 180° (по 90° в кожну сторону від середнього положення). Час повороту становить близько 0,10–0,12 секунди на 60°, що дозволяє реалізовувати швидкі та точні рухи. Серво має стандартний аналоговий сигнал керування (ширина імпульсу ~1–2 мс, період ~20 мс), сумісне з будь-яким кодом або бібліотекою для серводвигунів і постачається з трьома змінними “рогами” для кріплення механічних важелів. Корпус пристрою компактний (приблизно 23×12×29 мм), шестерні виконані з пластику, а ротор обертається на втулкових підшипниках, що забезпечує плавну роботу. SG90 підходить для використання в невеликих роботах, моделях літаків та гелікоптерів, де важливі низька вага та простота управління.

Для забезпечення руху мобільної платформи будуть використані два двигуни постійного струму з редуктором, що дозволяють досягти необхідної швидкості та крутного моменту для переміщення робота. Кожен мотор працює від напруги 6 В і споживає струм 120 мА у звичайному режимі. Використаний редуктор з передатним числом 48:1 забезпечує зниження швидкості обертання та збільшення крутного моменту, що дозволяє ефективно передавати силу на колеса. Максимальна швидкість обертання двигуна становить 240 об/хв, а при встановлених колесах діаметром 66 мм це забезпечує лінійну швидкість платформи приблизно 48 м/хв. Мотори мають

компактні габарити (70×22×18 мм) і малу вагу (50 г кожен), що робить їх оптимальними для невеликих робототехнічних систем і дозволяє легко інтегрувати їх із драйвером L298N для незалежного керування кожним колесом (рис. 2.2).

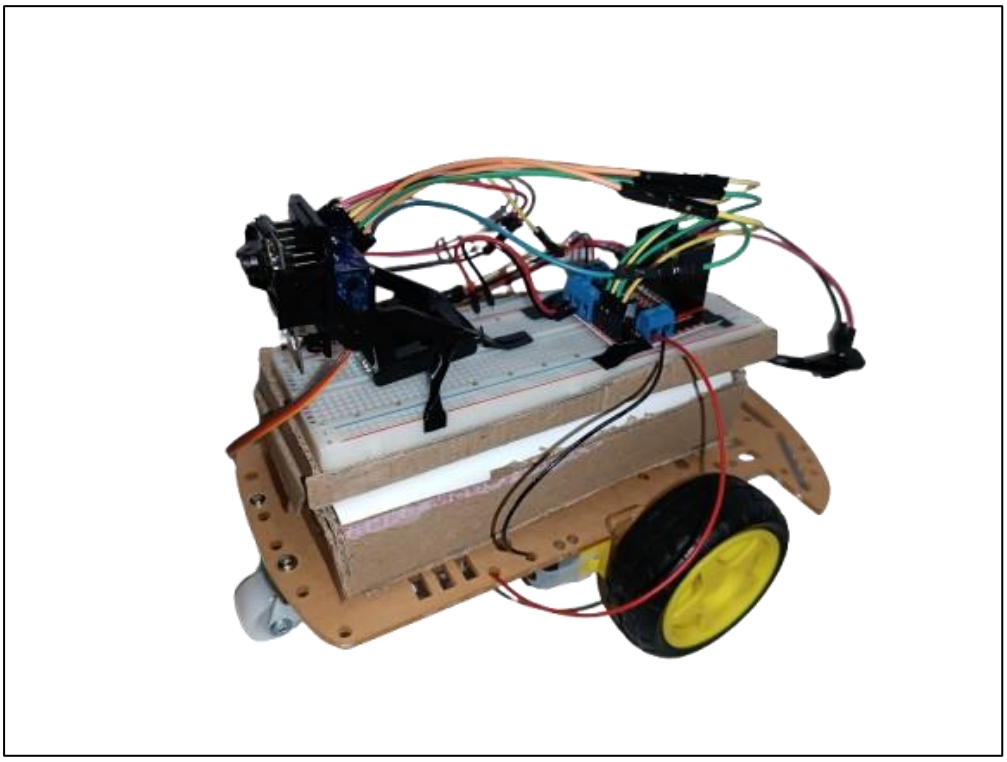

Рис. 2.2. Робототехнічний комплекс

В якості обчислювального пристрою для запуску мобільного додатку буде використано смартфон оснащений 12 Гб оперативної пам’яті та восьмиядерним процесором MediaTek Helio G96 з тактовою частотою 2,05 ГГц. Велика кількість оперативної пам’яті дозволяє одночасно обробляти відеопотік з камери та виконувати алгоритми комп’ютерного зору без затримок. Восьмиядерний процесор забезпечує паралельне виконання обчислень, що підвищує швидкодію додатку і дозволяє здійснювати аналіз кадрів у реальному часі. Смартфон також має достатню обчислювальну потужність для роботи з бібліотеками машинного навчання, детекції об’єктів, трекінгу та інших завдань комп’ютерного зору. Використання смартфона як центрального обчислювального вузла дозволяє поєднати високопродуктивну

обробку даних, інтерфейс для відображення результатів і зручне керування роботом через додаток, що робить систему більш автономною та практичною у використанні.

Для живлення мобільної робоплатформи використовується акумуляторний блок ємністю близько 20000 мА·год, який забезпечує тривалу автономну роботу та стабільні 5 В з вихідним струмом до 3 А.

2.3 Вибір програмної платформи

Прошивка плати буде реалізована на платформі Arduino IDE з використанням мови C++. Вона забезпечує роботу ESP32-CAM як автономного вузла для збору та передачі відеопотоку, а також керування рухом мобільної платформи та сервоприводом камери. Для роботи з мережею використовується бібліотека WebSocketsServer, що дозволяє організувати стабільний обмін даними з мобільним додатком. Прошивка розроблена з урахуванням обробки UDP-команд, керування колесами через PWM та реалізації плавного управління сервоприводом. Для прошивки використовується платформа ESP32 by Espressif Systems версія 2.0.6, сумісна з Arduino IDE та здатна забезпечувати ефективну обробку відеопотоку в режимі реального часу.

Мобільний додаток реалізовано за допомогою Android Studio, основна логіка написана на Java 11, а інтерфейс оформлений за допомогою XML. Цільова версія Android – API 36, мінімальна підтримувана – API 24, що забезпечує сумісність із більшістю сучасних пристроїв. Додаток використовує сторонні бібліотеки OpenCV 3.4.8 для обробки відеопотоку, детекції та трекінгу об’єктів, Java-WebSocket 1.5.6 для обміну даними з платою ESP32-CAM через WebSocket, а також Material Components 1.12.0 для реалізації сучасного інтерфейсу користувача, зокрема Swipe-панелей.

Таке поєднання платформ, мов програмування та бібліотек дозволяє забезпечити ефективне виконання алгоритмів комп’ютерного зору, обробку

відеопотоку та інтерактивне керування роботом з мобільного пристрою в режимі реального часу.

Висновок до другого розділу

Для реалізації системи розробляється Android-додаток, який забезпечує ручне та автономне керування робототехнічною платформою з функціями відстеження, ідентифікації та класифікації об’єктів. Відеопотік з камери передається в реальному часі через WebSocket, а команди управління надходять на робота через UDP, що забезпечує точність і швидку реакцію платформи.

В якості апаратної частини використано камеру OV2640 на модулі ESP32-CAM, двигуни з редуктором, драйвер L298N та сервопривід Micro Servo SG90. Смартфон виступає в ролі центрального обчислювального вузла з оптимальною процесорною потужністю для обробки відео та алгоритмів комп’ютерного зору.

Програмна платформа включає прошивку ESP32-CAM на Arduino IDE (C++) для керування рухом і передачею даних, а додаток на Android використовується для обробки відеопотоку, виявлення та відстеження об’єктів.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ РОБОТОТЕХНІЧНОГО КОМПЛЕКСУ

3.1 Збірка та налаштування апаратної частини

Плата ESP32-CAM отримує живлення від зовнішнього джерела напруги 5 В через пін 5V та GND. Інші пристрої, підключені до плати, також отримують живлення від того ж джерела через розгалужений кабель.

Пін IO4 використовується для керування вбудованим світлодіодом плати. Через цей пін надходять логічні сигнали, що дозволяють вмикати та вимикати світлодіод.

Сигнальні виходи для керування сервоприводом підключені до піна IO12. Через цей пін від плати надходять імпульсні сигнали широтно-імпульсної модуляції (ШІМ), що забезпечують точне позиціонування серводвигуна. Живлення сервопривода здійснюється безпосередньо від джерела.

Для керування колесами використовуються чотири піни плати ESP32-CAM. Ліве колесо керується сигналами з пінів IO15 (рух вперед) та IO13 (рух назад). Праве колесо керується сигналами з пінів IO2 (рух вперед) та IO14 (рух назад). Ці сигнали подаються на відповідні входи драйвера двигунів, який формує необхідну полярність та силу струму для обертання електродвигунів у заданому напрямку.

Таким чином, плата ESP32-CAM здійснює керування світлодіодом, сервоприводом та електродвигунами через призначені GPIO-піни, в той час як безпосереднє живлення виконується від зовнішнього джерела напруги.

Метод setup() використовується для одноразової ініціалізації апаратної та програмної частини роботизованої платформи. Він готує піні, сенсори, сервопривід, камеру та комунікаційні інтерфейси, щоб система могла коректно працювати після запуску.

У наведеному фрагменті коду (рис. 3.1) відбувається апаратна ініціалізація вихідних сигналів роботизованої платформи. Зокрема, визначаються цифрові виходи для управління напрямком обертання коліс, канали ШІМ для регулювання швидкості коліс, а також канал ШІМ для керування кутом нахилу сервоприводу. Крім того, задається початкове положення сервоприводу для встановлення початкового нахилу камери.

```
const int LED_BUILT_IN = 4; // Пін світлодіода
const int PINDC_LEFT_BACK = 13; // Пін лівого колеса назад
const int LEFT_CHANNEL = 2; // ШІМ канал для лівого колеса
const int PINDC_LEFT_FORWARD = 15; // Пін лівого колеса вперед
const int PINDC_RIGHT_BACK = 14; // Пін правого колеса назад
const int RIGHT_CHANNEL = 3; // ШІМ канал для правого колеса
const int PINDC_RIGHT_FORWARD = 2; // Пін правого колеса вперед
const int SERVO_PITCH_CHANNEL = 4; // ШІМ канал для серво нахилу
const int PIN_SERVO_PITCH = 12; // Пін для серво нахилу
int posServo = 75; // Початкове положення серво у градусах
```

Рис. 3.1. Ініціалізація пінів плати ESP32-CAM

У наступному фрагменті коду (рис. 3.2) відбувається конкретне налаштування пінів та каналів ШІМ. Вбудований світлодіод налаштований як вихід і вимкнений на старті. Піни для руху коліс «назад» встановлені як цифрові виходи, а піни «вперед» підключені до відповідних каналів ШІМ для подальшого регулювання швидкості. Початково обертання коліс відключене. Для серводвигуна, який керує нахилом камери, налаштований канал ШІМ із потрібною частотою та роздільною здатністю, і сервопривід встановлюється у початкову позицію.

Функція controlDC (рис. 3.3) забезпечує ручне керування колесами роботизованої платформи, комбінуючи цифрові виходи для визначення напрямку руху та сигнали ШІМ для регулювання швидкості обертання коліс. Функція servoWrite (рис. 3.4) відповідає за точне встановлення кута серводвигуна шляхом перетворення бажаного кута нахилу у відповідне значення широтно-імпульсного сигналу, що подається на канал сервоприводу.

```
// Налаштування піну для світлодіода
pinMode(LED_BUILT_IN, OUTPUT); // Пін буде виходом
digitalWrite(LED_BUILT_IN, LOW); // Пін вимкнуто

// Налаштування пінів та ШІМ для колес
pinMode(PINDC_LEFT_BACK, OUTPUT);
ledcSetup(LEFT_CHANNEL, 100, 8);
ledcAttachPin(PINDC_LEFT_FORWARD, LEFT_CHANNEL);
pinMode(PINDC_RIGHT_BACK, OUTPUT);
ledcSetup(RIGHT_CHANNEL, 100, 8);
ledcAttachPin(PINDC_RIGHT_FORWARD, RIGHT_CHANNEL); // Пін до ШІМ
controlDC(LOW,LOW,LOW,LOW); // Вимикаємо обидва піни колес

// Налаштування пінів та ШІМ для серводвигуна (нахил камери)
ledcSetup(SERVO_PITCH_CHANNEL, 50, 16);
ledcAttachPin(PIN_SERVO_PITCH, SERVO_PITCH_CHANNEL); // Пін серво
servoWrite(SERVO_PITCH_CHANNEL, posServo); // Поч. позиція серво
```

Рис. 3.2. Налаштування пінів

```
void controlDC(int left0, int left1, int right0, int right1){
  digitalWrite(PINDC_LEFT_BACK, left0); // Стан лівого колеса
  if(left1 == HIGH){
    ledcWrite(LEFT_CHANNEL, 255);
  }else{
    ledcWrite(LEFT_CHANNEL, 0);
  }
  digitalWrite(PINDC_RIGHT_BACK, right0); // Стан правого колеса
  if(right1 == HIGH){
    ledcWrite(RIGHT_CHANNEL, 255);
  }else{
    ledcWrite(RIGHT_CHANNEL, 0);
  }
}
```

Рис. 3.3. Метод controlDC

```
void servoWrite(uint8_t channel, uint8_t angle) {
  // Максимальна заповненість ШІМ
  uint32_t maxDuty = (pow(2,SERVO_RESOLUTION)-1)/10;
  // Мінімальна заповненість ШІМ
  uint32_t minDuty = (pow(2,SERVO_RESOLUTION)-1)/20;
  // Перетворення куту у відповідне значення ШІМ
  uint32_t duty = (maxDuty-minDuty)*angle/180 + minDuty;
  // Подання ШІМ на канал серво
  ledcWrite(channel, duty);
}
```

Рис. 3.4. Метод servoWrite

Після визначення та налаштування параметрів камери виконується виклик функції initCamera() (рис. 3.5), результат якої зберігається у змінній cameraInitState. Якщо ініціалізація не була успішною, подальше виконання функції setup() припиняється, що запобігає роботі системи з некоректно налаштованою камерою.

```
int cameraInitState = -1; // Стан ініціалізації камери

cameraInitState = initCamera();
if (cameraInitState != 0){
  return;
}
```

Рис. 3.5. Ініціалізація камери

Наведений фрагмент коду (рис. 3.6.1, 3.6.2) відповідає за апаратну ініціалізацію камери на платформі ESP32-CAM. У функції initCamera() конфігуруються всі необхідні піні для передачі даних, синхронізаційні сигнали, інтерфейси SDA/SCL, живлення та скидання сенсора. Крім того, задаються параметри роботи камери: частота тактового сигналу XCLK, формат зображення JPEG, розмір кадру та якість JPEG. При наявності PSRAM встановлюються більший розмір кадру, краща якість JPEG і кілька буферів для одночасного зберігання кадрів; у разі її відсутності використовується менший

кадр і один буфер. Ініціалізація камери виконується викликом esp_camera_init(), а у разі помилки функція повертає код -1, що сигналізує про неможливість роботи камери. Після успішної ініціалізації отримується об’єкт сенсора і встановлюється початковий розмір кадру для захоплення зображень.

```
int initCamera(){
  camera_config_t config; // Всі налаштування камери
  config.ledc_channel = LEDC_CHANNEL_0; // ШІМ канал
  config.ledc_timer = LEDC_TIMER_0; // Таймер
  // Лінії даних, що передають значення про піксель зображення
  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;
  // Синхронізаційні сигнали
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  // SDA та SCL
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  // Живлення та скидання
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  // Параметри
  config.xclk_freq_hz = 20000000; // Частота тактового сигналу XCLK
  config.pixel_format = PIXFORMAT_JPEG; // Формат зображення - JPEG
  // Ініціалізація камери з високими параметрами для заздалегідь
виділених великих буферів
  if(psramFound()){ // Якщо є PSRAM
    config.frame_size = FRAMESIZE_SVGA; // 1600x1200
```

Рис.3.6.1 Метод initCamera

```
    config.jpeg_quality = 18; // Краща якість jpeg
    // Кількість кадрів, що одночасно зберігаються в буфері
    config.fb_count = 2;
  } else { // Якщо нема PSRAM
    config.frame_size = FRAMESIZE_VGA;  // 640x480
    config.jpeg_quality = 12; // Гірша якість jpeg
    config.fb_count = 1;
  }
  // Виклик функції ініціалізації камери
  // Змінна збереження результату ініціалізації
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) { // Чи пройшла ініціалізація успішно
    Serial.printf("Camera init failed with error 0x%x", err);
    return -1;
  }
  sensor_t * s = esp_camera_sensor_get(); // Сенсор камери
  // Розмір кадру камери
  s->set_framesize(s, FRAMESIZE_VGA);
  return 0;
}
```

Рис. 3.6.2. Метод initCamera (продовження)

Директиви препроцесора #define створюють макроси, які не виділяють пам’ять, а просто замінюють текст на етапі компіляції. У випадку ESP32-CAM вони визначають фізичні піни для сенсора камери: GPIO для живлення, сигналу тактування XCLK, ліній даних Y2–Y9, синхронізаційних сигналів VSYNC, HREF, PCLK та шини керування SCCB (SDA/SCL) (рис. 3.7). Завдяки цьому у функції initCamera() можна використовувати зрозумілі іменовані константи замість чисел, що забезпечує правильне апаратне підключення, коректну ініціалізацію сенсора та стабільне захоплення кадрів, без додаткового використання пам’яті для цих значень.

Встановлюється режим точки доступу (AP) на ESP32 для забезпечення прямого підключення інших пристроїв до плати (рис. 3.9). Створюється власна Wi-Fi мережа з заданим ім’ям та паролем, після чого відображається IP-адреса

точки доступу для спрощеного підключення клієнтів. Одночасно запускається WebSocket-сервер для обміну даними в реальному часі з реєстрацією обробника подій та активується UDP-сервер для прийому команд і даних через UDP-протокол, що забезпечує двосторонній зв'язок із зовнішніми пристроями.

Для роботи зі смартфоном нам необхідно передавати відео на пристрій та отримувати з нього команди для руху (рис. 3.8).

```
#define CAMERA_MODEL_AI_THINKER // Модель камери
// Контрольні сигнали
#define PWDN_GPIO_NUM 32 // Живлення камери
#define RESET_GPIO_NUM -1 // Скидання камери (не використовується)
#define XCLK_GPIO_NUM 0 // Генератор тактового сигналу XCLK
#define SIOD_GPIO_NUM 26 // SDA шини SCCB
#define SIOC_GPIO_NUM 27 // SCL шини SCCB
// Лінії даних, що передають пікселі зображення по байту за такт
#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
// Синхронізаційні сигнали
#define VSYNC_GPIO_NUM 25 // Вертикальна синхронізація. Кадр
завершено.
#define HREF_GPIO_NUM 23 // Горизонтальна синхронізація. Рядок
завершено.
#define PCLK_GPIO_NUM 22 // Піксельний такт. Піксель завершено.
```

Рис. 3.7. Визначення пінів для камери

```
#include <WebSocketsServer.h>
#include <WiFi.h>
#include <WiFiUdp.h>
```

Рис. 3.8. Бібліотеки для взаємодії між пристроями

Бібліотека WebSocketsServer забезпечує створення та обробку WebSocket-підключень, що дозволяє реалізувати двосторонній обмін даними між мікроконтролером і клієнтом у реальному часі. Бібліотека WiFi відповідає за підключення пристрою до бездротової мережі, налаштування параметрів Wi-Fi та підтримку мережевої взаємодії. Бібліотека WiFiUdp використовується для передачі та приймання даних через протокол UDP, що дозволяє здійснювати легкий та швидкий обмін пакетами між пристроями без встановлення постійного з'єднання (рис. 3.8).

Створюються об'єкти для обробки мережевих з'єднань: WebSocket-сервер на порту 86 для обміну даними з клієнтами в реальному часі та UDP-сервер на порту 6868 для прийому команд.

```
WiFi.mode(WIFI_AP); // Встановлюємо режим точки доступу
WiFi.softAP(ssid, password); // Створюємо власну Wi-Fi мережу
webSocket.begin(); // Запуск WebSocket-серверу
webSocket.onEvent(webSocketEvent); // Реєстрація функції
UDPServer.begin(UDPPort); // Запуск UDP-серверу
```

Рис.3.9. Режим точки доступу

Метод webSocketEvent обробляє події WebSocket, відстежуючи підключення та відключення клієнтів і реєструючи тип отриманих повідомлень (текстові, бінарні, фрагментовані або помилки). Він забезпечує контроль стану з'єднання та прийом даних від клієнта (рис. 3.10).

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload,
size_t length) {
  switch(type) {  // Перевіряємо тип події
      case WStype_DISCONNECTED: // Клієнт від'єднався
          camNo = num;
          clientConnected = false;
          break;
      case WStype_CONNECTED: // Клієнт під'єднався
          clientConnected = true;
```

Рис. 3.10. Метод webSocketEvent

Метод loop() забезпечує безперервну роботу платформи, організовуючи обробку подій, комунікацію з клієнтами та виконання основних завдань керування роботом у реальному часі.

Якщо клієнт під’єднаний до плати, спочатку захоплюється кадр з камери за допомогою grabImage() (рис. 3.11.1, 3.11.2), при цьому кадр перевіряється на формат JPEG і конвертується, якщо потрібно. Після цього готовий кадр надсилається клієнту через WebSocket у вигляді масиву байтів, що забезпечує передачу зображень у реальному часі при активному підключенні.

```
webSocket.loop(); // Обробка подій
  if(clientConnected == true){  // Якщо клієнт під'єднаний до плати
    grabImage(jpgLength, jpgBuff);  // Захоплення кадру
    // Надсилання кадра клієнту
    webSocket.sendBIN(camNo, jpgBuff, jpgLength);
  }

// Захоплення кадру з камери й конвертація у JPEG
// Код успіху або помилки (довжина jpeg-зображення, вказівник на буфер пам'яті)
esp_err_t grabImage( size_t& jpg_buf_len, uint8_t *jpg_buf){
  camera_fb_t * fb = NULL;      // Кадр зображення
  esp_err_t res = ESP_OK;       // Код результату виконання функції
  fb = esp_camera_fb_get();     // Функція отримання кадру з камери
  uint8_t *jpg_buf_tmp = NULL;  // Тимчасово зберігає JPEG-image
  if (!fb) {  // Чи отримано кадр
      res = ESP_FAIL;
  }else{
    // Перевірка формату кадра
    if(fb->format != PIXFORMAT_JPEG){ // Якщо кадр не JPEG
        // Конвертація у JPEG
        bool jpeg_converted = frame2jpg(fb, 80, &jpg_buf_tmp,
        &jpg_buf_len);
        // Копіювання до основного буферу
        memcpy(jpg_buf, jpg_buf_tmp, jpg_buf_len);
```

Рис. 3.11.1. Метод захоплення зображення з камери

```
            fb = NULL;
            if(!jpeg_converted){  // Якщо не успішно конвертовано
                res = ESP_FAIL; // Помилка конвертації
            }
        } else {  // Якщо кадр в JPEG
            // Копіювання до буферу без конвертації
            jpg_buf_len = fb->len;
            memcpy(jpg_buf, fb->buf, jpg_buf_len);
        }
        // Звільнення кадрового буферу
        esp_camera_fb_return(fb);
    }
    return res;
}
```

Рис. 3.11.2. Метод захоплення зображення з камери (продовження)

Конвертація кадрів у формат JPEG використовується для зменшення розміру зображень та економії пам’яті ESP32-CAM, що дозволяє ефективно передавати їх через WebSocket та відображати на клієнтських пристроях без додаткової обробки.

Необхідно періодично обробляти UDP-пакети та оновлювати положення сервоприводу. Для цього відстежується час від останньої дії, і як тільки проходить заданий інтервал, виконується обробка вхідних команд і керування сервоприводом (рис. 3.12). Такий підхід дозволяє одночасно підтримувати зв’язок з клієнтом і управляти механізмами у реальному часі без затримок.

```
unsigned long currentMillis = millis(); // Отримання часу у мілісекундах від моменту увімкнення плати
// Затримка
if (currentMillis - previousMillisServo >= intervalServo) {
   previousMillisServo = currentMillis;  // Оновлення змінної на поточний час
  processUDPData();    // Обробка UDP-пакетів
  controlServo();      // Керування серво
```

Рис. 3.12. Обробка UDP-запитів

Для обробки UDP-команд отримується UDP-пакет, визначається IP та порт відправника, а потім команда порівнюється зі списком доступних дій. Якщо команда збігається, виконується відповідна функція для керування рухом двигунів, сервоприводу або активації світлодіоду. Після обробки буфер очищується, щоб підготуватися до наступного пакета (рис. 3.13.1, 3.13.2).

```
struct CommandHandler {
    const char* cmd;
    std::function<void()> func;
};

CommandHandler handlers[] = {
    {"whoami", [](){
        UDPServer.beginPacket(addrRemote, portRemote);
        String res = "ESP32-CAM";
        UDPServer.write((const uint8_t*)res.c_str(), res.length());
        UDPServer.endPacket();
    }},
    {"forward", [](){ controlDC(LOW,HIGH,LOW,HIGH); }},
    {"backward", [](){ controlDC(HIGH,LOW,HIGH,LOW); }},
    {"left", [](){ controlDC(LOW,LOW,LOW,HIGH); }},
    {"right", [](){ controlDC(LOW,HIGH,LOW,LOW); }},
    {"stop", [](){ controlDC(LOW,LOW,LOW,LOW); }},
    {"camup", [](){ servoUp = true; }},
    {"camdown", [](){ servoDown = true; }},
    {"camstill", [](){ servoUp = false; servoDown = false; }},
    {"ledon", [](){ digitalWrite(LED_BUILT_IN, HIGH); }},
    {"ledoff", [](){ digitalWrite(LED_BUILT_IN, LOW); }},
};

void processUDPData(){
  int cb = UDPServer.parsePacket();
  if (!cb) return;
  UDPServer.read(packetBuffer, RECVLENGTH);
```

Рис. 3.13.1 Реагування системи на отриманий UDP-пакет

```
    addrRemote = UDPServer.remoteIP(); // IP-адреса відправника
    portRemote = UDPServer.remotePort(); // Порт відправн. UDP-пакету
    String cmd = String((char*)packetBuffer);
    // Загальні команди
    for (auto &h : handlers) {
      if (cmd.equals(h.cmd)) {
        h.func();
          memset(packetBuffer, 0, RECVLENGTH);
          return;
        }
    }
    memset(packetBuffer, 0, RECVLENGTH);
}
```

Рис. 3.13.2 Реагування системи на отриманий UDP-пакет (продовження)

Необхідно керувати положенням сервоприводом в залежності від команд на підйом або опускання камери (рис. 3.14). Якщо активовано підйом, сервопривід поступово нахиляється вгору, якщо опускання вниз, з кроком у 1 градус за цикл. Після обчислення нового положення воно передається до сервоприводу через servoWrite, забезпечуючи плавний рух механізму.

```
void controlServo(){
  if(servoUp){
    if(posServo>2){
      posServo -= 1;  // Нахиляється вгору на 2 градуси
    }
  }
  if(servoDown){
    if(posServo<180){
      posServo += 1;  // Нахиляється вниз на 2 градуси
    }
  }
  servoWrite(SERVO_PITCH_CHANNEL,posServo);
}
```

Рис. 3.14. Контроль сервоприводу

Після всіх цих дій плата прошита та готова отримувати команди та транслювати відео.

3.2 Розробка програмного забезпечення для керування рухом мобільної робототехнічної системи

Керування роботом та виконання автоматизованих задач здійснюватиметься за допомогою Android-додатка. Для його розробки використовуються Java та XML. Вибір Java зумовлений її стабільністю, широкою підтримкою, сумісністю з великою кількістю існуючих бібліотек, а також наявністю значної кількості прикладів і документації, що спрощує інтеграцію з апаратною частиною та мережевими модулями. Інтерфейс користувача, призначений для керування роботом, реалізовано у вигляді фрагментів (Fragment), що забезпечує гнучку структуру застосунку та дає змогу ефективно розподіляти функціональні модулі в межах одного Activity.

Для розробки буде використовуватись бібліотека OpenCV завантажена з офіційного GitHub репозиторію. В більшості бібліотек відсутні класичні трекери, тому їх буде отримано з репозиторію з додатковими модулями – opencv_contrib.

Першим та найважливішим кроком при розробці андроїд додатку, що буде взаємодіяти з платою мобільного роутера через інтернет є надання відповідного доступу у Android Manifest (рис. 3.15). Без цього додаток не зможе здійснити підключення до мережі, включно з доступом до Wi-Fi, що робить неможливою передачу команд або отримання відеопотоку з плати.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Рис. 3.15. Необхідний дозвіл для роботи IoT системи

Буде створено кнопки для ручного керування роботом, для увімкнення світлодіоду, режиму object detection та режиму object tracking (рис. 3.16), також буде створено свайп-меню (рис. 3.17) для налаштування системи (зміна моделі

object tracking, вибір елемента дії – нахил серводвигуна або рух колес назад/вперед, розмір центральної рамки).

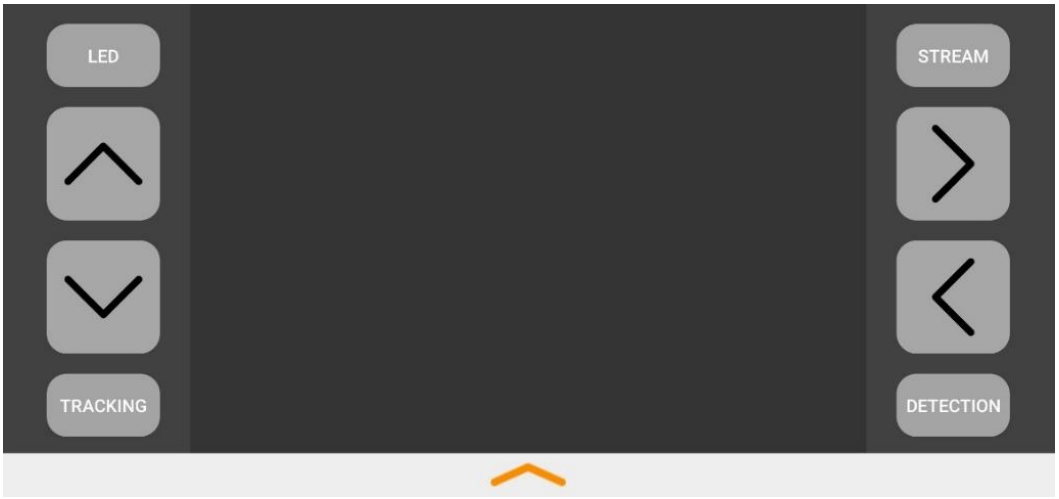

Рис. 3.16. Інтерфейс користувача

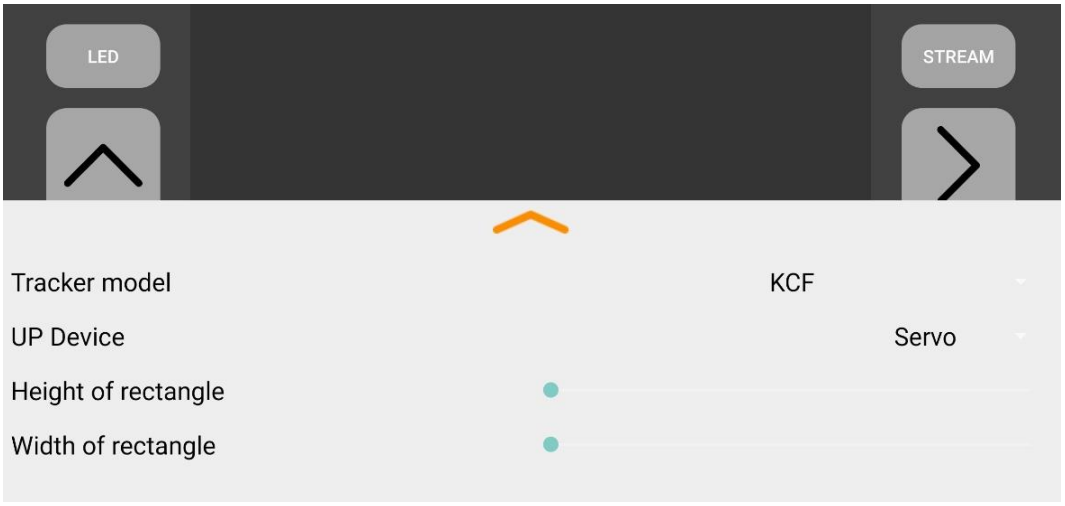

Рис. 3.17. Свайп-меню

Метод onCreate() у Android (рис. 3.18) це один із ключових методів життєвого циклу. Він призначений для початкової ініціалізації: налаштування змінних, створення об’єктів, підключення до ресурсів або запуску фонових процесів. У цьому методі зазвичай не відбувається безпосереднє створення UI для цього існує onCreateView().

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Створення UDP-клієнта та запуск сервера
    mUdpClient = new UDPSocket(12345);
    mUdpClient.runUdpServer();
    try {
        // IP-адреса плати
        mServerAddr = InetAddress.getByName(DEFAULT_ESP_IP);
    } catch (UnknownHostException e) {
        Log.e(TAG, "Invalid server IP: " + DEFAULT_ESP_IP, e);
    }
    // Ініціалізація OpenCV бібліотеки
    if (!OpenCVLoader.initDebug()) {
        Log.d(TAG, "Internal OpenCV library not found. Using OpenCV Manager for
initialization");
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_4_0, getActivity(),
null);
    }
}
```

Рис. 3.18. Метод onCreate

Метод onCreateView() у фрагменті (рис. 3.19) відповідає за створення та відображення інтерфейсу користувача. У ньому зазвичай виконується inflation XML-розмітки, пошук елементів UI та їхня початкова конфігурація. Фактично це точка, де фрагмент налаштовує свій візуальний вигляд і стає готовим до взаємодії з користувачем.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle
savedInstanceState) {
    // Створення UI з xml файлу
    View rootView = inflater.inflate(R.layout.esp32cam_fragment, parent, false);
    // Посилання на елемент, що буде виводити зображення з плати
    mServerImageView = rootView.findViewById(R.id.imageView);
```

Рис. 3.19. Метод onCreateView

Для коректної взаємодії користувача з інтерфейсом необхідно реалізувати візуальне відображення стану кнопки під час її натискання (рис. 3.20). Після активації елемент повинен змінювати фон і колір тексту, а система має надсилати відповідну команду на контролер. Це забезпечує зрозумілий зворотний зв’язок і підтверджує виконання дії.

```
private boolean toggleButton(Button btn, boolean state, byte[] onCommand, byte[]
offCommand) {
    state = !state; // Перемикання стану активації кнопки
    // Зміна фону кнопки
    btn.setBackgroundResource(state ? R.drawable.active_button :
R.drawable.button_bg);
    // Зміна кольору тексту
    btn.setTextColor(state ? Color.BLACK : Color.WHITE);
    // Надсилання команди на плату
    if (onCommand != null && offCommand != null) {
        mUdpClient.sendBytes(mServerAddr, mServerPort, state ? onCommand :
offCommand);
    }
    return state;
}
```

Рис. 3.20. Зміна стану кнопки

Для реалізації передачі команд на мікроконтролер використовується відправлення UDP-пакетів у фоновому режимі (рис. 3.21). Дані надсилаються в окремому потоці, що запобігає блокуванню основного інтерфейсу. Перед формуванням пакета виконується перевірка доступності сокету, після чого створений UDP-пакет передається на вказану IP-адресу та порт. Це забезпечує стабільну та асинхронну роботу мережевої взаємодії.

```
@SuppressLint("NewApi")
public void sendBytes(final InetAddress addr, final int port, final byte[] data)
{
    // Відправка команд відбувається в окремому потоці
    Executors.newSingleThreadExecutor().submit(() -> {
        try { // Перевірка доступності сокету
            if (udpSendSocket != null && !udpSendSocket.isClosed()) {
                // Створення та відправка UDP-пакету
                DatagramPacket dp = new DatagramPacket(data, data.length, addr,
port);
                udpSendSocket.send(dp);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
}
```

Рис. 3.21. Відправлення UDP-пакетів на плату

Для забезпечення ручного керування мобільною платформою в інтерфейс були додані кнопки напрямків руху. Кожна з них використовує обробник подій дотику, який реагує як на натискання, так і на відпускання (рис. 3.22). У момент натискання кнопки формується й надсилається відповідна UDP-команда для

руху вперед, назад, ліворуч або праворуч. Після відпускання кнопки система автоматично передає команду зупинки. Такий підхід дозволяє реалізувати інтуїтивне та оперативне керування платформою у режимі реального часу.

```
ImageButton upBtn = rootView.findViewById(R.id.upBtn);  // Вперед
ImageButton downBtn = rootView.findViewById(R.id.downBtn); // Назад
ImageButton leftBtn = rootView.findViewById(R.id.leftBtn); // Вліво
ImageButton rightBtn = rootView.findViewById(R.id.rightBtn); // Вправо

View.OnTouchListener driveListener = new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        int action = event.getActionMasked();
        if (action == MotionEvent.ACTION_DOWN) { // Натискання кнопки
            int id = v.getId();
            if (id == R.id.upBtn) {
                mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestForward);
                Log.d(TAG, "Drive: FORWARD (button)");
            } else if (id == R.id.downBtn) {
                mUdpClient.sendBytes(mServerAddr, mServerPort,
mRequestBackward);
                Log.d(TAG, "Drive: BACKWARD (button)");
            } else if (id == R.id.leftBtn) {
                mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestLeft);
                Log.d(TAG, "Drive: LEFT (button)");
            } else if (id == R.id.rightBtn) {
                mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestRight);
                Log.d(TAG, "Drive: RIGHT (button)");
            }
            return true;
        } else if (action == MotionEvent.ACTION_UP || action ==
MotionEvent.ACTION_CANCEL) {
            // Зупинка руху при відпусканні кнопок
            mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestStop);
            Log.d(TAG, "Drive: STOP (button release)");
            v.setPressed(false);
            return true;
        }
        return false;
    }
};

// Прив'язка слухача до кнопок
if (upBtn != null) upBtn.setOnTouchListener(driveListener);
if (downBtn != null) downBtn.setOnTouchListener(driveListener);
if (leftBtn != null) leftBtn.setOnTouchListener(driveListener);
if (rightBtn != null) rightBtn.setOnTouchListener(driveListener);
```

Рис. 3.22. Ручне керування роботом (продовження)

Для організації отримання відеопотоку з робоплатформи використовується підключення через WebSocket. У інтерфейсі реалізовано кнопку, яка вмикає або вимикає цей режим (рис. 3.23). Після активації кнопки відбувається ініціалізація встановлення WebSocket-з’єднання, через яке

надходять кадри з плати. У разі повторного натискання з'єднання коректно закривається, а клієнтський об'єкт видаляється.

```
Button streamBtn = rootView.findViewById(R.id.streamBtn);
streamBtn.setOnClickListener(v -> {
    mStream = toggleButton(streamBtn, mStream, null, null);
    // Якщо кнопку активовано
    if (mStream) {
        connectWebSocket();
    }
    // Якщо кнопку деактивовано
    else if (mWebSocketClient != null) {
        // Закриття websocket
        try {
            mWebSocketClient.close();
        } catch (Exception ignored) {}
        // Видалення клієнта
        mWebSocketClient = null;
    }
});
```

Рис. 3.23. Запуск трансляції

На етапі ініціалізації WebSocket формується URI сервера та створюється клієнтський об'єкт. Після відкриття з'єднання система переходить у режим приймання даних. У разі отримання двійкових повідомлень виконується декодування JPEG-кадрів у формат Bitmap та подальше оновлення інтерфейсу користувача. У випадках розриву з'єднання або виникнення помилок реалізовано автоматичне перепідключення з використанням затримки, що забезпечує стабільність роботи відеотрансляції та безперервність обробки вхідних кадрів (рис. 3.24.1, 3.24.2).

```
public void connectWebSocket() {
    URI uri;
    try {
        // Адреса WebSocket-сервера
        uri = new URI("ws://" + DEFAULT_ESP_IP + ":" + WS_PORT + "/");
    } catch (URISyntaxException e) {
        e.printStackTrace();
        return;
    }
    mWebSocketClient = new WebSocketClient(uri) {
        @Override
        public void onClose(int i, String s, boolean b) {
            Log.d("Websocket", "Closed " + s);
            // Спроба перепідключення
```

Рис. 3.24.1. WebSocket підключення

```
            handler.postDelayed(() -> {
                try {
                    Log.d("Websocket", "Reconnecting...");
                    connectWebSocket();
                } catch (Exception e) {
                    Log.e("Websocket", "Reconnect failed", e);
                }
            }, 1000);
        }
        @Override
        public void onMessage(String message) {
            Log.d("Websocket", "Receive text");
        }
        @Override
        public void onMessage(ByteBuffer message) {
            byte[] imageBytes = new byte[message.remaining()];
            message.get(imageBytes);
            // Декодування зображення
            final Bitmap bmp = BitmapFactory.decodeByteArray(imageBytes, 0,
imageBytes.length);
            if (bmp == null) {
                return;
            }
            // Оновлення UI
            getActivity().runOnUiThread(() -> {
                mBitmapGrab = bmp;  // Оригінальний кадр
                // Ширина елементу ImageView
                int viewWidth = mServerImageView.getWidth();
                // Співвідношення сторін
                float imageRatio = (float) bmp.getHeight() / bmp.getWidth();
                // Нова висота зображення
                int dispViewH = (int) (viewWidth * imageRatio);
                Bitmap bmpToDisplay = bmp;
                // Якщо розмір було змінено
                if (viewWidth != lastViewWidth || dispViewH != lastDispHeight) {
                    // Масштабування зображення
                    bmpToDisplay = Bitmap.createScaledBitmap(bmp, viewWidth,
dispViewH, false);
                    lastViewWidth = viewWidth;
                    lastDispHeight = dispViewH;
                }
                // Встановлення зображення в UI
                mServerImageView.setImageBitmap(bmpToDisplay);
                // Якщо увімкнено обробку кадру
                if ((mObjTracking || mObjDet) && mBitmapGrab != null) {
                    processing();
                }
            });
        }
        @Override
        public void onError(Exception e) {
            Log.d("Websocket", "Error " + e.getMessage());
            // Спробувати перепідключитись
            handler.postDelayed(() -> {
                try {
                    connectWebSocket();
                } catch (Exception ex) { }
            }, 1000);
        }
    };
    mWebSocketClient.connect();
}
```

Рис. 3.24.2. WebSocket підключення

Після декодування чергового кадру, у методі connectWebSocket() система одразу викликає метод processing(), що працює паралельно з оновленням відеопотоку на екрані. Це означає, що кожен отриманий кадр не лише відображається, але й додатково проходить обробку в режимі реального часу. Метод processing() готує зображення, визначає активний режим (виявлення або відстеження об’єкта) та викликає відповідний алгоритм (рис. 3.25).

```
public void processing()
{
    int overlayWidth = mTrackingOverlay.getWidth();      // Ширина overlay
    int overlayHeight = mTrackingOverlay.getHeight();    // Висота overlay
    mRadiusCircle = 0;                                   // Радіус кола

    prepareBitmapForDetection();

    switch (mSelectedTracker)
    {
        case "ObjectTracking":
            processObjectTracking(overlayWidth, overlayHeight);
            break;
        default:
            processNone();
            break;
    }

    mSelectedTrackerPre = mSelectedTracker; // Овновлення стану трекера
    mTrackingOverlay.invalidate();          // Оновлення графічного відображення
}
```

Рис. 3.25. Метод обробки візуальних даних

Для реалізації детекції об’єктів у мобільному застосунку було використано нейромережеву модель SSDLite MobileNet V2 Quantized. Ця модель належить до сімейства одностадійних детекторів (Single Shot Detector), які виконують виявлення об’єктів без проміжного етапу регіональних пропозицій. SSDLite – це оптимізована версія класичного SSD, що використовує групові та глибокі згортки (Depthwise Separable Convolutions).

Дана модель навчена на наборі даних COCO та використовує фіксований перелік класів об’єктів (близько 80). До них належать люди, транспортні засоби, тварини, предмети побуту, харчові продукти та інші типові об’єкти навколишнього середовища.

Під час ініціалізації детектора (рис. 3.26) задаються основні параметри: шлях до моделі, розмір вхідного зображення (300×300 пікселів), поріг впевненості, кількість потоків та тип пристрою виконання (CPU). Якщо інтерфейс реалізації підтримує окремий фоновий потік, детектор запускається у ньому, що дозволяє виконувати обробку кадрів паралельно з відтворенням відеопотоку.

```
try {
    detectorSSD = TFLiteObjectDetectionSSDAPIModel.create(
            getActivity().getAssets(),
            "ssdlite_mobilenet_v2_quantized.tflite",
            "",
            300,
            Classifier.Device.CPU,
            MyConstants.MODEL_TYPE.UINT8,
            0.5f,
            1,
            CamResolution.getWidth(),
            CamResolution.getHeight()
    );
    try { detectorSSD.startThread(); } catch (Exception ignored) {}
} catch (IOException e) {
    detectorSSD = null;
}
```

Рис. 3.26. Ініціалізація детектора

Для підготовки кадру до детекції необхідно конвертувати його у формат, придатний для роботи нейромережі (рис. 3.27). Якщо активовано режим виявлення об’єктів (mObjDet) і детектор ініціалізовано, останній отримує актуальний кадр mBitmapGrab через метод setBitmap().

```
private void prepareBitmapForDetection()
{
    if(mObjDet) {    // Увімкнено режим виявлення об'єктів
        if (detectorSSD != null && mBitmapGrab != null) {
            try {
                detectorSSD.setBitmap(mBitmapGrab);
            } catch (Exception e) {
                Log.w(TAG, "setBitmap failed: " + e.getMessage());
            }
        }
    }
}
```

Рис. 3.27. Підготовка кадру до режиму Detection

Після отримання списку розпізнаних об’єктів (detectorSSDResult) для кожного знайденого об’єкта обчислюються координати рамки (bbox) з урахуванням розміру елемента overlay. Далі на Canvas малюється зелена рамка навколо об’єкта та підпис із назвою класу і ймовірністю детекції (рис. 3.28.1, 3.28.2).

```
mTrackingOverlay.addCallback (
        new OverlayView.DrawCallback()
        {
            @Override
            public void drawCallback(Canvas canvas)
            {
                // Ширина та висота overlay
                int overlayW = mTrackingOverlay.getWidth();
                int overlayH = mTrackingOverlay.getHeight();
                if (detectorSSD != null && mObjDet && mBitmapGrab != null) {
                    try {
                        // Результати детекції
                        ((TFLiteObjectDetectionSSDAPIModel)
detectorSSD).getResult(detectorSSDResult);
                    } catch (Exception e) {
                        Log.w(TAG, "getResult() failed: " + e.getMessage());
                    }

                    List<TFLiteObjectDetectionSSDAPIModel.Recognition>
localResults;
                    synchronized (detectorLock) {
                        localResults = new ArrayList<>(detectorSSDResult);
                    }

                    // Якщо виявлено об'єкти на кадрі
                    if (!localResults.isEmpty()) {
                        Paint paint = new Paint();
                        paint.setColor(Color.rgb(0, 255, 0));
                        paint.setStrokeWidth(6);
                        paint.setStyle(Paint.Style.STROKE);

                        Paint paintText = new Paint();
                        paintText.setColor(Color.WHITE);
                        paintText.setTextSize(36);
                        paintText.setStyle(Paint.Style.FILL);
                        for (TFLiteObjectDetectionSSDAPIModel.Recognition det :
localResults) {
                            RectF loc = det.getLocation(); // Координати bbox
                            if (loc == null) continue;
                            // Масштабування до розміру overlay
                            float left = loc.left * overlayW;
                            float top = loc.top * overlayH;
                            float right = loc.right * overlayW;
                            float bottom = loc.bottom * overlayH;
                            // Обрізання координат, щоб не виходили за межі
overlay
                            if (left < 0) left = 0;
                            if (top < 0) top = 0;
                            if (right > overlayW) right = overlayW;
                            if (bottom > overlayH) bottom = overlayH;
```

Рис. 3.28.1. Малювання обмежувальної рамки об’єкта

```
                            // Малювання bbox
                            canvas.drawRect(left, top, right, bottom, paint);
                            // Назва об'єкта
                            String title = det.getTitle() != null ?
det.getTitle() : "obj";
                            String label = String.format("%s %.2f", title,
det.getConfidence());
                            float padding = 6f;
                            float textW = paintText.measureText(label);
                            float textH = paintText.getTextSize();
                            float bgLeft = left;
                            float bgTop = Math.max(0, top - textH - 2*padding);
                            float bgRight = Math.min(overlayW, bgLeft + textW +
2*padding);
                            float bgBottom = bgTop + textH + 2*padding;
                            Paint bg = new Paint();
                            bg.setColor(Color.argb(180, 0, 0, 0));
                            bg.setStyle(Paint.Style.FILL);
                            canvas.drawRect(bgLeft, bgTop, bgRight, bgBottom,
bg);
                            paintText.setColor(Color.WHITE);
                            canvas.drawText(label, bgLeft + padding, bgTop +
textH + padding/2, paintText);
                        }
                    }
                }
...
```

Рис. 3.28.2. Малювання обмежувальної рамки об’єкта (продовження)

Задля реалізувати відстеження об’єкта, потрібно спочатку перевірити, чи об’єкт заблоковано для трекінгу. Якщо ні, то пропустити обробку. Далі підготувати кадр для обробки, перетворивши його у формат OpenCV Mat і конвертувавши кольори у BGR (рис. 3.29).

```
private void processObjectTracking(int overlayWidth, int overlayHeight)
{
    if (!mTargetLocked)
    {
        processNone();
        return;
    }

    ensureMat();
    Utils.bitmapToMat(mBitmapGrab, mMatGrab);
    Imgproc.cvtColor(mMatGrab, mMatGrab, Imgproc.COLOR_RGBA2BGR);

    if (mDrawing == Esp32CamFragment.Drawing.DRAWING)
    {
        startObjectSelection(overlayWidth, overlayHeight);
    }
    else
    {
        updateObjectTracking(overlayWidth, overlayHeight);
    }
}
```

Рис. 3.29. Підготовка кадру до режиму Tracking

Щоб користувач обрав об’єкт необхідно визначити координати прямокутника навколо цього об’єкту на overlay. Ці координати перетворюються у систему кадру OpenCV з урахуванням масштабування. Потім перевіряється валідність розмірів та положення прямокутника, щоб він не виходив за межі кадру. Після цього створюється трекер, ініціалізується об’єкт у вибраному прямокутнику і перемикається у режим відстеження (рис. 3.30).

```
private void startObjectSelection(int overlayWidth, int overlayHeight)
{
    int x0 = Math.min(mPoints[0].x, mPoints[1].x);
    int y0 = Math.min(mPoints[0].y, mPoints[1].y);
    int x1 = Math.max(mPoints[0].x, mPoints[1].x);
    int y1 = Math.max(mPoints[0].y, mPoints[1].y);

    // overlay → матриця
    int minX = (int)(x0 * mMatGrab.cols() / (float)overlayWidth);
    int minY = (int)(y0 * mMatGrab.rows() / (float)overlayHeight);
    int width = (int)((x1 - x0) * mMatGrab.cols() / (float)overlayWidth);
    int height = (int)((y1 - y0) * mMatGrab.rows() / (float)overlayHeight);

    // перевірка валідності
    if (width <= 0) width = 2;
    if (height <= 0) height = 2;
    if (minX < 0) minX = 0;
    if (minY < 0) minY = 0;
    if (minX + width > mMatGrab.cols()) width = mMatGrab.cols() - minX;
    if (minY + height > mMatGrab.rows()) height = mMatGrab.rows() - minY;

    mInitRectangle = new Rect2d(minX, minY, width, height);

    // 3-канальний мат
    mMatGrabInit = new Mat();
    Imgproc.cvtColor(mMatGrab, mMatGrabInit, Imgproc.COLOR_RGBA2BGR);

    createTracker();
    mTracker.init(mMatGrabInit, mInitRectangle);

    mDrawing = Drawing.TRACKING;
}
```

Рисунок 3.30. Початок виділення об’єкта для відстеження

Необхідно обрати трекер для відстеження об’єкта (рис. 3.31). Серед доступних варіантів: MedianFlow – швидкий і точний для плавного руху, але погано працює при різких переміщеннях; CSRT – більш точний і стійкий до змін масштабу та часткового перекриття, але повільніший; KCF – швидкий, добре працює на невеликих об’єктах, чутливий до масштабних змін; MOSSE – дуже швидкий, оптимальний для реального часу, але менш точний; TLD –

здатний відновлювати втрачений трек після зникнення об’єкта, але менш стабільний на коротких кадрах; MIL – стабільний для часткових перекриттів, але має нижчу швидкість. Вибір трекера залежить від вимог до швидкості, точності та умов руху об’єкта.

```
private void createTracker() {
    switch (mSelectedTracker) {
        case "TrackerMedianFlow":
            mTracker = TrackerMedianFlow.create();
            break;
        case "TrackerCSRT":
            mTracker = TrackerCSRT.create();
            break;
        case "TrackerKCF":
            mTracker = TrackerKCF.create();
            break;
        case "TrackerMOSSE":
            mTracker = TrackerMOSSE.create();
            break;
        case "TrackerTLD":
            mTracker = TrackerTLD.create();
            break;
        case "TrackerMIL":
            mTracker = TrackerMIL.create();
            break;
    }
}
```

Рис. 3.31. Вибір трекера для відстеження

Після ініціалізації трекера необхідно регулярно оновлювати його положення на нових кадрах (рис. 3.32). У разі успішного відстеження, координати об’єкта масштабуються під розміри інтерфейсу користувача, визначається центр прямокутника, що обмежує об’єкт, і на його основі надсилаються команди для керування рухом робоплатформи. Після цього оновлюється графічне відображення трекера.

Необхідно реалізувати автоматичне надсилання команд руху в залежності від положення об’єкта відносно камери (рис. 3.33.1, 3.33.2). Для цього спочатку визначається центральна зона, яка вважається нейтральною. Якщо об’єкт опиняється вище або нижче цієї зони, платформа отримує команду руху вперед або назад. Якщо об’єкт знаходиться всередині центральної зони, оцінюється горизонтальне зміщення від центру: при значному зміщенні вправо або вліво відправляється відповідна команда повороту, інакше рух зупиняється.

```
Private void updateObjectTracking(int overlayWidth, int overlayHeight)
{
    if (mTracker == null || mInitRectangle == null)
        return;

    Rect2d newRect = new Rect2d();
    boolean ok = mTracker.update(mMatGrab, newRect);

    if (!ok) {
        Log.d(TAG, «Tracker update failed (ok==false).»);
        // зупинити рух при втраті трекера
        sendDriveCommand(DriveCmd.STOP);
        return;
    }

    mInitRectangle = newRect;

    float scaleX = (float)overlayWidth / mMatGrab.cols();
    float scaleY = (float)overlayHeight / mMatGrab.rows();

    mPoints[0].x = (int)(newRect.x * scaleX);
    mPoints[0].y = (int)(newRect.y * scaleY);
    mPoints[1].x = (int)((newRect.x + newRect.width) * scaleX);
    mPoints[1].y = (int)((newRect.y + newRect.height) * scaleY);

    int centerYOverlay = (int)((newRect.y + newRect.height * 0.5) * scaleY);
    int centerXOverlay = (int)((newRect.x + newRect.width * 0.5) * scaleX);

    // Відправляємо керування рухом
    sendDriveControl(centerXOverlay, centerYOverlay, overlayWidth,
overlayHeight);

    mTrackingOverlay.postInvalidate();
}
```

Рис. 3.32. Оновлення кадрів

```
private void sendDriveControl(int centerXOverlay, int centerYOverlay, int
overlayWidth, int overlayHeight)
{
    // параметри центрального прямокутника (в тих же значеннях, що і
відображаємо)
    int centerBoxW = (int)(overlayWidth * CENTER_BOX_WIDTH_RATIO);
    int centerBoxH = (int)(overlayHeight * CENTER_BOX_HEIGHT_RATIO);
    int centerLeft = (overlayWidth - centerBoxW) / 2;
    int centerTop = (overlayHeight - centerBoxH) / 2;
    int centerRight = centerLeft + centerBoxW;
    int centerBottom = centerTop + centerBoxH;


    // Якщо об'єкт вище центра -> FORWARD
    if (centerYOverlay < centerTop)
    {
        sendDriveCommand(DriveCmd.FORWARD);
        return;
```

Рис. 3.33.1. Автоматичне надсилання сигналів на плату

```
    // Якщо об'єкт нижче центра -> BACKWARD
    if (centerYOverlay > centerBottom)
    {
        Log.d(TAG, "Decision: BACKWARD");
        sendDriveCommand(DriveCmd.BACKWARD);
        return;
    }

    // Інакше — горизонтальна логіка
    int midX = overlayWidth / 2;
    int deltaX = centerXOverlay - midX;

    // Адаптивний поріг: 10% від ширини або мінімум 60 пікселів
    int threshold = Math.max(60, overlayWidth / 10);
    Log.d(TAG, "sendDriveControl: deltaX=" + deltaX + " threshold=" +
threshold);

    if (deltaX > threshold) // праворуч -> turn right (праве колесо вперед)
    {
        Log.d(TAG, "Decision: RIGHT_TRACK (turn right)");
        sendDriveCommand(DriveCmd.RIGHT_TRACK);
    }
    else if (deltaX < -threshold) // ліворуч -> turn left (ліве колесо вперед)
    {
        Log.d(TAG, "Decision: LEFT_TRACK (turn left)");
        sendDriveCommand(DriveCmd.LEFT_TRACK);
    }
    else
    {
        Log.d(TAG, "Decision: STOP");
        sendDriveCommand(DriveCmd.STOP);
    }
}
```

Рис. 3.33.2. Автоматичне надсилання сигналів на плату (продовження)

Як альтернативний варіант, замість керування колесами реалізується управління сервоприводом, що тримає камеру, дозволяючи автоматично нахиляти її вгору або вниз для відстеження положення об’єкта (рис. 3.34).

```
private void sendCameraControl(int centerYOverlay, int overlayHeight) {
    int midY = overlayHeight / 2;
    int deltaY = centerYOverlay - midY;

    if (deltaY < -CAM_THRESHOLD) {
        mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestCamUp);
    } else if (deltaY > CAM_THRESHOLD) {
        mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestCamDown);
    } else {
        mUdpClient.sendBytes(mServerAddr, mServerPort, mRequestCamStill);
    }
}
```

Рис. 3.34. Автоматичне керування сервоприводом

Для відображення трекера на відео реалізується малювання графічних елементів поверх кадру (рис. 3.35.1, 3.35.2). Створюється центральний прямокутник, який слугує зоною контролю, а також bounding box об’єкта, що відстежується. При активному трекінгу на екрані додатково малюються лінії перетину через центр об’єкта та координати його положення, що допомагає візуалізувати поточне місцезнаходження об’єкта. Користувач може взаємодіяти з overlay через сенсорний екран.

```
mTrackingOverlay.addCallback (
    new OverlayView.DrawCallback() {
        @Override
        public void drawCallback(Canvas canvas) {
            ...
            if(mSelectedTracker.equals("ObjectTracking") && mStream) {
                Paint paint = new Paint();
                paint.setStrokeWidth(6);

                // Малюємо центральний прямокутник
                int centerBoxW = (int)(overlayW * CENTER_BOX_WIDTH_RATIO);
                int centerBoxH = (int)(overlayH * CENTER_BOX_HEIGHT_RATIO);
                int centerLeft = (overlayW - centerBoxW) / 2;
                int centerTop = (overlayH - centerBoxH) / 2;
                int centerRight = centerLeft + centerBoxW;
                int centerBottom = centerTop + centerBoxH;

                paint.setStyle(Paint.Style.STROKE);
                paint.setColor(Color.CYAN);
                canvas.drawRect(centerLeft, centerTop, centerRight,
centerBottom, paint);

                // Якщо є bounding box від трекера — намалювати його
                if(mDrawing != Drawing.CLEAR) {
                    paint.setStyle(Paint.Style.STROKE);
                    paint.setColor(Color.rgb(0, 0, 255));
                    paint.setStrokeWidth(8);
                    canvas.drawRect(mPoints[0].x, mPoints[0].y, mPoints[1].x,
mPoints[1].y, paint);

                    if (mDrawing == Drawing.TRACKING) {
                        paint.setColor(Color.GREEN);
                        float centerX = (mPoints[0].x + mPoints[1].x) / 2f;
                        float centerY = (mPoints[0].y + mPoints[1].y) / 2f;
                        canvas.drawLine(centerX, 0, centerX,
mTrackingOverlay.getHeight(), paint);
                        canvas.drawLine(0, centerY, mTrackingOverlay.getWidth(),
centerY, paint);

                        paint.setColor(Color.YELLOW);
                        paint.setStyle(Paint.Style.FILL);
                        String strX = (int)centerX + "/" +
mTrackingOverlay.getWidth();
```

Рис. 3.35.1. Малювання графічних елементів

```
                        String strY = (int)centerY + "/" +
mTrackingOverlay.getHeight();
                        canvas.drawText(strX, centerX / 2, centerY / 2 - 10,
paint);
                        canvas.drawText(strY, centerX + 10, centerY / 4, paint);
                    }
                }
            }
            else if(mSelectedTracker.equals("None") && mStream) {
                mInitTrackObj = false;
            }
        }
    }
);

mTrackingOverlay.setOnTouchListener((view, event) -> {
    if (!mObjTracking) return false; // Не малюємо, якщо трекер вимкнений

    final int X = (int) event.getX();
    final int Y = (int) event.getY();
    Log.d(TAG, ": " + X + " " + Y);
    mInitStream = true;
    mInitTrackObj = true;

    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
            handleActionDown(X, Y);
            break;
        case MotionEvent.ACTION_MOVE:
            handleActionMove(X, Y);
            break;
        case MotionEvent.ACTION_UP:
            handleActionUp();
            break;
        case MotionEvent.ACTION_POINTER_DOWN:
            handlePointerDown();
            break;
        case MotionEvent.ACTION_POINTER_UP:
            handlePointerUp();
            break;
    }
    return true;
});
```

Рис. 3.35.2. Малювання графічних елементів

Скомпілюємо та запустимо розроблений застосунок (рис. 3.36, 3.37).

Після запуску додатку потрібно підключитися до Wi-Fi точки доступу плати, після чого з’являється можливість керувати нею. Натискання кнопки «LED» вмикає світлодіод, а «Stream» запускає трансляцію відео з камери (рис. 3.36). Щоб відстежувати об’єкт, слід обвести його пальцем у прямокутник, після чого зафіксувати рамку другим натисканням. Після цього платформа

починає стежити за об’єктом, рухаючись так, щоб утримувати його в центрі встановленого прямокутника (рис. 3.37).

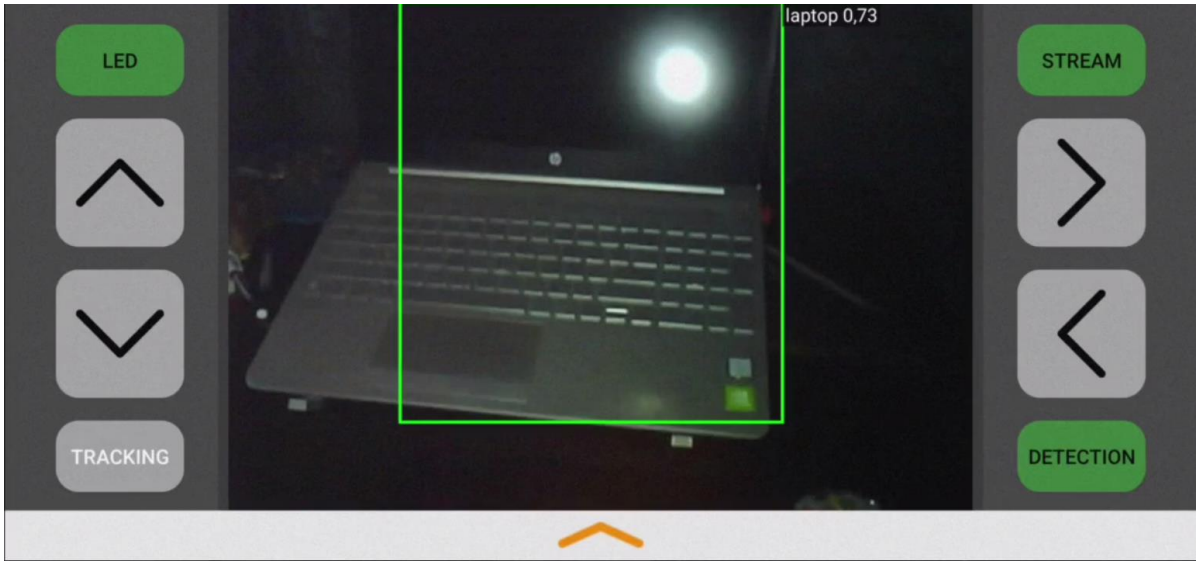

Рис. 3.36. Режим виявлення об’єктів

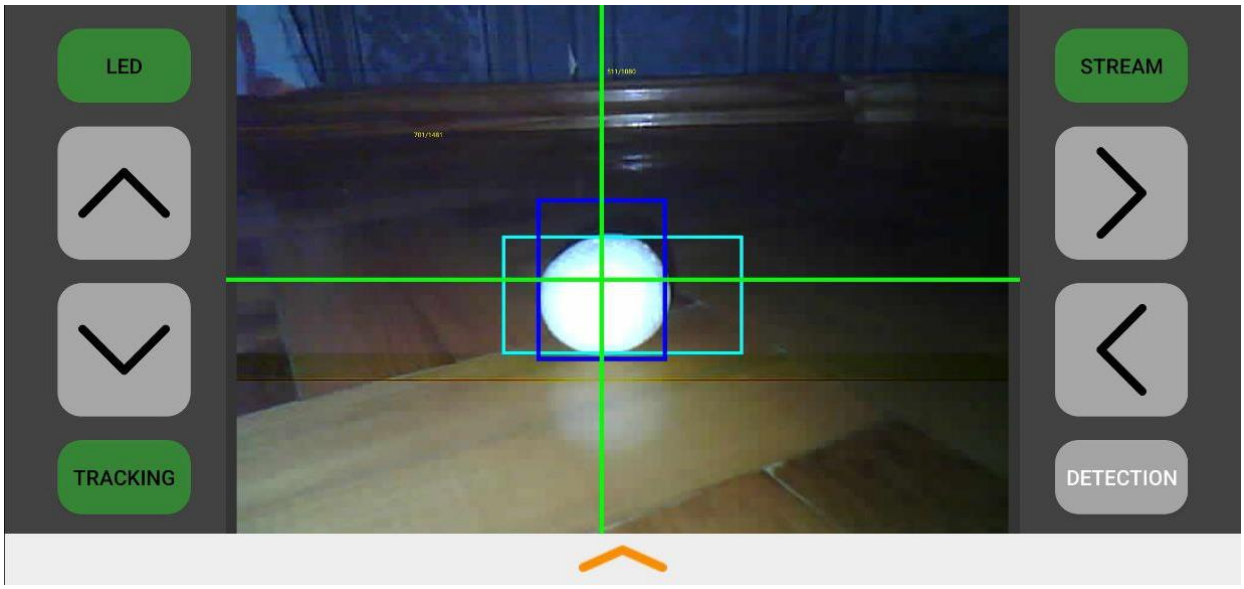

Рис. 3.37. Режим відстеження об’єкта

3.3. Аналіз працездатності, продуктивності та точності системи

Під час відстежування об’єктів на кадрі використовувались трекери MedianFlow, CSRT, KCF, MOSSE, TLD, MIL. Під час тестування застосунку, було проведено порівняння ефективності наведених трекерів (табл. 2).

Таблиця 2

Порівняння роботи трекерів для відстеження об’єкта

Трекер	FPS	Точність утримування об’єкта	Стійкість до змін масштабу	Особливості
MedianFlow	18-25	Висока	Висока	Швидкий, точний, стабільний
KCF	15-18	Середня	Середня	Збалансований
TLD	8-12	Середня	Висока	Кращий за CSRT та MIL, але гірший за MedianFlow
CSRT	5-8	Висока	Висока	Надто повільний, викликає смикання робота та втрату об’єкта через низький FPS, але при цьому точний
MOSSE	25-30	Низька	Низька	Найшвидший, але непридатний через вкрай низьку точність утримування
MIL	< 1	Висока	Низька	Найгірший, кадри майже не оновлюються, але фактично відстеження працює

Отже найкращі результати точності та ефективності показали MedianFlow, KCF, та TLD.

Окрім Tracking-режиму, у системі також був реалізований Detection-режим, що працює на основі моделі SSDLite MobileNet V2 Quantized. Обробка виконується незалежно для кожного кадру без збереження інформації про об’єкти. Результатом роботи алгоритму є візуальне відображення виявлених об’єктів у вигляді обмежувальних рамок з відповідними назвами класів.

Під час роботи системи не спостерігалося помітних затримок у відображенні відеопотоку, що свідчить про достатню продуктивність обраної моделі для роботи в режимі, близькому до реального часу. Алгоритм object detection працює стабільно та не створює відчутного навантаження на застосунок.

Якість локалізації об’єктів оцінювалась візуально. Було виявлено, що обмежувальні рамки можуть змінювати своє положення між послідовними кадрами, що проявляється у вигляді нестабільності локалізації. Це зумовлено тим, що модель виконує детекцію незалежно для кожного кадру без використання алгоритмів відстеження.

Алгоритм демонструє здатність виявляти об’єкти при зміні їх масштабу або положення.

Робота модуля object detection не пов’язана з логікою керування рухом робота та не впливає на процес навігації.

Ця модель продемонструвала доволі високу швидкість роботи й здатність обробляти відеопотік у реальному часі на мобільних пристроях. Вона має низьке навантаження на процесор, що робить її практичною для робототехнічних застосунків. Проте її можливості певною мірою обмежені: невелика кількість навчальних даних і порівняно вузький набір класів знижують точність у складніших або нестандартних сценах.

Подальше вдосконалення системи може передбачати застосування алгоритмів відстеження об’єктів для підвищення стабільності локалізації, а також використання більш точних моделей object detection. Застосування таких моделей може покращити якість виявлення, проте супроводжується зростанням обчислювального навантаження, що є критичним фактором для роботи в режимі реального часу на обмежених апаратних ресурсах.

Загалом підвищити продуктивність обробки можна як за рахунок більш потужних апаратних засобів, так і шляхом оптимізації алгоритмів, що керують відстеженням і аналізом зображення. Використання сильнішого процесора або прискорювачів штучного інтелекту дозволило б застосовувати важчі, але

значно точніші моделі детекції. Проте в умовах мобільної платформи більш раціональним підходом є оптимізація наявних методів.

У такій архітектурі саме трекери відіграють ключову роль в цілях відстеження, оскільки вони набагато швидші за детектори й здатні підтримувати стабільне відстеження без суттєвого падіння FPS. Найефективнішим є комбінований підхід, коли легка модель детекції визначає об’єкт на початку, а надалі за рухом стежить продуктивний трекер. Така структура забезпечує баланс між швидкістю, точністю та енергоефективністю, що є важливим для робототехнічних систем із мобільним живленням.

У плані освітлення встановлений світлодіод забезпечує достатній рівень підсвічування робочої зони, однак сама камера демонструє низьку ефективність у слабо освітлених приміщеннях. Через це підсвічування доводиться вмикати практично постійно, що збільшує енергоспоживання та створює додаткове навантаження на систему живлення.

Під час роботи також спостерігались перешкоди у відеопотоці: короткочасні шуми та «ривки» зображення. Такі ефекти можуть бути наслідком нестабільного або недостатнього живлення для камери. Додатково, при активації двигунів інколи виникали стрибки напруги, що призводило до перезапуску всієї системи. Це свідчить про те, що джерело живлення не забезпечує необхідних запасів струму, або ж у схемі відсутня достатня фільтрація та розділення живлення між високоспоживчими компонентами.

У рамках аналізу роботи системи слід зазначити, що не реалізовано регулювання широтно-імпульсною модуляцією (ШІМ) для керування швидкістю та точністю руху коліс. Причиною є обмежена кількість вільних пінів на платі ESP32-CAM, через що неможливо підключити сигнали ENA та ENB, необхідні для повноцінного ШІМ-контролю. Для розширення функціональності рекомендується використання плати з більшою кількістю доступних пінів, що дозволить реалізувати плавне регулювання швидкості та підвищити точність управління мобільною платформою.

У сукупності ці фактори вказують на потребу в модернізації системи живлення (підвищення ємності, стабілізація, розділення каналів для камери та двигунів), а також у використанні камери з кращою світлочутливістю або встановленні більш ефективного модулю підсвічування.

Висновок до третього розділу

Було розроблено Android-застосунок для автономного керування мобільним робототехнічним комплексом на базі плати ESP32-CAM. Застосунок забезпечує двосторонню взаємодію з роботом: надсилає керуючі команди через UDP-протокол та отримує відеопотік з камери плати для відображення на екрані користувача. Така архітектура дозволяє як дистанційно керувати роботом у реальному часі, так і автоматизувати його рухи на основі аналізу зображень.

Для ручного керування користувач може скористатися кнопками на екрані, які відповідають руху вперед, назад, вліво та вправо. Крім того, реалізована автоматична система Tracking, яка відстежує виділений об’єкт у кадрі та коригує рух платформи або камери, щоб тримати його в центрі зображення. Цей режим дозволяє робототехнічному комплексу самостійно реагувати на рух об’єкта.

Окремо реалізовано режим Detection, який виконує детекцію та класифікацію об’єктів на кадрі за допомогою моделі SSDLite Mobilenet V2 Quantized. Детекція здійснюється в реальному часі на отриманих з камери кадрах, при цьому результати відображаються на екрані у вигляді bounding box та підпису класу об’єкта. Це дозволяє не тільки відстежувати рухомі об’єкти, але й аналізувати наявність та типи об’єктів у середовищі робота, забезпечуючи основу для подальших автономних рішень.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

4.1. Впливи шкідливих та небезпечних факторів на розробників програмного забезпечення для робототехнічних систем та засоби зменшення цих впливів

Охорона праці є невід'ємною складовою будь-якої професійної діяльності та має на меті створення безпечних та комфортних умов для працівників. Вона включає в себе комплекс правових, соціально-економічних, організаційних і технічних заходів, спрямованих на збереження життя, здоров'я та працездатності людей під час виконання ними робочих обов'язків.

Основним завданням охорони праці є зменшення або повне усунення шкідливих і небезпечних факторів, які можуть негативно впливати на фізичний та психологічний стан працівників. Це дозволяє скоротити випадки травмування та професійних захворювань, а також підвищити загальну ефективність роботи підприємства.

Крім фізичної безпеки, охорона праці також націлена на поліпшення робочого середовища та впровадження сучасних методів управління ризиками, що дозволяє не лише зменшити кількість нещасних випадків, але й підвищити продуктивність та ефективність роботи колективу.

Відповідно до Конституції України, кожна людина має право на безпечні умови праці, що не шкодять здоров'ю. Роботодавець повинен організовувати робочий процес так, що він відповідав нормам безпеки, а працівники були захищені від потенційних ризиків.

Умови праці істотно впливають на ефективність роботи, а також на фізичний та психологічний стан. Виробничі фактори можна поділити на шкідливі (поступовий вплив на здоров'я, що спричиняє професійні захворювання) та небезпечні (що призводять до травм або раптового

погіршення здоров’я за певних обставин). Шкідливі фактори можуть стати небезпечними у випадку їх значного посилення або тривалого впливу.

Робота за комп’ютером супроводжується дією різних фізичних та організаційних факторів, які можуть негативно позначатися на самопочутті та здоров’ї працівника. Ці фактори пов’язані як із особливостями технічного обладнання, так і з умовами робочого середовища. Одним із найбільш поширених чинників є нерівномірне або недостатнє освітлення, що у поєднанні з неправильно налаштованою яскравістю монітора може викликати перенапруження зору, швидку втому та головні болі. Суттєве значення має і якість повітря у приміщенні: за недостатньої вентиляції відбувається накопичення пилу, який подразнює очі та дихальні шляхи, сприяє виникненню алергічних реакцій та загального дискомфорту.

Додатково на працівника може впливати шум від периферійних пристроїв, таких як принтери, сканери, системи охолодження тощо. Хоч цей шум зазвичай не має критичного рівня, його постійна дія здатна викликати роздратування, зниження концентрації та погіршення працездатності. Важливою складовою є і загальна організація робочого місця: неправильне розташування обладнання, відсутність достатнього освітлення або неправильна висота стільця та столу значно підвищують ризик фізичного дискомфорту та перевтоми.

Для мінімізації впливу цих чинників доцільно використовувати сучасні монітори з комфортною частотою оновлення та якісним підсвічуванням, забезпечувати рівномірне освітлення робочого простору, а також підтримувати належний мікроклімат і чистоту в приміщенні. У випадках, коли рівень шуму є підвищеним, ефективним рішенням може стати використання шумопоглинальних матеріалів або зонування простору.

Окрім фізичних факторів, значний вплив на працівників ІТ-сфери мають психофізіологічні навантаження, що виникають через особливості інтелектуальної праці. Робота за комп’ютером вимагає високої концентрації уваги та постійної роботи з великими обсягами інформації. Це створює

інтенсивне когнітивне навантаження, яке з часом може призвести до стресу, розсіяності та зниження продуктивності. Робота у багатозадачному режимі додатково ускладнює ситуацію, оскільки працівнику доводиться перемикатися між різними процесами, що поступово виснажує нервову систему.

Тривале перебування у статичній позі є ще одним несприятливим фактором. Постійне сидіння без зміни положення сприяє розвитку захворювань опорно-рухового апарату, таких як остеохондроз, викривлення хребта або м’язові затиснення. Ці проблеми посилюються у разі відсутності ергономічного крісла, неправильного розміщення монітора чи клавіатури та загальної невідповідності робочого місця фізіологічним потребам людини.

Значне навантаження припадає і на органи зору. Тривала фокусування погляду на екран призводить до розвитку так званого синдрому комп’ютерного зору, який проявляється сухістю очей, розмитістю картинки, відчуттям піску в очах та головним болем. Погана якість зображення, надто низький або високий рівень яскравості, а також несвоєчасні перерви лише підсилюють ці ефекти.

Ще одним суттєвим психофізіологічним фактором є монотонність роботи. Виконання повторюваних завдань протягом тривалого часу знижує мотивацію, сприяє емоційному вигоранню та загальному виснаженню. Додатковий вплив має нераціональна організація робочого процесу, тобто відсутність регламентованих перерв, надмірні вимоги або нечітко сформульовані завдання, що створює додаткове психоемоційне навантаження на працівника.

Для зменшення негативного впливу психофізіологічних факторів важливо забезпечити раціональний режим праці та відпочинку. Регулярні перерви, короткі фізичні вправи, вправи для очей та зміна робочої пози допомагають запобігти перенапруженню. Використання ергономічних меблів знижує ризик розвитку захворювань опорно-рухового апарату, а автоматизація рутинних процесів дозволяє зменшити навантаження на пам’ять та увагу. Також корисним є чергування різних видів діяльності, що допомагає уникнути монотонності та підтримувати стабільний рівень працездатності.

Ефективна організація робочого місця розробника програмного забезпечення є важливою умовою для створення безпечного, комфортного та продуктивного середовища. Правильне розташування обладнання, відповідні меблі та належне освітлення дозволяють зменшити негативний вплив виробничих факторів та забезпечити оптимальні умови для тривалої роботи за комп’ютером.

Одним із ключових елементів є монітор, який має забезпечувати якісне зображення та можливість регулювання яскравості й контрастності для зниження навантаження на зір. Відстань між очима та екраном повинна становити приблизно 50–70 см, а верхня межа екрана знаходитися на рівні очей або трохи нижче. Персональний комп’ютер разом з клавіатурою, мишею та іншими периферійними пристроями має бути підключений до стабільного й захищеного джерела електроживлення, що мінімізує ризик пошкодження обладнання та випадків ураження електричним струмом.

Не менш важливу роль відіграють меблі. Ергономічне крісло з можливістю регулювання висоти, підтримкою поперекового відділу та зручною спинкою допомагає запобігти перенапруженню хребта. Робочий стіл повинен бути достатньо просторим для комфортного розміщення техніки та матеріалів, що використовуються у процесі роботи. Значний вплив має і освітлення: робоче місце повинно мати достатню кількість природного й штучного світла, причому важливо уникати появи відблисків на моніторі, які спричиняють додаткове навантаження на очі.

Для забезпечення безпеки обладнання доцільно використовувати джерела безперебійного живлення, які захищають техніку від перепадів напруги. У деяких випадках також застосовують засоби для зменшення статичної електрики. Крім правильної організації робочого місця, велике значення має й режим праці. Розробникам рекомендується робити регулярні короткі перерви, виконувати вправи для очей та легкі фізичні розминки, що сприяє зниженню втоми, підтриманню концентрації та загального працездатного стану.

Під час роботи за комп’ютером працівник стикається не лише з фізичними, а й з інформаційними ризиками, які можуть впливати на збереження даних та стабільність роботи обладнання. Однією з найпоширеніших загроз є втрата важливої інформації внаслідок помилок користувача, збою програмного забезпечення або пошкодження носіїв. Якщо дані не мають резервних копій, будь-який збій може призвести до значних труднощів та порушення робочого процесу.

Важливою складовою інформаційної безпеки є фізичний захист обладнання. Комп’ютери та периферійні пристрої можуть отримати ушкодження через перегрів, падіння, удари, надмірну вологість або перепади напруги в електромережі. Такі ситуації здатні не лише вивести техніку з ладу, а й спричинити втрату даних, що зберігаються на внутрішніх носіях.

Крім того, на ризики впливають технічний стан обладнання та стабільність роботи програмного забезпечення. Невчасні оновлення, неправильні налаштування або надмірне навантаження на систему можуть спричиняти зависання, некоректну роботу програм і навіть втрату незбережених файлів. Це особливо актуально в умовах інтенсивної роботи з великими обсягами інформації.

Для запобігання подібним ризикам у роботі застосовують резервне копіювання даних, використання стабілізаторів напруги, дотримання правил експлуатації та своєчасне обслуговування техніки. Крім того, важливо підтримувати порядок на робочому місці та забезпечувати правильні умови зберігання обладнання, щоб уникнути випадкових пошкоджень. Комплексний підхід до захисту інформації та техніки дозволяє забезпечити стабільну та безпечну роботу з комп’ютерними системами.

Електротравми належать до найпоширеніших небезпечних факторів, що можуть виникати у виробничому середовищі. Ступінь їх тяжкості залежить від сили та напруги струму, частоти, тривалості впливу, а також від фізичного стану людини. Ураження може проявлятися як у вигляді локальних пошкоджень, так і у формі загальних реакцій організму. До локальних

електротравм зазвичай належать опіки, характерні сліди проходження струму на шкірі та механічні ушкодження, які виникають через різкий неконтрольований рух тіла або падіння. Загальні електротравми проявляються серйознішими наслідками: від судом та втрати свідомості до критичних порушень життєвих функцій.

За ступенем ураження електричним струмом виділяють чотири рівні. Найлегший супроводжується болючими м’язовими спазмами без втрати свідомості. Наступний рівень характеризується судомами та короткочасною втратою свідомості при збереженому диханні й роботі серця. Третій ступінь є значно небезпечнішим, оскільки струм викликає серйозні порушення дихання та серцевої діяльності. Найважчий, четвертий ступінь, фактично відповідає стану клінічної смерті з повним припиненням дихання та серцебиття.

Дія електричного струму на організм може проявлятися у тепловій, хімічній, механічній та біологічній формах. Теплова дія спричиняє опіки тканин, хімічна змінює склад крові та біохімічні процеси в організмі, механічна веде до розривів тканин, а біологічна безпосередньо впливає на роботу нервової системи й серця, порушуючи їх нормальне функціонування. Сукупність цих ефектів визначає небезпеку електротравм і підкреслює важливість дотримання правил електробезпеки у виробничих умовах.

При роботі з електронними схемами, сенсорами та компонентами робототехнічного комплексу існує ризик короткого замикання та пошкодження обладнання. Некоректне підключення або помилки при складанні схем можуть не лише вивести техніку з ладу, а й створити потенційну небезпеку для користувача. Тому дуже важливо дотримуватися правил підключення та перевіряти всі контакти перед подачею живлення.

Іншою значущою загрозою є перегрів компонентів або неправильне живлення. Перевищення допустимих параметрів струму чи напруги може призвести до пошкодження плати, сенсорів або інших електронних елементів. Для запобігання цьому слід контролювати потужність джерел живлення та

забезпечувати належне охолодження компонентів, особливо у випадках тривалого функціонування системи.

Статична електрика також становить серйозну небезпеку для чутливої електроніки. Накопичений на корпусі або руках заряд може раптово перейти на компоненти, пошкоджуючи їх або порушуючи роботу. Використання антистатичних засобів і заземлення обладнання дозволяє мінімізувати ці ризики та зберегти працездатність системи.

Крім електричних небезпек, важливу роль відіграють механічні пошкодження. Під час монтажу або налаштування робототехнічних компонентів можна випадково пошкодити датчики, плату або інші крихкі деталі. Такий вплив здатний знизити ефективність роботи системи та створити додаткові проблеми для експлуатації.

4.2. Охорона навколишнього середовища

Забезпечення екологічної безпеки та охорона навколишнього середовища є важливою складовою діяльності будь-якого підприємства, зокрема тих, що впроваджують інноваційні технології, такі як системи штучного інтелекту та робототехнічні комплекси. У контексті розробки, експлуатації та обслуговування робототехніки особливу увагу приділяють мінімізації шкідливого впливу на довкілля, раціональному використанню ресурсів та оптимізації енергоспоживання. Робототехнічні системи, як і комп’ютерне обладнання, споживають значні обсяги електроенергії, створюють теплове навантаження та генерують відходи у вигляді зношених електронних компонентів, моторів, сенсорів або акумуляторів, що потребує правильної утилізації.

Під час розробки робототехнічних комплексів важливо застосовувати енергоефективні двигуни, актуатори та електронні модулі, які дозволяють зменшити витрати електроенергії без зниження продуктивності. Додатковим напрямом є використання відновлюваних джерел енергії, наприклад:

застосування сонячних панелей для заряджання автономних роботів або впровадження енергоощадних алгоритмів керування. Такі рішення сприяють зниженню навантаження на електромережу та допомагають скоротити викиди парникових газів, роблячи виробничі процеси більш екологічними.

Разом з тим робототехніка створює і специфічні екологічні виклики. Неправильне поводження з відпрацьованими літій-іонними акумуляторами, сервомоторами, оптичними системами чи сенсорами може призвести до забруднення ґрунту та води токсичними речовинами. Тому підприємства повинні організовувати системи збору, сортування та переробки електронних і механічних відходів, співпрацювати із сертифікованими організаціями та суворо дотримуватися норм екологічного законодавства. Завчасний контроль стану робототехнічних елементів, своєчасна заміна зношених частин та проектування модульних роботів, придатних до ремонту, дозволяють істотно зменшити кількість утворених відходів і підвищити загальну екологічну стабільність системи.

У більш широкому контексті екологічна безпека сучасних підприємств, що працюють із нейромережами, штучним інтелектом чи робототехнікою, включає також контроль якості повітря та мікроклімату приміщень, де розміщене обладнання, зниження рівня тепловиділення, моніторинг електромагнітного фону та забезпечення належної вентиляції. Ефективні фільтраційні системи, рекуперація тепла, автоматизовані засоби контролю температури й вологості не лише зменшують вплив на навколишнє середовище, а й підвищують комфорт та безпеку для персоналу.

Важливою складовою природоохоронної діяльності є відповідальне поводження з електронними відходами. Підприємства створюють спеціальні майданчики для тимчасового зберігання списаного обладнання, використовують герметичні контейнери для компонентів, що можуть бути небезпечними, після чого передають їх для переробки професійним компаніям. Такий підхід забезпечує відповідність екологічним стандартам та істотно зменшує ризики забруднення навколишнього середовища.

Узагальнюючим документом, що фіксує всі ключові екологічні характеристики діяльності підприємства, є екологічний паспорт. Він містить інформацію про технологічні процеси, рівень шкідливих викидів, утворення та утилізацію відходів, системи очищення повітря та води, а також загальні природоохоронні заходи. У разі експлуатації робототехнічних або нейромережевих систем екологічний паспорт відіграє особливо важливу роль, оскільки дозволяє комплексно контролювати вплив обладнання на довкілля та забезпечувати відповідність чинним нормам.

Не менш важливою частиною забезпечення екологічної безпеки є організаційні заходи: оформлення документів, ведення звітності, інвентаризація джерел можливого забруднення та співпраця з державними екологічними структурами. Регулярний контроль параметрів навколишнього середовища, проведення перевірок стану ґрунтів та води, моніторинг викидів допомагають своєчасно виявляти проблеми та вживати коригувальних заходів.

Важливу роль відіграє і формування екологічної свідомості працівників. Проведення навчань, інформаційних заходів, запровадження корпоративних програм сталого розвитку та залучення персоналу до природоохоронних ініціатив сприяють формуванню відповідального ставлення до навколишнього середовища на всіх рівнях підприємства.

Таким чином, системний підхід до екологічної безпеки у сфері робототехнічних проєктів та сучасних інформаційних технологій дозволяє досягти балансу між високою продуктивністю, безпекою персоналу та збереженням навколишнього середовища. Комплексне впровадження технічних, організаційних та управлінських заходів створює умови для сталого розвитку й мінімізації негативного екологічного впливу.

Висновок до четвертого розділу

У цьому розділі було розглянуто умови праці розробників програмного забезпечення та робототехнічних систем, визначено основні шкідливі й

небезпечні фактори та оцінено їхній вплив на здоров’я та працездатність працівників. Проаналізовано фізичні, психофізіологічні та інформаційні ризики, що виникають під час роботи з комп’ютерним і робототехнічним обладнанням, а також виявлено їх взаємозв’язок із якістю організації робочого місця та режимом праці.

Особливу увагу приділено засобам зниження цих ризиків. Встановлено, що використання ергономічних меблів, якісного освітлення, сучасних моніторів, систем вентиляції, антистатичного захисту, стабілізаторів напруги та засобів контролю мікроклімату дозволяє суттєво зменшити вплив шкідливих факторів. Важливими також є раціональний режим праці, регулярні перерви та автоматизація рутинних процесів, що допомагають знизити психофізіологічне навантаження.

Додатково було проаналізовано екологічні аспекти діяльності, пов’язані з розробкою та експлуатацією робототехніки. Показано, що енергоефективність, правильна утилізація електронних відходів, контроль акумуляторів і технічних компонентів, зниження теплового навантаження та організація системи екологічного моніторингу є ключовими напрямами забезпечення екологічної безпеки. Екологічний паспорт підприємства, відповідальне поводження з відходами та залучення персоналу до природоохоронних ініціатив сприяють зменшенню негативного впливу на довкілля.

Комплексний підхід до охорони праці та екологічної безпеки забезпечує стабільну, ефективну й безпечну роботу персоналу, підвищує надійність робототехнічних систем та сприяє сталому розвитку підприємства.

ВИСНОВКИ

У даній магістерській роботі розроблено Android-застосунок для автономного керування робототехнічним комплексом на базі плати ESP32-CAM. Проведено аналіз сучасних методів відстеження об’єктів та детекції, вибрано оптимальні алгоритми для реалізації режимів Tracking та Detection, а також визначено ефективні підходи для обробки відеопотоку в реальному часі на мобільних пристроях. Розглянуто особливості різних трекерів (MedianFlow, KCF, TLD, CSRT, MOSSE, MIL) та оцінено їхні переваги і недоліки у контексті продуктивності та точності відстеження.

Результатом роботи стало створення функціональної системи, що інтегрує інтерфейс користувача, модуль обробки зображень та механізм відправки UDP-команд для управління рухом робота та сервоприводом камери. Система дозволяє користувачу керувати роботом вручну або в автоматичному режимі, коли робот самостійно відстежує об’єкт у полі зору камери, намагаючись утримати його в центрі кадру. Для режиму Detection використано модель SSDLite MobileNet V2 Quantized, що забезпечує швидке та ефективне виявлення об’єктів з низьким навантаженням на мобільний процесор.

Під час тестування системи було підтверджено її здатність обробляти відеопотік у реальному часі, підтримуючи стабільне відстеження та детекцію об’єктів. Найефективнішими трекерами виявилися MedianFlow, KCF та TLD, які забезпечують баланс між швидкістю обробки та точністю відстеження. Виявлено також обмеження у роботі при поганому освітленні, нестабільному живленні та високій навантаженості на камеру, що потребує застосування світлодіодного підсвічування та оптимізації алгоритмів. Отримана система показала ефективність і стабільність роботи, що робить її придатною для використання у сфері мобільної робототехніки та комп’ютерного зору на обмежених апаратних платформах.

Розглянуті питання охорони праці та навколишнього середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alsakka F., Assaf S., El-Chami I., Al-Hussein M. Computer vision applications in offsite construction / F. Alsakka, S. Assaf, I. El-Chami, M. Al-Hussein // Automation in Construction. – 2023. – Vol. 154. – P. 104980.
2. Клімов О. П., Ісаков О. В., Мартиненко М. М. Дрони FPV з машинним зором // Social ways of training specialists in the social sphere and inclusive education: Abstracts of the XIII International Scientific and Practical Conference (April 01–03, 2024, Prague, Czech Republic). – Prague: European Conference, 2024. – С. 334.
3. Szeliski R. Computer Vision: Algorithms and Applications / R. Szeliski. — 2nd ed. — Cham: Springer, 2022. — С. 3–8.
4. Gonzalez R. C., Woods R. E. Digital Image Processing / R. C. Gonzalez, R. E. Woods. – 2nd ed. – Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2002. – С. 5.
5. Іванов А. І. Всебічний огляд історії та методів комп’ютерного зору. — Телекомунікаційні та інформаційні технології. — 2025. — № 1 (86). — С. 178. — ISSN 2412-4338. — DOI: 10.31673/2412-4338.2025.011767.
6. Alzubaidi L., Zhang J., Humaidi A. J., Al-Dujaili A., Duan Y., Al-Shamma O., Santamaría J., Fadhel M. A., Al-Amidie M., Farhan L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions // Journal of Big Data. – 2021. – Vol. 8, Article 53. – P. 13–20. – DOI: https://doi.org/10.1186/s40537-021-00444-8
7. Singh A., Kalaichelvi V., Karthikeyan R. A survey on vision guided robotic systems with intelligent control strategies for autonomous tasks / A. Singh, V. Kalaichelvi, R. Karthikeyan // Cogent Engineering. – 2022. – Vol. 9, № 1. – Article 2050020. – DOI: https://doi.org/10.1080/23311916.2022.2050020
8. AZoOptics. Infrared Imaging: How It Works and Its Applications [Електронний ресурс]. – Режим доступу: https://www.azooptics.com/Article.aspx?ArticleID=2599

9. 3D reconstruction from stereo camera images [Електронний ресурс] // ResearchGate. – Режим доступу: https://www.researchgate.net/figure/D-reconstruction-from-stereo-camera-images_fig6_318452089
10. Lucid Vision Labs. Helios2+ Time-of-Flight (ToF) IP67 3D Camera [Електронний ресурс]. – Режим доступу: https://thinklucid.com/product/helios2plus-time-of-flight-tof-ip67-3d-camera/
11. Aerial Precision. What is LiDAR? [Електронний ресурс]. – Режим доступу: https://www.aerial-precision.com/blogs/blog/what-is-lidar
12. Feng X., Jiang Y., Yang X., Du M., Li X. Computer vision algorithms and hardware implementations: A survey / X. Feng, Y. Jiang, X. Yang, M. Du, X. Li // Integration, the VLSI Journal. – 2023. – Vol. [номер тома]. – С. 315–316.
13. Zekovic A. L. Hardware and Software of Computer Vision IoT Solutions Leveraging Raspberry Pi Boards / A. L. Zekovic // Telfor Journal. – 2025. – Vol. 17, No. 1.
14. Проноза Я. Сучасні засоби та бібліотеки для обробки і трансформації зображень / Я. Проноза. – Житомир: Житомирський державний університет імені Івана Франка, 2024. – [Електронний ресурс]. – С. 196–198.
15. Robotko S., Topalov A., Susak O., Gerasin O. Enabling Cloud-based Data Analysis for Analog Metal Detectors Using Microcomputer Systems. The 14th International Conference on Advanced Computer Information Technologies (ACIT'2024). University of South Bohemia in České Budějovice, pp. 670 – 673. DOI: 10.1109/ACIT62333.2024.10712463 https://ieeexplore.ieee.org/document/10712463