Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

СИСТЕМА КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ЦІЛЕЙ БЕЗПІЛОТНИХ АПАРАТІВ

Виконав студент 2 курсу, групи 2КІ-24м
спеціальності 123 — Комп'ютерна інженерія
Довгань Д.С.
(прізвище та ініціали)
Керівник к.т.н. доц. каф. ОТ
Томчук М.А.
(прізвище та ініціали)
"12" грудня 2025р.
Опонент к.т.н, доцент каф. ПЗ
Ткаченко О.М.
(прізвище та ініціали)
"12" грудня 2025р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«16» 12 2025 р.

ВНТУ 2025

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
"25" вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту **Довганю Дмитру Сергійовичу**

1 Тема роботи «Система комп'ютерного зору для розпізнавання цілей безпілотних апаратів» керівник роботи к.т.н., доцент каф. ОТ Томчук Микола Антонович, затверджено наказом вищого навчального закладу від 24.09.2025 року № 313

2 Строк подання студентом роботи 10.12.2025 р.

3 Вихідні дані до роботи: розробка блоку комп'ютерного зору для автоматичного розпізнавання цілей у реальному часі та інтеграції з FPV-дроном, апаратна платформа — Raspberry Pi 5, програмні засоби: OpenCV, YOLO, Python.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз та дослідження методів розпізнавання і відстеження об'єктів у відеопотоці БПЛА, аналіз технологій та вибір архітектури системи комп'ютерного зору, розробка системи комп'ютерного зору для розпізнавання цілей безпілотних апаратів, експериментальне дослідження та оцінка ефективності системи, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язков
креслень) — блок-схема алгоритму розпізнавання та трекінгу цілей
Перелік ілюстративного матеріалу — фото прототипа модуля комп'ютерног
зору.

6 Консультанти розділів роботи представлені у таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н. доц. каф. ОТ Томчук М.А.		12.10.25
5	к.т.н., доц. каф. ЕПВМ Адлер О.О.		
Нормо контроль	асист. каф. ОТ Швець Сергій Ілліч		

7 Дата видачі завдання 25.09.2025

8 Календарний план виконання МКР наведений у таблиці 2.

Таблиця 2 — Календарний план

з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	10.09.2025	
2	Огляд сучасних технологій комп'ютерного зору для безпілотних систем	28.09.2025	
3	Аналіз методів розпізнавання і відстеження об'єктів у відеопотоці	18.10.2025	
4	Розробка архітектури та програмної реалізації системи комп'ютерного зору	01.11.2025	
5	Розрахунок економічної частини	09.11.2025	
6	Попередній захист	11.11.2025	
7	Оформлення пояснювальної записки	15.11.2025	
8	Виконання магістерської кваліфікаційної роботи	28.11.2025	
9	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	11.12.2025	
10	Підписи супроводжувальних документів у керівника, опонента, нормо-контролера	12.12.2025	
11	Перевірка «Антиплагіат»	13.12.2025	

Студент

Керівник

(підпис) Довгань Д.С.
(прізвище та ініціали)

(підпис) Томчук М.А.
(прізвище та ініціали)

АНОТАЦІЯ

УДК 004.932

Довгань Д. С. Система комп’ютерного зору для розпізнавання цілей безпілотних апаратів. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна інженерія, освітня програма — Комп’ютерна інженерія. Вінниця: ВНТУ, 2025. 136 с.

На укр. Мові. Бібліогр.:25назв; рис.:19; табл.:12.

У магістерській кваліфікаційній роботі розроблено систему комп’ютерного зору для розпізнавання та відстеження цілей безпілотних літальних апаратів у реальному часі. Досліджено сучасні методи розпізнавання та відстеження об’єктів у відеопотоці БПЛА та створено оптимізовану архітектуру, придатну для роботи на вбудованих пристроях з обмеженими ресурсами. Основу системи становлять нейронні мережі сімейства YOLO для високоточого розпізнавання та трекер MOSSE для швидкого відстеження об’єктів з мінімальним навантаженням на процесор.

У роботі застосовано бібліотеки OpenCV для обробки зображень, NumPy для обчислень, PyMAVLink для зв’язку між Raspberry Pi та польотним контролером, а також рушій NCNN для прискореного виконання моделей.

Систему реалізовано на платформі Raspberry Pi 5 із використанням мови Python, що забезпечує мобільність і просту інтеграцію з дронами. Проведено тестування швидкодії та точності, виконано порівняння з аналогічними рішеннями, визначено переваги та шляхи подальшого вдосконалення системи.

Ключові слова**:** комп’ютерний зір, безпілотний літальний апарат, розпізнавання об’єктів, YOLO, OpenCV, трекінг, Raspberry Pi.

ABSTRACT

UDC 004.932

Dovhan D. S. Computer Vision System for Target Recognition of Unmanned Aerial Vehicles. Master’s Thesis in Specialty 123 - Computer Engineering, Educational Program - Computer Engineering. Vinnytsia: VNTU, 2025. 136 pages.

In Ukrainian. Bibliography: 25, figures: 19, listings:12.

This Master's qualification thesis presents the development of a real-time computer vision system for Unmanned Aerial Vehicle (UAV) target recognition and tracking. The research investigates modern methods for object detection and tracking in UAV video streams and creates an optimized architecture suitable for operation on resource-constrained embedded devices. The system's core is based on the YOLO family of neural networks, which ensures high recognition accuracy, and the MOSSE tracker, which guarantees high-speed performance with minimal CPU load.

The work utilizes the OpenCV library for image processing, NumPy for computations, PyMAVLink for communication between the Raspberry Pi and the flight controller, and the NCNN engine for accelerated model inference.

The system is implemented on the Raspberry Pi 5 platform using the Python programming language, ensuring mobility and straightforward integration with unmanned aerial vehicles. Performance and accuracy testing was conducted, a comparison with similar solutions was performed, and the system's advantages as well as potential directions for further improvement were identified.

Keywords**:** computer vision, unmanned aerial vehicle, object recognition, YOLO, OpenCV, tracking, Raspberry Pi.

ЗМІСТ

ВСТУП

Актуальність теми зумовлена необхідністю підвищення рівня автономності та точності безпілотних літальних апаратів. Одним із шляхів вирішення цього завдання є розробка інтелектуальних систем сприйняття, які здатні самостійно аналізувати навколишнє середовище та приймати рішення щодо керування БПЛА в реальному часі. Сучасні умови застосування безпілотних систем вимагають високої надійності роботи навіть за наявності радіоперешкод і ситуацій радіозатінення. Розробка ефективних алгоритмів комп'ютерного зору дозволяє підвищити стійкість апаратів до зовнішніх впливів та зменшити залежність від каналів зв'язку. Це особливо актуально для України, де безпілотні технології активно використовуються у оборонній сфері.

Мета роботи полягає у розширенні функціональних можливостей безпілотних літальних апаратів шляхом розробки та експериментальної оцінки програмно-апаратної системи комп'ютерного зору для автоматичного виявлення та відстеження об'єктів у відеопотоці БПЛА. Система забезпечує високу точність розпізнавання у реальному часі та інтеграцію з системами керування дроном**.**

Для досягнення поставленої мети необхідно вирішити такі **задачі**, зокрема:

— провести аналіз сучасних методів детекції та трекінгу об'єктів у відеопотоці на основі технологій глибокого навчання;

— розробити архітектуру системи, що забезпечує поєднання покадрового розпізнавання з багатокадровим трекінгом;

— створити прототип системи комп'ютерного зору та інтегрувати його з модулем керування БПЛА;

— провести експериментальні дослідження точності, швидкодії та стабільності роботи системи;

— визначити напрямки подальшої оптимізації алгоритмів і засобів реалізації.

Об’єктом є процес розпізнавання та відстеження об’єктів у відеопотоці за допомогою системи комп’ютерного зору для БПЛА.

Предмет дослідження — методи, алгоритми та програмно-апаратні засоби детекції, класифікації та трекінгу об’єктів у реальному часі, а також способи їх інтеграції з системами керування БПЛА.

Новизна роботи полягає у вдосконаленні методу розпізнавання цілей безпілотних літальних апаратів шляхом поєднння сучасних архітектур глибокого навчання для покадрової детекції та багатокадрового трекінгу об’єктів. Такий підхід забезпечує підвищення точності, стійкості та швидкодії системи в реальному часі навіть за умов обмежених апаратних ресурсів. Зокрема, новизна проявляється в:

— розробці та реалізації оптимізованого алгоритму трекінгу рухомих цілей із використанням багатокадрового аналізу , який відрізняється оптимізацією під обмежені обчислювальні ресурси і забезпечує швидке та точне відстеження об’єктів у реальному часі;

— адаптації архітектури системи до обмежених обчислювальних ресурсів вбудованих платформ, таких як Raspberry Pi ;

— інтеграції модулів комп’ютерного зору з польотним контролером для реалізації автономних режимів ураження цілі.

Практичне значення отриманих результатів полягає у можливості використання розробленої системи для створення безпілотних платформ з підтримкою автономного спостереження та супроводу цілей, а також систем моніторингу в оборонних і безпекових сферах. Система функціонує у режимі реального часу і може бути використана як прототип для подальшого впровадження. Результати дослідження також можуть бути адаптовані для інших систем комп'ютерного зору, що підвищує оперативність та точність обробки відеопотоку у реальних умовах.

Апробація результатів наукової роботи: результатів наукової роботи було проведено на науковій конференції Актуальні проблеми бойового застосування, експлуатації і ремонту зразків озброєння та військової

техніки (2025).

Матеріали роботи доповідались та опубліковувались [1]:

Д. С. Довгань, О. Д. Азаров, Томчук М. А. / Оптимізація систем комп’ютерного зору для використання із обмеженими апаратними платформами // Тези доповіді. Актуальні проблеми бойового застосування, експлуатації і ремонту зразків озброєння та військової техніки (2025). Вінниця 2025 р. Режим доступу:https://conferences.vntu.edu.ua/index.php/apozbt/apozbt2025/paper/view/26170

Також результати наукової роботи було проведено на науковій конференції Молодь в науці:дослідження, проблеми, перспективи(МН-2026).

Матеріали роботи доповідались та опубліковувались [2]:

Д. С. Довгань, О. Д. Азаров, Томчук М. А. / Інтеграція системи комп’ютерного зору на основі OpenCV з автопілотом FPV-дрона через протокол MSP // Тези доповіді. Молодь в науці:дослідження, проблеми, перспективи(МН-2026). Вінниця 2025 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26608

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ МЕТОДІВ РОЗПІЗНАВАННЯ І ВІДСТЕЖЕННЯ ОБ'ЄКТІВ У ВІДЕОПОТОЦІ З БПЛА

1.1 Актуальність створення систем комп'ютерного зору для безпілотних літальних апаратів

Стрімкий розвиток технологій штучного інтелекту, цифрової обробки сигналів і систем автоматизації започаткував новий етап еволюції безпілотних літальних апаратів (БПЛА). Якщо раніше дрони виконували переважно функції дистанційного спостереження під контролем оператора, то сьогодні вони дедалі частіше виступають як автономні інтелектуальні системи, здатні самостійно аналізувати навколишнє середовище, розпізнавати об'єкти та ухвалювати рішення в реальному часі.

Центральним елементом таких систем є комп'ютерний зір — технологічна основа, що забезпечує здатність обчислювальної системи «бачити» та інтерпретувати візуальну інформацію, подібно до людського сприйняття. Саме комп'ютерний зір надає безпілотним платформам можливість виконувати складні завдання навігації, виявлення, розпізнавання та супроводу об'єктів без постійного втручання оператора [3].

Актуальність розроблення таких систем зумовлена необхідністю підвищення ефективності, точності та автономності БПЛА під час виконання завдань у різноманітних умовах. У цивільному секторі це стосується моніторингу навколишнього середовища, сільськогосподарського аналізу, технічного обстеження інфраструктури, енергетики, рятувальних операцій тощо. Проте найбільш гостро потреба у високоточних системах розпізнавання та відстеження об'єктів проявляється в оборонній сфері — особливо в умовах збройної агресії проти України.

Під час сучасних бойових дій БПЛА стали одним із ключових елементів розвідувально-ударних комплексів. Завдяки системам комп'ютерного зору дрони здатні виявляти техніку противника, визначати її координати, супроводжувати рухомі цілі та передавати точні дані для наведення вогневих

засобів. Ефективність таких систем безпосередньо впливає на бойову стійкість підрозділів, точність ураження цілей, збереження життя військових і мінімізацію побічних втрат серед цивільного населення.

Водночас, складність сучасного бойового середовища — динамічні об’єкти, зміни освітлення, дим, пил, радіоелектроні перешкоди — вимагають нових підходів до побудови адаптивних систем комп’ютерного зору, здатних функціонувати в умовах неповних, зашумлених або спотворених даних. Саме тому дослідження методів детекції та трекінгу об’єктів у відеопотоці набуває стратегічного значення не лише для наукової спільноти, але й для оборонно-промислового комплексу України.

Таким чином, актуальність створення систем комп’ютерного зору для безпілотних літальних апаратів визначається сукупністю таких чинників:

— глобальними тенденціями розвитку автономних інтелектуальних систем;

— потребами обороноздатності України в умовах воєнного стану;

— зростанням ролі цифрових технологій у сфері безпеки та моніторингу;

— науково-технічним викликом створення універсальних алгоритмів розпізнавання в реальному часі.

Отже, розробка й удосконалення систем комп’ютерного зору для БПЛА є не лише науково обґрунтованим, а й стратегічно необхідним завданням, що сприяє зміцненню національної безпеки та розвитку інтелектуальних технологій майбутнього.

1.2 Класифікація безпілотних літальних апаратів та напрями їх застосування

Існує велика кількість різновидів літальних апаратів, тому і класифікаційних ознак також багато [4]. Сучасні БПЛА класифікують за такими основними ознаками: призначення, тип системи керування, правила польоту, тип крила, радіус дії, висота, функціональне призначення.

За призначенням БПЛА поділяються на:

— військові — спостережні, розвідувальні, ударні, розвідувально-ударні, бомбардувальні, винищувальні, радіотрансляційні, БПЛА РЕБ, транспортні, БПЛА-мішені, багатоцільові;

— цивільні — моніторинг об'єктів, відеоспостереження, картографування, пошук корисних копалин, доставка вантажів, ретрансляція сигналів.

За типом крила розрізняють два основних типи:

— БПЛА фіксованого крила — мають високу швидкість і тривалість польоту, приклад продемонстровано на рисунку 1.1;

— БПЛА обертового крила — здатні до зависання і високої маневреності.

Рисунок 1.1 — Військовий БПЛА

Тактичні БПЛА зазвичай належать до апаратів фіксованого крила і призначені для виконання бойових завдань на обмеженій території.

Мультироторні БПЛА (квадрокоптери, гексакоптери, октокоптери) належать до апаратів обертового крила і особливо ефективні для розвідки та ураження в умовах міста.

За методом керування виокремлюють:

— дистанційно-пілотований спосіб — ручне або автоматизоване керування оператором;

— автоматичний спосіб — керування автопілотом за заданою траєкторією.

За правилами польотів БПЛА поділяються на візуальні, приладові та візуально-приладові, що визначає умови їх експлуатації.

Візуальні — політ здійснюється в межах прямої видимості оператора у світлий час доби. Оператор постійно бачить БПЛА і може візуально контролювати його положення в просторі.

Приладові — політ здійснюється поза межами прямої видимості оператора, в тому числі вночі. Керування відбувається за допомогою інструментальних засобів (відеокамери, радари, телеметричні системи).

Візуально-приладові — комбінований режим, коли окремі етапи польоту (зліт, посадка) виконуються за візуальними правилами, а основна частина маршруту за приладовими.

Ця класифікація демонструє різноманітність безпілотних апаратів та обумовлює вибір конкретного типу БПЛА для вирішення певних завдань.

1.3 Основні принципи роботи систем комп’ютерного зору

Людина має здатність миттєво сприймати та інтерпретувати навколишній світ, визначаючи форму, розмір, колір, текстуру та просторове розташування об'єктів. Цей процес, здається нам природним, а для машини є одним із найскладніших обчислювальних задач [5]. Комп'ютерний зір (Computer Vision, CV) — це галузь штучного інтелекту, що має на меті навчити машини «бачити» та розуміти візуальну інформацію подібно до людини. Приклад відміност відмінності сприйняття однакових зображень людиною та комп’ютером можна наглядно побачити на рисунку 1.2 . Ця різниця демонструє, що комп’ютер оперує числовими ознаками й алгоритмами, тоді як людина сприймає зображення цілісно та контекстно.

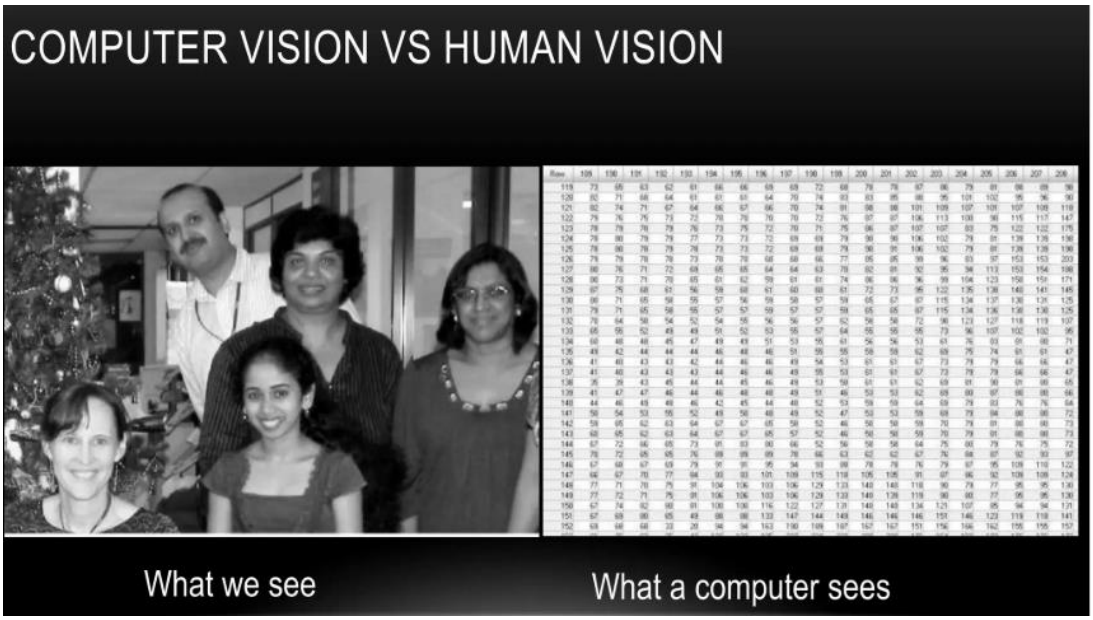

Рисунок 1.2 — Порівняння людського та машинного сприйняття зображення

Від пікселів до семантики рівнія обробки зображень. Роботу систем комп'ютерного зору можна поділити на кілька послідовних рівнів.

Низькорівнева обробка, на цьому етапі відбувається попередня обробка сирих даних (пікселів). Застосовуються такі операції, як фільтрація шуму, покращення контрастності, виявлення границь (edge detection) за допомогою фільтрів Собеля або Кенні. Метою є покращення якості зображення та виділення ключових структур без їх семантичної інтерпретації.

Середньорівнева обробка, тут відбувається сегментація зображення — розділення на області, що, ймовірно, належать різним об'єктам. Виділяються контури, знаходяться особливі точки (keypoints) та дескриптори (наприклад, SIFT, ORB), які інваріантні до масштабу та повороту. Це дозволяє системі описати зображення у вигляді набору характерних ознак.

Високорівнева обробка, це рівень семантичного розуміння. Система аналізує отримані ознаки, виконує класифікацію об'єктів (що це?), детекцію (де це знаходиться?) та інтерпретацію сцени (як об'єкти взаємодіють?). Дослідники комп’ютерного зору розробляють математичні моделі, що імітують процес сприйняття.

Ці методи базуються на знаннях з фізики (оптики) та геометрії, які моделюють:

— проєктивну геометрію, як тривимірні об'єкти проєктуються на двовимірну площину сенсора камери;

— фотометрію, як світло взаємодіє з поверхнями об'єктів, формуючи їх яскравість та колір на зображенні.

Сьогодні, завдяки методам стереобачення (stereo vision) та структурованого світла, можливо з високою точністю будувати часткові 3D-моделі середовищ (хмари точок). Сучасні алгоритми, такі як SLAM (Simultaneous Localization and Mapping), дозволяють не лише будувати карту невідомого простору, але й одночасно визначати в ньому положення самого пристрою [6].

Історично задачі CV вирішувалися класичними алгоритмами, які покладалися на ручно crafted ознаки (hand-crafted features), такі як HOG (Histogram of Oriented Gradients) для детекції пішоходів або SIFT для пошуку відповідностей між зображеннями. Незважаючи на ефективність у певних умовах, ці методи були крихкими та чутливими до змін освітлення, масштабу та ракурсу.

Переломним моментом стала поява глибокого навчання, зокрема згорткових нейронних мереж (Convolutional Neural Networks, CNN) [7]. На відміну від класичних підходів, CNN здатні автоматично вивчати ієрархічні представлення ознак безпосередньо з даних:

— нижчі шари мережі виявляють прості елементи, грані, кутки, текстури;

— вищі шари комбінують їх у більш складні структури, частини обличчя, колеса автомобіля, а потім і цілі об'єкти.

Це дозволило радикально підвищити точність у таких завданнях, як класифікація зображень (змагання ImageNet), семантична сегментація та детекція об'єктів.

Ключові задачі комп'ютерного зору та їх практична реалізація, окрім розпізнавання, існує цілий спектр задач, що вирішуються системами CV.

Детекція об'єктів (Object Detection) — це пошук та локалізація всіх об'єктів певних класів на зображенні з визначенням їхніх обмежувальних рамок (bounding boxes). Сучасні детектори, такі як YOLO (You Only Look Once) або Faster R-CNN, досягають високої швидкості та точності [8].

Сегментація екземплярів (Instance Segmentation) — це більш складна задача, ніж детекція, що передбачає не тільки знаходження об'єкта, але й побудову піксельної маски для кожного його екземпляра (наприклад, Mask R-CNN).

Трекінг об'єктів (Object Tracking) — це відстеження траєкторії об'єкта в послідовності кадрів відео. Алгоритми на кшталт SORT та DeepSORT дозволяють стійко супроводжувати цілі навіть при тимчасових перекриттях [9].

Оцінка пози (Pose Estimation) — це визначення просторового положення та орієнтації об'єкта (наприклад, скелета людини) або камери.

Особливе місце серед прикладних задач комп’ютерного зору займає розпізнавання цілей (target recognition), що поєднує етапи детекції, класифікації та трекінгу об’єктів у реальному часі [10]. Такі системи широко застосовуються в безпілотних літальних апаратах для виявлення та супроводу рухомих об’єктів на місцевості. Основна мета полягає у визначенні положення цілі, її форми, класу (наприклад, транспортний засіб, людина, техніка) та відстеженні траєкторії руху. На рисунку 1.3 наведено приклад того, як система комп’ютерного зору виділяє ціль — у цьому випадку танк — шляхом формування контурів та обмежувальних рамок об’єкта.

Рисунок 1.3 — Візуалізація процесу розпізнавання цілі системою комп’ютерного зору

Основним недоліком навіть сучасних алгоритмів комп'ютерного зору залишається їхня вразливість до умов, що не представлені в тренувальних даних. Незначні зміни, такі як адверсійні атаки (спеціально підібрані шумові паттерни), різке змінення освітлення або незвичайний ракурс, можуть призвести до катастрофічних помилок. Крім того, системи потребують великих обсягів розмічених даних для навчання, а їхня робота часто залишається «чорною скринькою», що ускладнює інтерпретацію прийнятих рішень.

Таким чином, комп'ютерний зір — це комплексна дисципліна, що поєднує методики з обробки сигналів, машинного навчання, геометрії та фізики. Від виділення границь на зображенні до складного семантичного аналізу сцени, системи CV проходять складний шлях трансформації пікселів у знання. Розвиток глибокого навчання відкрив нову еру в цій галузі, однак завдання створення системи, що володіє гнучкістю, надійністю та здоровими глуздами людини, залишається відкритою та актуальною для дослідників.

1.4 Архітектура та принципи функціонування штучних нейронних мереж

Штучні нейронні мережі виникли як спроба відтворити принципи роботи людського мозку, який здатний виконувати складні, нелінійні й паралельні обчислення з високою ефективністю.

Біологічні нейрони передають електричні імпульси через аксони, формуючи тисячі зв’язків (синапсів) з іншими клітинами. Синапси можуть підсилювати або блокувати сигнали за допомогою нейромедіаторів. Саме ці зв’язки визначають “ваги” інформації, що згодом стало основою моделі ШНМ. Мозок обробляє інформацію паралельно за участю приблизно 10^{11} нейронів, тому він здатен швидко виконувати завдання розпізнавання образів, мови чи руху. На рис. 1.4 подано структурну модель біологічного нейрона, що демонструє складність взаємодії між дендритами, аксональним проходженням сигналу та синаптичними зв’язками. Багатошарова організація кори головного мозку, наведена на рис. 1.5, відображає принцип ієрархічної обробки інформації, у якій простіші ознаки комбінуються у складніші.

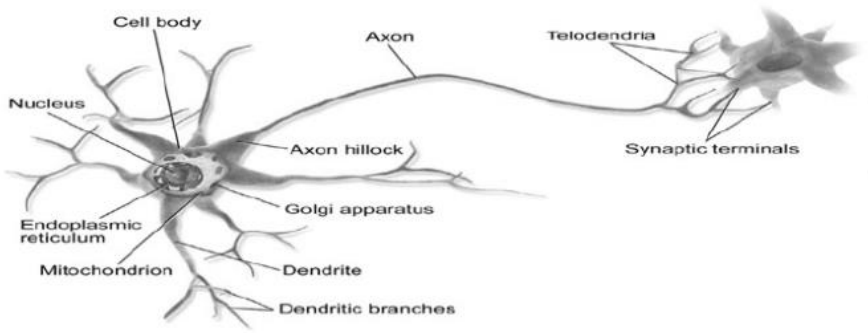

Рисунок 1.4 — Біологічний нейрон

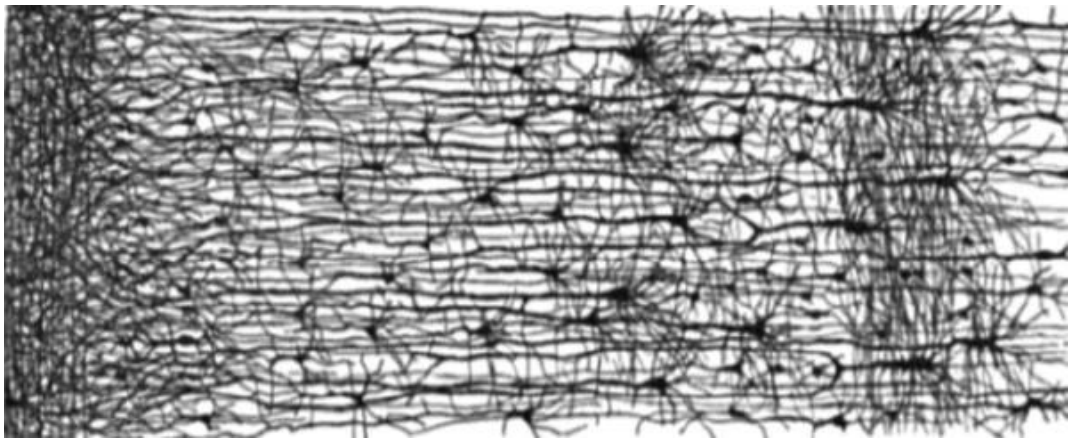

Рис. 1.5. Багатошарова структура кори головного мозку

Штучна нейрона мережа (ШНМ) — це обчислювальна система, що складається з великої кількості простих елементів — нейронів, які працюють паралельно та взаємодіють між собою через зважені зв’язки (ваги). Кожен нейрон отримує вхідні сигнали x1,x2,…xp, які множаться на відповідні ваги w1,w2,…,wp. Сума цих сигналів формує чистий вхід нейрона формула 1.1.

$$v_j = \sum_{i=1}^{p} w_{ij} x_i, \tag{1.1}$$

Результат проходить через функцію активації $g(v_j)$, яка визначає вихід нейрона y_i. Ця функція може бути пороговою, сигмоїдною, ReLU тощо.

Таким чином, нейрон активується лише тоді, коли вхідний сигнал перевищує певне порогове значення. Візуалізацію моделі штучного нейрона можна побачити на рис. 1.6 .

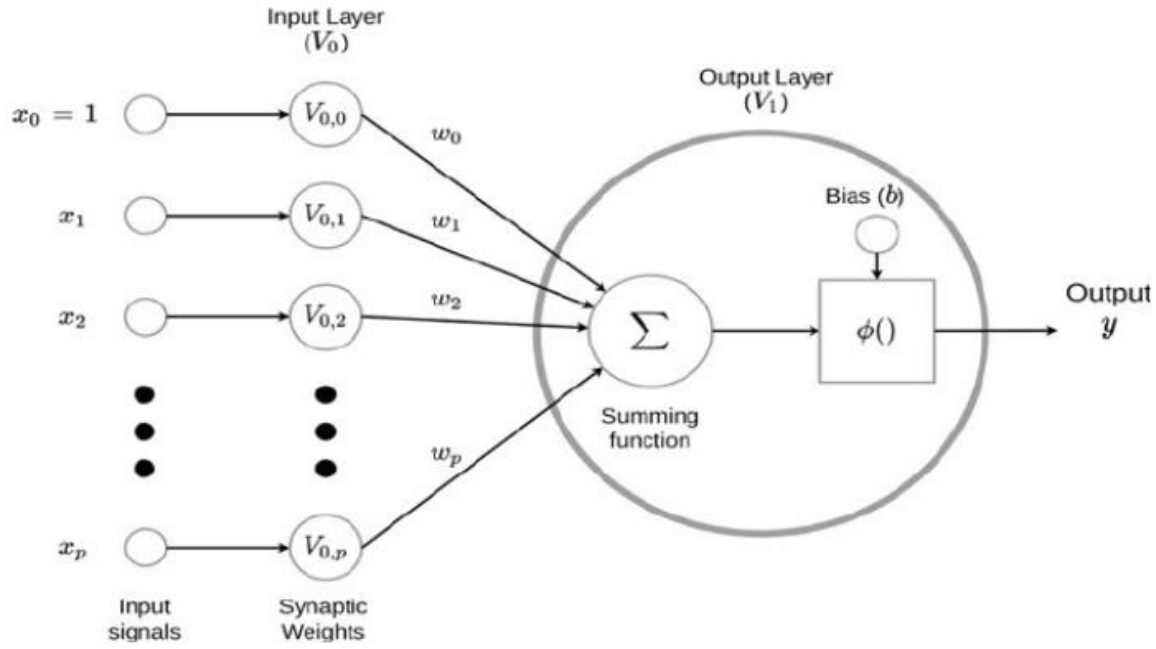

Рисунок 1.6 — Узагальнена модель штучного нейрона

Архітектура штучних нейронних мереж (ШНМ) є одним із ключових аспектів сучасних систем штучного інтелекту, що визначає їхню здатність аналізувати дані, виявляти закономірності та приймати рішення [11]. Під архітектурою розуміють структурну організацію нейронів, способи їхнього з’єднання між шарами та механізми обробки сигналів. Незважаючи на складність, основна ідея залишається простою, нейронна мережа моделює роботу мозку людини, у якому мільярди нейронів взаємодіють через синапси, формуючи гнучку та адаптивну систему обробки інформації.

Типова штучна нейронна мережа складається з набору шарів, кожен з яких виконує певну функцію. Вхідний шар сприймає дані ззовні, перетворюючи їх у числову форму, зручну для подальшої обробки. Наступні приховані шари поступово виділяють все складніші ознаки вхідних даних, а вихідний шар формує кінцевий результат — наприклад, імовірність належності об’єкта до певного класу або його просторові координати. У межах кожного шару

розташовані окремі обчислювальні елементи — штучні нейрони. Кожен нейрон приймає сигнали з попереднього шару, зважує їх відповідно до певних коефіцієнтів, підсумовує та передає результат далі через нелінійну функцію активації. Саме ця функція, що може мати сигмоїдальний, гіперболічний або ReLU-тип, надає мережі здатності до апроксимації нелінійних залежностей, що є критично важливим для задач розпізнавання образів і комп'ютерного зору.

Існує безліч варіантів архітектур нейронних мереж, кожна з яких пристосована до конкретного типу даних або завдання. Найпростішою формою є одношарова мережа, здатна лише до лінійного розділення даних, однак на практиці значно частіше використовуються багатошарові моделі — так звані перцептрони, де кілька прихованих шарів дозволяють виявляти складні структури у вхідних сигналах. Коли ж кількість таких шарів зростає до десятків і навіть сотень, говорять про глибокі нейронні мережі. Вони формують послідовну ієрархію представлень, нижчі шари розпізнають прості елементи, такі як контури чи кольори, тоді як вищі навчаються комбінувати їх у складні образи — наприклад, частини об'єктів або цілі сцени. Саме ця ієрархічна природа обробки даних дала поштовх розвитку сучасних систем глибокого навчання, які стали основою успіхів у комп'ютерному зорі, автономному керуванні та мовному розпізнаванні.

Одним з найважливіших компонентів архітектури є спосіб з'єднання нейронів між шарами. У класичних прямонаправлених (feedforward) мережах сигнал поширюється від входу до виходу лише в одному напрямку. Така структура є стабільною, легкою для реалізації та найчастіше використовується в задачах класифікації чи регресії. Натомість у рекурентних мережах, що застосовуються для обробки послідовностей, передбачено зворотні зв'язки, які дозволяють моделі "пам'ятати" попередні стани. Ця властивість робить їх ефективними для аналізу часових рядів, відеопотоків і сигналів. У системах комп'ютерного зору ключову роль відіграють згорткові нейронні мережі (Convolutional Neural Networks, CNN), які імітують роботу зорової кори людини. Вони використовують спеціальні фільтри, що автоматично виділяють характерні

ознаки зображення — краї, текстури або форми — та роблять це набагато ефективніше, ніж звичайні повнозв’язані структури. Саме CNN-архітектури, такі як YOLO, ResNet чи MobileNet, стали стандартом для задач детекції, класифікації та відстеження об’єктів, включно з використанням на обмежених апаратних платформах на кшталт Raspberry Pi.

Невід’ємною складовою роботи будь-якої нейронної мережі є процес навчання, який визначає, як саме система набуває “знання” про світ. На початковому етапі ваги з’єднань між нейронами встановлюються випадковим чином, а далі поступово змінюються відповідно до якості результатів. Якщо прогноз мережі не збігається з очікуваним значенням, обчислюється похибка, яка потім поширюється у зворотному напрямку — від вихідного шару до вхідного. Цей механізм, відомий як алгоритм зворотного поширення помилки (backpropagation), дозволяє коригувати ваги так, щоб у майбутніх ітераціях мережа давала точніші результати. Керуючись принципами градієнтного спуску, нейронна мережа поступово мінімізує функцію втрат, тобто наближає свої передбачення до реальних даних. На практиці навчання відбувається на великих наборах прикладів, що дозволяє мережі виявляти закономірності, які неможливо описати аналітично.

Процес навчання може бути різних типів — під контролем, без учителя або з підкріпленням. У першому випадку мережа отримує пари «вхід — правильна відповідь» і вчиться відтворювати закономірність. У другому — самостійно шукає структуру у даних, а в третьому — удосконалює свої дії, отримуючи винагороду за правильні рішення. У системах комп’ютерного зору, де навчальні вибірки великі, найчастіше застосовують саме навчання з учителем. Такий підхід дозволяє досягати високої точності при розпізнаванні об’єктів, облич, транспортних засобів чи інших цілей, що є особливо важливим у контексті автономних безпілотних систем [12].

Загалом, архітектура нейронної мережі визначає баланс між точністю, швидкодією та обчислювальною складністю. Для реалізації системи на пристроях із обмеженими ресурсами, важливо знаходити компроміс між

кількістю параметрів моделі та якістю розпізнавання. Тому в сучасних розробках активно застосовуються методи оптимізації — квантизація ваг, прунінг, використання легких архітектур на кшталт MobileNet або Tiny-YOLO. Це дозволяє забезпечити роботу систем комп’ютерного зору в реальному часі навіть на слабких апаратних платформах, що й становить практичну основу для створення модулів автономного розпізнавання цілей у FPV-дронах.

Таким чином, штучні нейронні мережі є результатом еволюції обчислювальних моделей, натхнених принципами роботи людського мозку. Їхня архітектура визначає здатність системи до узагальнення, а принцип навчання забезпечує адаптацію до нових даних. У поєднанні ці елементи формують фундамент сучасного штучного інтелекту, який дедалі глибше інтегрується в робототехніку, комп’ютерний зір та автономні системи керування.

1.5 Методи виявлення об’єктів у зображеннях та відеопотоці

Перші системи комп’ютерного зору для виявлення об’єктів ґрунтувалися на алгоритмах, які використовували ручно спроєктовані ознаки (features). Основна ідея полягала в тому, що кожне зображення можна представити як набір математичних дескрипторів — контурів, градієнтів, текстур або кольорових характеристик, — за допомогою яких можна було розрізняти об’єкти певного типу [13].

Одним із найвідоміших підходів був алгоритм Haar Cascades, запропонований Виолою та Джонсом (Viola & Jones, 2001). Він використовував набір простих фільтрів (Haar-like features), які описували локальні контрасти між ділянками зображення. На основі великої кількості позитивних і негативних прикладів навчалася каскадна модель, здатна швидко розпізнавати обличчя чи інші об’єкти. Хоча метод мав обмежену гнучкість, він став базовим для ранніх систем детекції в реальному часі, зокрема в застосуваннях OpenCV.

Іншим популярним підходом був метод HOG (Histogram of Oriented Gradients), розроблений Дала́лем і Тріггсом (Dalal & Triggs, 2005). Він полягав у тому, що кожна ділянка зображення описувалася гістограмою напрямків

градієнтів яскравості. Такі ознаки добре характеризували форму об’єктів і були стійкими до змін освітлення. Для класифікації використовувався метод Support Vector Machine (SVM), який відокремлював об’єкти від фону у просторі ознак.

Для аналізу відеопотоку класичні методи часто поєднували з Background Subtraction — відніманням поточного кадру від моделі фону, що дозволяло виділити рухомі об’єкти, або з Optical Flow, який оцінював напрям і швидкість переміщення точок між послідовними кадрами.

Однак такі методи мали суттєві обмеження, вони були чутливими до змін освітлення, поворотів, масштабу, частково закритих об’єктів і потребували ретельного ручного налаштування параметрів. Це зумовило перехід до моделей, які могли автоматично навчатися виділяти ознаки — тобто до нейронних мереж.

Поява згорткових нейронних мереж (Convolutional Neural Networks, CNN) докорінно змінила підхід до задач детекції об’єктів. На відміну від класичних методів, CNN автоматично формують ознаки без необхідності ручного опису контурів чи текстур. Вони навчаються безпосередньо з піксельних даних, виявляючи складні нелінійні залежності.

Першим проривом стала архітектура R-CNN , у якій використовувався дворівневий підхід, спочатку алгоритм створював регіони-кандидати, а потім CNN класифікувала кожен із них. Надалі з’явилися Fast R-CNN і Faster R-CNN, які оптимізували цей процес, перетворивши детекцію на єдину згорткову мережу, здатну працювати значно швидше.

Паралельно з цим розвивалися одностадійні (single-stage) методи, зокрема YOLO (You Only Look Once) та SSD (Single Shot MultiBox Detector) [14]. Вони здійснюють детекцію без етапу генерації регіонів, мережа безпосередньо прогнозує координати рамок (bounding boxes) та класи об’єктів у реальному часі. Це зробило можливим використання CNN у системах, де потрібна висока швидкодія — наприклад, у безпілотних апаратах чи мобільних роботах.

Архітектура YOLO стала символом балансу між точністю та швидкістю. Замість поетапної обробки вона розглядає зображення цілком, ділить його на сітку й для кожної клітинки передбачає кілька можливих рамок. Завдяки такій

структурі YOLO досягає високої продуктивності, зберігаючи прийнятну точність навіть на пристроях з обмеженими ресурсами.

Сучасні модифікації, такі як YOLOv7, YOLOv8 і YOLOv11, поєднують ідеї глибинних згорток, механізми уваги (attention), багаторівневу детекцію об’єктів різного масштабу та покращену оптимізацію втрат (loss functions). Це дозволяє системам комп’ютерного зору на основі CNN розпізнавати десятки категорій об’єктів із високою точністю навіть у складних сценах [15].

Детекція у відеопотоці є складнішою задачею, ніж на окремих зображеннях, оскільки вимагає врахування часової динаміки та зміни контексту. Об’єкти можуть переміщуватися, частково перекриватися або зникати з кадру, що створює додаткові труднощі для стабільного розпізнавання.

Першим кроком є покадрове виявлення за допомогою моделей, подібних до YOLO або SSD. Проте для забезпечення послідовності й точності результатів необхідно здійснювати відстеження об’єктів (tracking). Для цього застосовують алгоритми, які аналізують траєкторії об’єктів між кадрами.

Одним із найпростіших, але ефективних підходів є SORT (Simple Online and Realtime Tracking), який поєднує детекцію з фільтрами Калмана для прогнозування руху та евклідовою метрикою для зіставлення об’єктів між кадрами. Його розвитком став DeepSORT, який доповнив базовий алгоритм нейронною мережею для порівняння зовнішнього вигляду об’єктів, що значно підвищило стійкість до тимчасових втрат детекції [16].

Останніми роками з’явилися алгоритми нового покоління — ByteTrack і BoT-SORT, які інтегрують детекцію та трекінг у єдину систему, використовуючи навіть низькоконфідентні рамки для покращення асоціації між кадрами. Такі рішення особливо актуальні для FPV-дронів, де відеопотік обробляється в реальному часі, а затримка між кадрами може критично впливати на керування.

Додатково застосовуються методи оптичного потоку (Optical Flow), що оцінюють напрям і швидкість руху пікселів, дозволяючи відокремлювати об’єкти, які рухаються незалежно від фону. Поєднання CNN-детектора, трекера

та оптичного потоку дає змогу формувати цілісну картину сцени, аналізуючи не лише просторові, а й часові взаємозв’язки.

Таким чином, сучасні системи відеоаналізу базуються на поєднанні детекції та трекінгу, що дозволяє забезпечити стабільне розпізнавання об’єктів у складних динамічних умовах — таких як польоти безпілотників, відеоспостереження чи автономне водіння.

1.6 Методи трекінгу та відстеження рухомих цілей

Трекінг — це процес безперервного відстеження положення одного або кількох об’єктів у послідовності відеокадрів. Основна мета трекінгу полягає у збереженні ідентичності об’єкта між кадрами та побудові його траєкторії руху в просторі. На відміну від детекції, що визначає положення об’єктів у кожному окремому кадрі, трекінг забезпечує часову узгодженість, дозволяючи системі розуміти динаміку сцени.

Завдання трекінгу охоплюють:

— визначення координат об’єкта в кожному кадрі;

— прогнозування його положення при часткових перекриттях або втраті візуальної видимості;

— підтримку унікального ідентифікатора для кожної цілі;

— адаптацію до змін форми, освітлення, масштабу та фону.

Трекінг є критично важливою частиною систем комп’ютерного зору, особливо у випадках, коли необхідно здійснювати аналітику в реальному часі — наприклад, під час роботи безпілотних літальних апаратів, систем відеоспостереження або автономного транспорту.

Перші підходи до відстеження об’єктів базувалися на аналітичних моделях руху та статистичному аналізі зміни пікселів у часі.

Одним із найвідоміших методів є оптичний потік (Optical Flow), який оцінює зміщення кожного пікселя між двома кадрами. Алгоритм Lucas–Kanade дозволяє визначати напрямок і швидкість руху об’єкта, ґрунтуючись на

припущенні про сталість яскравості. Хоча він добре працює при малих переміщеннях, його точність знижується при швидкому русі чи зміні освітлення.

Методи MeanShift і CamShift (Continuously Adaptive MeanShift) базуються на аналізі гістограм кольорів об’єкта. Алгоритм ітеративно зсуває вікно пошуку в напрямку найвищої щільності пікселів, подібних до цілі. CamShift адаптує розмір вікна в залежності від зміни масштабу об’єкта, що дозволяє відстежувати цілі з різною відстанню до камери [17].

Для прогнозування траєкторії руху об’єкта використовуються фільтри Калмана (Kalman Filter) — математична модель, що оцінює поточний стан системи, комбінуючи вимірювання з попереднім прогнозом. У випадках нелінійного руху застосовуються частинкові фільтри (Particle Filter), які моделюють можливі стани системи за допомогою набору випадкових зразків (частинок).

Класичні методи відрізняються високою швидкодією та низькими обчислювальними вимогами, що робить їх придатними для простих або вбудованих систем. Проте вони є чутливими до шумів, зміни освітлення, масштабів та перекриття об’єктів, що обмежує їх ефективність у складних динамічних сценах.

З розвитком методів машинного навчання з’явилися підходи, які використовують попередньо обчислені ознаки (features) та алгоритми адаптивного навчання. До таких належать кореляційні фільтри (Correlation Filter Trackers), що визначають найбільш подібну область між шаблоном об’єкта та поточним кадром.

Алгоритм MOSSE (Minimum Output Sum of Squared Error) став першим трекером, здатним працювати в реальному часі, завдяки ефективному оновленню фільтра в частотній області. Подальші вдосконалення, як-от KCF (Kernelized Correlation Filter) та CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability), використовують багатоканальні ознаки (наприклад, HOG, колірні вектори) для підвищення точності.

Інші підходи, такі як Boosting Tracker і MIL (Multiple Instance Learning) Tracker, оновлюють модель об’єкта під час трекінгу, адаптуючи її до поступових змін зовнішнього вигляду. Це дозволяє системі краще справлятися зі змінами форми або орієнтації об’єкта.

Кореляційні трекери залишаються популярними завдяки оптимальному балансу між швидкодією і точністю. Вони часто використовуються як базові модулі у вбудованих системах, наприклад у бібліотеці OpenCV.

Розвиток глибокого навчання дав змогу створювати трекери, що поєднують переваги нейронних мереж із класичними принципами відстеження.

Одним із ключових напрямів є Siamese-based трекери, які використовують дві гілки згорткової мережі для порівняння шаблону об’єкта (з еталонного кадру) з поточним зображенням. Архітектури SiamFC (Fully Convolutional), SiamRPN, SiamMask та SiamBAN забезпечують високу точність завдяки глибоким фічам, що описують візуальні властивості об’єкта.

Інший підхід — інтеграція трекінгу з системами детекції. Алгоритм DeepSORT (Deep Simple Online and Realtime Tracking) поєднує базовий SORT з нейронною мережею, яка створює вектор ознак зовнішнього вигляду. Це дозволяє більш надійно асоціювати об’єкти між кадрами, навіть якщо їх частково закрито або вони перетинаються.

Сучасні рішення, такі як ByteTrack і BoT-SORT, удосконалюють процес асоціації, використовуючи як висококонфідентні, так і низькоконфідентні детекції для збереження цілісності траєкторій. Завдяки цьому досягається висока стабільність і точність при мінімальних втратах продуктивності.

Глибинні трекери мають суттєві переваги в складних умовах, однак потребують значних обчислювальних ресурсів. Для реального часу їх оптимізують за допомогою апаратних прискорювачів (GPU, TPU) або перетворення моделей у полегшені формати.

Сучасні системи комп’ютерного зору найчастіше реалізують підхід tracking-by-detection, у якому процес відстеження базується на результатах покадрової детекції. Спочатку модель (наприклад, YOLO, SSD або Faster R-

CNN) визначає положення об’єктів у кожному кадрі, після чого алгоритм трекінгу зіставляє їх між кадрами, формуючи безперервні траєкторії.

Цей підхід має низку переваг:

— гнучкість — можливість використання будь-якого детектора;

— точність — оновлення положення об’єктів у кожному кадрі;

— масштабованість — відстеження багатьох цілей одночасно.

У реальних системах часто застосовують комбінації YOLOv5/YOLOv8 із DeepSORT або ByteTrack, що забезпечує баланс між швидкістю та надійністю. Такі рішення особливо ефективні для FPV-дронів, де відеопотік обробляється в реальному часі, а система має не лише розпізнавати об’єкти, але й передбачати їх подальший рух для ухилення або наведення. Завдяки поєднанню детекції та трекінгу досягається комплексне розуміння сцени — система може не лише бачити, але й аналізувати поведінку об’єктів у просторі та часі.

1.7 Проблеми, обмеження та перспективи розвитку систем розпізнавання для БПЛА

Системи комп’ютерного зору, інтегровані у безпілотні літальні апарати (БПЛА), стикаються з низкою специфічних викликів, які відрізняють їх від стаціонарних або наземних систем. Основною особливістю є динамічний характер зйомки, дрон постійно змінює висоту, кут огляду, швидкість та напрям руху. Це призводить до варіацій масштабу об’єктів, змін освітлення, появи тіней і відблисків, що ускладнює стабільну роботу алгоритмів розпізнавання.

Додатковими факторами є вібрації корпусу дрона, обмежена стабілізація відеопотоку та наявність шумів у кадрах. Робота в реальному часі потребує високої швидкодії, тоді як канали зв’язку мають обмежену пропускну здатність і часто характеризуються затримками. Унаслідок цього система повинна забезпечувати баланс між точністю, швидкістю та енергоспоживанням, що є ключовим викликом для архітекторів подібних рішень.

Основне апаратне обмеження систем розпізнавання в БПЛА полягає у невисоких обчислювальних можливостях бортових пристроїв. У більшості

випадків використовуються енергоефективні платформи, такі як Raspberry Pi, Jetson Nano, Orange Pi або Google Coral Dev Board, які мають обмежену кількість ядер, невеликий обсяг пам’яті та обмежену продуктивність графічного прискорювача. Це створює труднощі при виконанні складних нейронних моделей у реальному часі.

Також важливу роль відіграє енергоспоживання — живлення від акумулятора дрона обмежує тривалість польоту та потужність, яку можна виділити на обчислення. Значний вплив має і тепловий режим, компактні корпуси без активного охолодження призводять до перегріву процесора, що викликає тротлінг і зниження швидкодії.

Для підвищення продуктивності та зменшення навантаження на обчислювальні ресурси вбудованих платформ застосовуються методи оптимізації нейронних мереж, зокрема квантізація, pruning та knowledge distillation. Крім того, використовуються апаратні прискорювачі нейронних обчислень, такі як NPU, TPU та VPU. Поєднання цих підходів дозволяє забезпечити прийнятний баланс між точністю розпізнавання та швидкодією системи на енергонезалежних та ресурсозалежних пристроях.

Алгоритмічна складність — одна з головних причин, що стримують впровадження високоточного комп’ютерного зору на борту дронів. Глибинні моделі, зокрема архітектури YOLO, Faster R-CNN або DETR, потребують значних обчислювальних ресурсів для обробки навіть одного кадру. У реальних умовах польоту, коли потрібно досягти частоти щонайменше 20–30 кадрів за секунду, це стає критичним чинником.

Іншою проблемою є узагальнення моделей (generalization). Мережі, навчені на стандартних датасетах (COCO, Pascal VOC, VisDrone), не завжди коректно працюють в умовах, відмінних від навчальних, зміна типу камери, кута огляду, погодних умов чи сезонності призводить до втрати точності. Крім того, у багатьох наборах даних спостерігається дисбаланс класів (class imbalance) — одні типи об’єктів представлені значно частіше, ніж інші, що знижує якість розпізнавання малопоширених цілей.

Особливо складним завданням залишається детекція малих або частково перекритих об’єктів. Для таких випадків стандартні моделі втрачають ефективність, тому застосовуються багаторівневі фіч-мапи (FPN), додаткові блоки уваги (Attention Mechanisms) та постобробка результатів.

Інтеграція системи розпізнавання у загальну архітектуру дрона пов’язана з численними технічними труднощами. По-перше, система комп’ютерного зору повинна взаємодіяти з іншими модулями — контролером польоту, модулем навігації, телеметрією, системою передачі відео. Кожен із цих компонентів має власні часові вимоги, що створює ризик затримки обробки (latency). Навіть кількасот мілісекунд затримки можуть спричинити помилку у визначенні позиції цілі чи некоректну реакцію дрона.

Складність викликає також синхронізація даних між сенсорами — камерою, GPS і інерційним блоком (IMU). Неправильне узгодження часових міток призводить до похибок при побудові траєкторії руху об’єктів. Ще одна проблема — обмежений канал зв’язку між обчислювальним блоком (наприклад, Raspberry Pi) і польотним контролером (через UART або Ethernet), що накладає обмеження на обсяг передаваних даних і частоту оновлення.

Надійність системи також має критичне значення, при тривалих польотах можливі перегріви**,** перезапуски або збої пам’яті, що потребує реалізації механізмів відновлення роботи та самодіагностики.

Незважаючи на перелічені труднощі, напрям розвитку систем комп’ютерного зору для БПЛА демонструє значний прогрес. Одним із ключових трендів є оптимізація нейронних мереж для вбудованих пристроїв. З’являються полегшені архітектури, такі як YOLOv8n, MobileNetV3, NanoDet, EfficientDet, які дозволяють досягати високої швидкодії без суттєвої втрати точності.

Важливу роль відіграє розвиток edge computing — обробка відеопотоку безпосередньо на борту дрона без відправлення даних у хмару. Це зменшує затримку та забезпечує автономність системи. Для підвищення швидкодії активно застосовуються TensorRT, OpenVINO, NCNN, TFLite, що дозволяють конвертувати й оптимізувати моделі для різних апаратних платформ.

Окремим перспективним напрямом є мультимодальні системи, які об’єднують інформацію з кількох сенсорів — оптичної камери, тепловізора, LiDAR та GPS. Такий підхід дозволяє підвищити стійкість до зовнішніх умов і точність розпізнавання.

Також активно розвиваються трансформерні архітектури (Vision Transformer, DETR), що дають змогу аналізувати просторові взаємозв’язки між об’єктами більш ефективно, ніж класичні CNN.

Серед довгострокових напрямів — самонавчання систем (online learning), коли модель здатна адаптувати свої параметри під час польоту, та інтелектуальна взаємодія кількох дронів (swarm intelligence) із обміном результатами розпізнавання між апаратами.

Системи розпізнавання об’єктів для безпілотних літальних апаратів стикаються з низкою викликів, серед яких — обмежені апаратні ресурси, складність алгоритмів, проблеми інтеграції та стабільності роботи в реальних умовах. Разом з тим, активний розвиток напрямів оптимізації нейронних мереж, використання апаратних прискорювачів і впровадження легких архітектур створюють передумови для побудови повністю автономних рішень комп’ютерного зору.

Подальший прогрес у галузі пов’язаний із пошуком оптимального балансу між точністю, швидкістю та енергоефективністю, а також із розвитком адаптивних систем, здатних працювати в непередбачуваних умовах реального світу. Таким чином, питання вибору оптимальної архітектури та технологічного стеку стає ключовим для побудови ефективної системи комп’ютерного зору на борту БПЛА.

2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ВИБІР АРХІТЕКТУРИ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ

2.1 Постановка задачі розпізнавання та відстеження цілей у реальному часі

Метою даної магістерської роботи є розробка та впровадження апаратно-програмного комплексу системи комп'ютерного зору для безпілотного літального апарата (БПЛА), призначеного для автоматизації процесу виявлення та супроводу цілей у реальному часі. Основним завданням комплексу є забезпечення двох режимів взаємодії оператора з системою, що поєднують людський інтелект із потужністю сучасних методів штучного інтелекту.

Головна ціль — створення стабільної системи, здатної функціонувати на одноплатному комп'ютері з обмеженими обчислювальними ресурсами та інтегруватися з польотним контролером БПЛА.

Для досягнення цієї мети необхідно вирішити такі основні завдання, як реалізація двох основних режимів роботи:

— режим ручного захоплення;

— режим автономної детекції.

Режим ручного захоплення (Manual Targeting Mode) забезпечить інтерфейс, в якому оператор виконує візуальне виявлення та ідентифікацію цілі, позначаючи її за допомогою маркера-прицілу на відеопотоці. Після цього система повинна автоматично ініціювати супровід обраної цілі, відповідаючи за траєкторію польоту дрона для її утримання в центрі кадру.

Режим автономної детекції (Autonomous Detection Mode) забезпечить автоматичне виявлення цілей заданих класів за допомогою нейромережі YOLO. Для оптимізації продуктивності детекція має обмежуватися центральною зоною екрану. Після виявлення цілі система чекає на підтвердження оператора, і лише після цього ініціює процес супроводу.

Організувати стабільне відстеження (трекінг) обраної цілі між кадрами з мінімальними втратами та перемиканнями ідентифікаторів.

Забезпечити передачу керуючих команд від одноплатного комп’ютера до польотного контролера для реалізації руху дрона по траєкторії, що забезпечує супровід цілі.

Система комп’ютерного зору повинна відповідати ряду функціональних та експлуатаційних вимог.

Функціональні вимоги:

— реалізація інтерфейсу взаємодії з оператором (візуалізація прицілу, індикація цілей, підтвердження вибору);

— стабільне відстеження цілей із компенсацією руху носія (дрона);

— інтеграція з польотним контролером ;

— ефективне функціонування на дистанціях від 300 метрів.

Вимоги до продуктивності:

— частота обробки відео — не менше 20 кадрів/с для забезпечення плавності керування;

— загальна затримка в контурі управління (від отримання кадру до передачі команди) — не більше 100 мс;

— стійкість до зовнішніх факторів (вібрації, зміни освітлення, рух платформи).

Розробка системи здійснюється в рамках наступних обмежень:

— апаратні обмеження — обмежена обчислювальна потужність, обсяг оперативної пам’яті та тепловий пакет доступного одноплатного комп’ютера;

— експлуатаційні обмеження — необхідність функціонування в умовах реального часу, при різному освітленні та на рухомій платформі.

Критеріями успішної реалізації системи вважаються:

— досягнення цільових показників продуктивності **(**20 FPS**,** затримка <100 мс);

— стабільне відстеження цілі протягом усього часу ураження без втрат;

— успішна інтеграція з польотним контролером та коректна передача керуючих команд;

— забезпечення зручного та інтуїтивного інтерфейсу для оператора в обох режимах роботи.

Таким чином, задача полягає у створенні гібридної системи, яка ефективно розподіляє функції між оператором (як джерелом цілевказання або ланкою підтвердження) та алгоритмами штучного інтелекту як інструментом детекції та стабілізації. Подолання апаратних обмежень для досягнення необхідної швидкодії та низької затримки є основним технічним викликом, що визначає необхідність подальшого аналізу технологій, архітектурних рішень та методів оптимізації, які будуть розглянуті в наступних підрозділах.

2.2 Аналіз сучасних бібліотек і фреймворків комп'ютерного зору

Сучасні системи комп'ютерного зору є результатом тісного поєднання математичних методів обробки зображень, глибокого навчання та апаратних технологій оптимізації. Ефективність таких систем значною мірою визначається вибором бібліотек і фреймворків, на основі яких реалізується обробка зображень, нейронна детекція об'єктів, трекінг і комунікація з іншими підсистемами. У цьому підрозділі проведено аналіз основних інструментів, що застосовуються у сфері комп'ютерного зору, з урахуванням специфіки побудови системи на одноплатному комп'ютері, де ключовими вимогами є продуктивність, компактність та енергоефективність.

OpenCV (Open Source Computer Vision Library) є фундаментальним інструментом у більшості сучасних систем комп'ютерного зору. Це відкрита кросплатформна бібліотека, що забезпечує широкий спектр алгоритмів для обробки, аналізу та інтерпретації зображень і відео у реальному часі. Вона розроблялася з початку 2000-х років та отримала надзвичайно велику спільноту користувачів і розробників, що постійно розширюють її функціональність. Наразі OpenCV містить понад 2500 оптимізованих алгоритмів, реалізованих на C++ та доступних через інтерфейси для Python, Java і JavaScript [18].

Архітектура бібліотеки модульна, вона складається з окремих підсистем, таких як core, imgproc, video, calib3d, features2d, objdetect, highgui, dnn тощо.

Такий поділ дозволяє вибирати тільки потрібні компоненти при збірці проєкту, що є важливою перевагою у випадку систем з обмеженими ресурсами.

Модуль core відповідає за базові структури даних (наприклад, cv::Mat), що використовуються для представлення зображень, а також за математичні операції над матрицями та тензорами. imgproc містить алгоритми для фільтрації, перетворення кольорових просторів, морфологічної обробки, детекції контурів і трансформацій. video використовується для задач трекінгу, оцінки оптичного потоку та аналізу руху.

Особливе значення у системах реального часу має модуль highgui, який забезпечує захоплення та відображення відеопотоку з камер, підтримує роботу із форматами відео та фотофайлів, а також дозволяє взаємодіяти з інтерфейсом користувача (наприклад, відображення прицілу, маркерів тощо). На Raspberry Pi цей модуль відповідає за взаємодію з камерою через API Video4Linux2 або libcamera, що дозволяє безпосередньо працювати з потоком із сенсора.

Крім цього, OpenCV дозволяє здійснювати геометричні корекції зображення, що є важливим при використанні ширококутних камер дронів, де спотворення перспективи можуть впливати на точність детекції. За допомогою функцій cv2.undistort() або cv2.initUndistortRectifyMap() можна проводити калібрування камери, що підвищує стабільність розпізнавання.

Бібліотека підтримує апаратне прискорення через OpenCL, що дозволяє виконувати частину обчислень (наприклад, згортки, розмиття, ресайзинг) на GPU. Це забезпечує суттєве зменшення часу обробки одного кадру — у деяких тестах до 25–30 % швидше порівняно з чистим CPU-виконанням. Додатково можливе використання SIMD-інструкцій (NEON) для прискорення векторних операцій.

Також варто відзначити, що OpenCV має можливість збирання з опцією оптимізації під ARM-архітектуру, що дозволяє максимально ефективно використовувати апаратні ресурси одноплатних комп'ютерів. Для задач реального часу важливо налаштувати конвеєр так, щоб уникати зайвих копій даних між CPU та GPU, оскільки це може створювати затримки.

Однією з найбільш потужних особливостей OpenCV є модуль DNN (Deep Neural Networks), який надає можливість виконувати інференс уже натренованих моделей глибокого навчання без потреби у сторонніх фреймворках. DNN-модуль підтримує широкий набір форматів моделей — Caffe, TensorFlow, ONNX, Darknet (YOLO), Torch, а також інтерфейси для TensorFlow Lite.

Переваги використання OpenCV у системі:

— висока універсальність і сумісність — бібліотека сумісна з багатьма мовами програмування та фреймворками, має підтримку різних форматів нейронних моделей;

— оптимізація під ARM-платформи — можливість компіляції з NEON-інструкціями, OpenCL-підтримкою і мінімальними залежностями;

— стабільність та перевіреність часом — OpenCV використовується у промислових, наукових і навчальних проєктах по всьому світу.

Таким чином, OpenCV виступає ключовим компонентом розроблюваної системи комп’ютерного зору. Вона поєднує класичні алгоритми аналізу зображень з можливостями нейронних мереж, забезпечує необхідну швидкодію та адаптованість.

TensorFlow та PyTorch є двома провідними фреймворками у сфері глибокого навчання, які забезпечують повний цикл створення, тренування, тестування та розгортання нейронних мереж. Вони стали основними інструментами для реалізації сучасних моделей комп’ютерного зору, включно з архітектурами сімейств YOLO, SSD, Faster R-CNN, DeepSORT та ін [19].

TensorFlow — це масштабований фреймворк, розроблений компанією Google, який відзначається високим рівнем оптимізації для виконання на різних апаратних платформах — від серверних систем з потужними GPU до вбудованих пристроїв на базі ARM-процесорів. Основною перевагою TensorFlow є його гнучка екосистема, що включає не лише базову бібліотеку, а й допоміжні модулі — Keras для швидкого створення моделей, TensorBoard для моніторингу процесу навчання та TensorFlow Lite для роботи на пристроях із низькою обчислювальною потужністю.

TensorFlow Lite є ключовим компонентом для застосування у вбудованих системах, подібних до Raspberry Pi. Його основна особливість полягає у можливості квантизації моделей — тобто зниження точності представлення ваг до форматів float16 або int8. Це дозволяє зменшити обсяг моделі в кілька разів без суттєвих втрат точності, що є критично важливим при роботі з обмеженою оперативною пам’яттю. Крім того, TensorFlow Lite підтримує апаратне прискорення через GPU Delegate, NNAPI або NPU Delegate, завдяки чому моделі можуть працювати у реальному часі навіть на енергоефективних пристроях.

TensorFlow також має добре розвинуті засоби конвертації та інтеграції з іншими форматами моделей, зокрема через ONNX або SavedModel. Це дозволяє переносити нейронні мережі між різними середовищами — наприклад, з десктопної версії TensorFlow у вбудовану TFLite без необхідності повторного навчання. Завдяки цьому TensorFlow залишається одним із найкращих варіантів для побудови повноцінного циклу — від дослідження до розгортання.

PyTorch, розроблений компанією Meta, у свою чергу, став стандартом де-факто для досліджень, навчання та експериментів у галузі глибокого навчання. Його головною перевагою є динамічний обчислювальний граф, який дозволяє змінювати архітектуру моделі “на льоту” — без необхідності статичної компіляції. Це особливо зручно при тестуванні нових ідей, створенні кастомних шарів або оптимізаційних алгоритмів.

PyTorch також має високу інтеграцію з CUDA (для NVIDIA GPU) та Metal (на macOS), що забезпечує швидке тренування навіть великих моделей. Для розгортання на малопотужних пристроях передбачено експорт у формат TorchScript або ONNX (Open Neural Network Exchange), який виступає універсальним проміжним стандартом. Через ONNX модель PyTorch може бути легко інтегрована в інші середовища — наприклад, у TensorRT, OpenVINO чи NCNN, де вона отримує додаткові оптимізації для виконання.

PyTorch також підтримує набір бібліотек для візуальних завдань — TorchVision, яка містить попередньо навчені моделі для детекції, класифікації та сегментації об’єктів (ResNet, YOLOv8, Mask R-CNN тощо). Це значно

прискорює розробку систем комп’ютерного зору, дозволяючи зосередитися на адаптації моделей під конкретну задачу, а не на створенні архітектури з нуля.

У контексті даної роботи TensorFlow розглядається як ефективний інструмент для оптимізації моделей під обмежені обчислювальні можливості, тоді як PyTorch — як зручне середовище для експериментів, навчання та первинного тестування моделей перед подальшою конвертацією у легші формати. Такий підхід поєднує наукову гнучкість PyTorch із прикладною ефективністю TensorFlow Lite, що дозволяє досягти балансу між точністю, швидкодією та апаратними обмеженнями.

Формат ONNX є відкритим стандартом, що дозволяє передавати нейронні мережі між різними фреймворками без втрати сумісності. Наприклад, модель, навчена у PyTorch, може бути експортована у формат ONNX і виконуватись у TensorFlow, OpenCV або OpenVINO. Для системи комп’ютерного зору на Raspberry Pi це особливо важливо, оскільки ONNX дозволяє створити універсальний конвеєр, тренування моделі відбувається на потужному ПК або в хмарі, після чого модель у форматі ONNX переноситься на вбудований пристрій для виконання.

OpenVINO — це набір інструментів від Intel, призначений для оптимізації та прискорення інференсу нейронних мереж на процесорах Intel, GPU та VPU (наприклад, Intel Movidius). Основна його перевага полягає у використанні внутрішніх оптимізацій, таких як fusing операцій, квантизація, розподіл обчислень між ядрами CPU та GPU.

Хоча одноплатні комп'ютери в основному базуються на процесорі ARM, OpenVINO є корисним з точки зору етапу попередньої оптимізації та тестування моделі, особливо коли розробка ведеться спочатку на звичайному ПК. Завдяки OpenVINO можна оцінити потенційний приріст швидкодії від зменшення розміру моделі або зниження точності параметрів, а потім перенести результат на платформу іншу платформу.

MediaPipe — це фреймворк, розроблений компанією Google для побудови мультимедійних конвеєрів реального часу. Він забезпечує модульну структуру

обробки відеопотоків, включаючи детекцію облич, рук, об’єктів і пози людини. Хоча його застосування більше орієнтоване на мобільні пристрої, окремі компоненти MediaPipe можуть бути використані для реалізації допоміжних задач, таких як трекінг цілі або попередня фільтрація кадрів перед передачею до нейромережі YOLO. Для одноплатних комп'ютерів важливою перевагою MediaPipe є можливість побудови легковагових графів обробки, що ефективно використовують наявні ресурси CPU [21].

Проведений аналіз показує, що оптимальним рішенням для реалізації системи комп’ютерного зору є комбінація OpenCV як основної бібліотеки для роботи з відеопотоком і реалізації базових операцій, у поєднанні з PyTorch або TensorFlow для етапу тренування та тестування моделей нейронних мереж. Формат ONNX доцільно використовувати як проміжний обмінний формат для зручного перенесення моделі між середовищами.

2.3 Обґрунтування вибору програмних засобів реалізації системи

Проєктування системи комп’ютерного зору для безпілотного літального апарата вимагає ретельного вибору програмних засобів, здатних забезпечити баланс між точністю, швидкодією та енергоефективністю. Зважаючи на те, що система має функціонувати на обмеженому апаратному забезпеченні — одноплатному комп’ютері — вибір бібліотек і фреймворків здійснювався з урахуванням кількох критеріїв, сумісність із ARM-архітектурою, стабільність роботи в реальному часі, відкритість коду, наявність активної спільноти підтримки та можливість інтеграції з іншими компонентами системи зокрема, польотним контролером. У цьому підрозділі розглядаються ключові програмні компоненти системи, а також обґрунтовується доцільність їх використання в межах даного проєкту.

Основою розроблюваної системи є бібліотека OpenCV (Open Source Computer Vision Library), яка виступає базовим інструментом для захоплення, обробки та аналізу відеопотоку. OpenCV обрана завдяки поєднанню продуктивності, простоти інтеграції та широкого функціоналу, що охоплює всі

етапи попередньої обробки зображень — від зчитування кадрів із камери до фільтрації шумів і геометричних перетворень.

Однією з головних переваг OpenCV є її висока оптимізованість для роботи в реальному часі. Бібліотека підтримує використання апаратного прискорення через OpenCL,. Це особливо актуально при роботі з відео високої роздільної здатності (720p або 1080p), де навіть незначне зменшення затримки суттєво впливає на стабільність трекінгу цілі.

Крім того, OpenCV забезпечує повну сумісність із Python API, що значно спрощує розробку, налагодження та інтеграцію з іншими модулями системи. Підтримка стандартів V4L2 (Video4Linux2) гарантує стабільне зчитування потоку з камери, без потреби в додаткових драйверах.

У системі, що розробляється, OpenCV виконує такі функції:

— захоплення відеопотоку з камери в реальному часі;

— конвертація форматів кадрів (YUV, RGB, BGR);

— зменшення роздільної здатності для підвищення швидкодії;

— фільтрація шумів та компенсація освітлення;

— побудова зони інтересу (ROI) для фокусування обчислень лише на центральній частині кадру;

— передача підготовлених даних до нейронної мережі.

Таким чином, OpenCV є базовим рівнем системи, що забезпечує надійний фундамент для подальших модулів машинного навчання, трекінгу та керування.

Для розробки та тренування моделі детекції цілей обрано фреймворк PyTorch, який на сьогодні є одним із провідних інструментів у сфері глибокого навчання. PyTorch відзначається динамічним обчислювальним графом, що забезпечує високу гнучкість при розробці архітектури нейронних мереж і під час експериментів із параметрами. Це дає змогу швидко модифікувати структуру моделі, вносити зміни в шари або функції активації без необхідності повного перевизначення графа, як у TensorFlow 1.x.

Крім того, PyTorch має велику кількість попередньо навчених моделей, серед яких і серія YOLO, що стала базою для даного проєкту. Саме завдяки

відкритим реалізаціям YOLOv5–v11 можливо швидко провести адаптацію під власні набори даних і потреби конкретного застосування. PyTorch також відзначається зручністю роботи з GPU, але водночас дозволяє виконувати моделі й на CPU, що є важливим для середовищ без окремого графічного прискорювача. У процесі розробки використовується стандартний робочий цикл:

— тренування моделі на потужнішому середовищі ПК із GPU NVIDIA;

— збереження ваг у форматі .pt;

— подальше перенесення моделі на Raspberry Pi для виконання .

PyTorch забезпечує гнучкість при переході від стадії навчання до виконання, а його підтримка ONNX дає можливість експортувати модель у формат, сумісний із різними рушіями оптимізації. Це спрощує інтеграцію в системи з обмеженими ресурсами, зокрема у вбудовані платформи.

У межах магістерської роботи обрано архітектуру YOLOv11n (nano) як основну модель детекції. Серія YOLO є одним із найуспішніших підходів до розпізнавання об’єктів у реальному часі, адже поєднує високу точність (mAP) із мінімальною затримкою. YOLOv11n — це найкомпактніша версія з лінійки YOLOv11, оптимізована для роботи на пристроях із низькою обчислювальною потужністю. Вона використовує скорочену кількість шарів і параметрів, що дозволяє значно зменшити розмір моделі (до кількох мегабайт) при збереженні задовільної точності.

Основними аргументами на користь вибору YOLOv11n стали:

— оптимальний баланс між швидкістю та точністю для систем реального часу;

— підтримка багатокласової детекції, що дозволяє одночасно розпізнавати різні типи об’єктів;

— висока стійкість до змін освітлення та фону, що особливо важливо для польотів на відкритій місцевості;

— можливість подальшої оптимізації (наприклад, квантизація або обрізання шарів без істотної втрати точності).

Ці кроки дозволяють забезпечити стабільну роботу нейромережі із досягненні частоти кадрів не нижче 20 FPS, що відповідає вимогам системи реального часу.

Для забезпечення повного функціонування системи комп'ютерного зору, окрім основних бібліотек, використовуються кілька допоміжних засобів. NumPy — це фундаментальна бібліотека для наукових обчислень на Python, що забезпечує ефективну роботу з багатовимірними масивами. У межах даного проєкту вона використовується для попередньої підготовки зображень перед передачею в нейронну мережу, обчислення координат центроїдів цілей, перетворення типів даних і нормалізації. NumPy також виступає посередником між OpenCV (яке повертає кадри у форматі NumPy-масивів) і PyTorch, який очікує тензори у вигляді torch.Tensor.

MAVLink — це легковаговий протокол обміну даними між польотним контролером та зовнішніми пристроями. У системі він використовується для передачі координат цілей, векторів руху та команд керування. Завдяки відкритій структурі і підтримці Python-бібліотеки pymavlink забезпечується зручна інтеграція з контролерами сімейства Pixhawk або аналогічними. MAVLink дозволяє організувати двосторонній зв'язок.

Крім того, система використовує механізми багатопоточності для розділення завдань:

— окремий потік для захоплення відео;

— потік для обробки кадрів нейронною мережею;

— потік для передачі даних через MAVLink.

Такий підхід дозволяє уникнути "вузьких місць" та підвищити загальну стабільність системи при обмеженій продуктивності процесора.

Після вибору всіх компонентів формується повний програмний стек системи комп'ютерного зору. Мова програмування Python 3.11 — через її сумісність із більшістю бібліотек машинного навчання та зручність розробки в умовах швидкого прототипування. Бібліотеки**:** OpenCV, NumPy, PyTorch, pymavlink, threading.

Перевагами обраного стеку є:

— відкритість і кросплатформність (усі компоненти мають відкритий код);

— низька залежність від зовнішніх сервісів;

— гнучкість і розширюваність (можливість швидко змінювати модель або джерело відео);

— оптимальна швидкодія при мінімальних ресурсах.

Таким чином, обрані програмні засоби формують збалансований і взаємодоповнюючий стек, який забезпечує виконання всіх функцій системи комп'ютерного зору — від отримання відеопотоку до передачі команд керування дроном у реальному часі.

2.4 Вибір апаратного забезпечення для системи комп'ютерного зору

Побудова системи комп'ютерного зору для безпілотного літального апарата вимагає ретельного підбору апаратного забезпечення, яке поєднує високу енергоефективність, достатню обчислювальну потужність та компактні габарити. У даному підрозділі здійснюється порівняльний аналіз кількох можливих варіантів обчислювальних платформ, що можуть бути використані для реалізації системи, а також обґрунтовується вибір оптимального рішення для створення прототипу.

Одним із найпоширеніших і найдоступніших рішень для керування малими робототехнічними системами є платформи на базі мікроконтролерів, такі як Arduino Uno, Arduino Mega, ESP32 тощо. Вони характеризуються низьким енергоспоживанням, простотою програмування та великою кількістю сумісних модулів (датчики, контролери двигунів, системи телеметрії).

Однак ключовим недоліком мікроконтролерів у контексті задач комп'ютерного зору є відсутність повноцінного процесора із підтримкою операційної системи та багатопотокової обробки. Arduino не має достатнього обсягу оперативної пам'яті (типово 2–8 КБ SRAM) і не підтримує роботу з великими масивами зображень, які вимагають сотень мегабайт пам'яті для

зберігання та аналізу відеопотоку в реальному часі. Крім того, у мікроконтролерах відсутній графічний прискорювач (GPU), що є критично важливим для виконання згорткових нейронних мереж.

Таким чином, Arduino може бути використано лише як допоміжний елемент у загальній системі — наприклад, для керування електронікою безпілотника (ESC, серводвигуни, телеметрія), але не як основна платформа для виконання алгоритмів комп'ютерного зору.

Альтернативним варіантом є використання одноплатних комп'ютерів, що поєднують компактність мікроконтролерів із функціональністю повноцінного комп'ютера. Серед таких пристроїв виділяється Orange Pi 5, побудований на базі процесора Rockchip RK3588S (8-ядерний ARM Cortex-A76/A55, GPU Mali-G610 MP4) із підтримкою до 32 ГБ оперативної пам'яті LPDDR4.

Orange Pi 5 має чудові показники продуктивності, можливість апаратного декодування відео до 8K, підтримку інтерфейсів MIPI-CSI для підключення камер, USB 3.0, Ethernet, Wi-Fi та Bluetooth. Завдяки потужному GPU він може виконувати інференс моделей глибокого навчання з прийнятною швидкістю навіть без зовнішнього прискорювача.

У контексті системи комп'ютерного зору для безпілотного апарата Orange Pi 5 є потенційно придатною платформою. Проте, незважаючи на високу продуктивність, цей пристрій поступається Raspberry Pi за рівнем програмної підтримки, кількістю готових бібліотек і сумісних модулів. Система розробки під Orange Pi є менш стабільною, а документація часто фрагментарна, що ускладнює швидке створення робочого прототипу. Таким чином, Orange Pi 5 можна розглядати як гідну альтернативу, однак для науково-дослідницького прототипу доцільніше обрати більш підтримувану та популярну платформу.

З огляду на зазначені фактори, для реалізації системи комп'ютерного зору було обрано Raspberry Pi 5 — найновішу версію популярної серії одноплатних комп'ютерів.

Цей пристрій побудований на базі процесора Broadcom BCM2712 (4 ядра ARM Cortex-A76, 2.4 ГГц), має GPU VideoCore VII, підтримує до 16 ГБ

LPDDR4X-4267 оперативної пам’яті та оснащений інтерфейсами CSI/DSI, USB 3.0, PCIe 2.0 для підключення зовнішніх прискорювачів [20].

Основні переваги Raspberry Pi 5 у контексті даної роботи:

— велика екосистема програмного забезпечення, повна підтримка бібліотек OpenCV, NumPy, PyTorch, TensorFlow Lite та інших компонентів, що є критичними для розробки систем комп’ютерного зору;

— апаратне прискорення завдяки GPU VideoCore VII забезпечує значне прискорення графічних операцій і базових обчислень згорток, що дозволяє обробляти відеопотік у реальному часі;

— підтримка операційної системи Raspberry Pi OS (Bookworm), оптимізованої під ARM64-процесори з розширенням NEON, яке ефективно використовується у бібліотеках OpenCV;

— наявність великої кількості прикладів, відкритих проєктів і готових драйверів для камер (IMX219, IMX477 тощо) значно спрощує інтеграцію;

— Raspberry Pi 5 забезпечує достатню обчислювальну потужність для виконання моделей типу YOLOv8/YOLOv11n при спрощеній архітектурі, споживаючи при цьому мінімум енергії.

Для порівняння варто розглянути і потужніші варіанти — міні-ПК на базі процесорів AMD Ryzen або Intel Core, наприклад GenMachine Mini PC з процесором AMD Ryzen 3 4300U (2.7–3.7 ГГц). Такі пристрої мають у декілька разів вищу продуктивність CPU і GPU, більший обсяг пам’яті (8–16 ГБ RAM) та повну сумісність із десктопними ОС (Windows 10/11, Ubuntu). Вони здатні забезпечити виконання навіть великих моделей типу YOLOv8m або DeepSORT у реальному часі з FPS понад 60, що робить їх ідеальними для промислових застосувань або лабораторних експериментів.

Однак такі рішення не відповідають концепції енергоефективного та легкого вбудованого пристрою, який може бути інтегрований у FPV-дрон або подібну платформу. Крім того, вартість міні-ПК є десь у 2 рази більшою за ціну Raspberry Pi 5, що робить їх економічно недоцільними для серійного виробництва систем такого типу. Таким чином, вибір на користь Raspberry Pi 5

обумовлений концепцією дослідження — розробкою системи в умовах обмежених апаратних ресурсів, що наближено до реальних сценаріїв використання у недорогих безпілотних системах. Порівняльний аналіз можна побачити на таблиці 2.1.

Таблиця 2.1 — Порівняльний аналіз апаратних засобів

Платформа	Продуктивність	Енергоспоживання	Сумісність із бібліотеками CV/AI	Орієнтовна ціна	Доцільність використання
Arduino Uno / Mega	Дуже низька	Дуже низьке	Немає підтримки OpenCV/NN	<10 USD	Тільки допоміжні задачі
Orange Pi 5	Висока	Середнє	Часткова підтримка, нестабільна	80 USD	Альтернатива, але не основна
Raspberry Pi 5	Оптимальна	Низьке	Повна підтримка CV/AI	80 USD	Основна платформа
GenMachine Mini PC (Ryzen 3 4300U)	Дуже висока	Високе	Повна	>150 USD	Недоцільна для fpv камікадзе

Виходячи з отриманих результатів, Raspberry Pi 5 забезпечує найкращий компроміс між вартістю, енергоспоживанням, продуктивністю та доступністю програмного забезпечення. Саме тому вона була обрана як базова обчислювальна платформа для створення та тестування системи комп’ютерного зору в даному магістерському проєкті.

2.5 Оптимізація роботи моделей нейронних мереж на обмежених пристроях

Застосування нейронних мереж у системах комп’ютерного зору, що функціонують у режимі реального часу, потребує високих обчислювальних ресурсів. Проте для систем, які реалізуються на одноплатних комп’ютерах типу Raspberry Pi, постає завдання адаптації моделей до обмежених апаратних

можливостей. Тому оптимізація моделей нейронних мереж є критично важливою частиною розробки ефективного програмно-апаратного комплексу.

Raspberry Pi 5, хоча й є найпотужнішою моделлю серії, має обмежену кількість обчислювальних ядер і порівняно слабкий GPU VideoCore VII, який не забезпечує рівня прискорення, характерного для повноцінних графічних процесорів NVIDIA.

Крім того, обсяг оперативної пам’яті не дозволяє одночасно завантажувати великі нейронні моделі, вести буферизацію відеопотоку високої роздільної здатності й виконувати інші задачі керування дроном.

Без оптимізації модель типу YOLOv11 або YOLOv8 із 30+ млн параметрів може працювати зі швидкістю менше 5 кадрів за секунду, що є неприйнятним для системи навігації або розпізнавання цілей. Саме тому важливо застосувати низку методів, які зменшують розмір моделі, скорочують обчислення та знижують вимоги до пам’яті [21].

Найефективнішими засобами оптимізації є ті, що змінюють внутрішню структуру або представлення моделі.

Квантизація — це зменшення точності чисел, які зберігають ваги та активації мережі. Наприклад, перехід із 32-бітної точності (float32) на 8-бітну (int8) дозволяє зменшити розмір моделі у 4 рази й суттєво пришвидшити виконання, при цьому втрата точності детекції зазвичай не перевищує 1–3 %. TensorFlow Lite підтримує кілька режимів квантизації (динамічну, пост-тренувальну, змішану), що дозволяє підібрати компроміс між швидкодією та точністю.

Прунінг — це видалення нейронних зв’язків або фільтрів із незначним впливом на кінцевий результат. Унаслідок прюінгу мережа стає рідшою, зменшуються розміри матриць згортки, що полегшує подальшу компресію. Поєднання прюінгу та квантизації може скоротити модель YOLO на 50–70 % без помітної втрати mAP (середньої точності детекції).

Для систем із обмеженими ресурсами доречно обирати малі варіанти моделей — YOLOv8n, YOLOv11n, MobileNetV3, EfficientDet-Lite тощо. У даній

роботі використовується YOLOv11n, яка забезпечує хорошу якість розпізнавання при мінімальних апаратних вимогах. Для подальшого пришвидшення можливо застосувати скорочення кількості каналів або шарів у конфігурації моделі.

Програмно-апаратна оптимізація, навіть після внутрішньої оптимізації ваг важливу роль відіграє те, на якому рушії виконується модель.

NCNN — це високоефективний inference-рушій, розроблений компанією Tencent для мобільних платформ [22]. NCNN оптимізований під архітектуру ARM і використовує SIMD-інструкції, що дозволяє виконувати згортки значно швидше, ніж у звичайних фреймворках. У подальшому проєкті NCNN може бути використаний як експериментальний варіант для порівняння швидкодії з TensorFlow Lite.

OpenVINO — це набір інструментів розроблений Intel для прискорення виконання моделей. Хоча Raspberry Pi не має процесорів Intel, OpenVINO має ARM порт, який підтримує виконання оптимізованих моделей у форматі IR. Цей підхід особливо ефективний для зменшення часу завантаження та оптимізації графу обчислень.

Застосування цих рушіїв дозволяє досягти швидкості 15–25 FPS для спрощених моделей YOLO на Raspberry Pi 5, що вже відповідає вимогам систем реального часу.

Окрім оптимізації самої нейронної мережі, важливим є зменшення навантаження на систему за рахунок ефективнішої обробки відеопотоку.

Серед основних методів:

— зменшення роздільності кадрів, обробка зображення 640×480 потребує у 4 рази менше обчислень, ніж 1280×960, при цьому вплив на точність детекції незначний;

— замість повного кадру аналізується лише центральна область, де ймовірніше розташована ціль;

— обробка не кожного кадру, для стабільних сцен можна аналізувати, наприклад, кожен 5 кадр, використовуючи трекер для проміжних кадрів;

Такі підходи не лише підвищують швидкодію, але й знижують теплове навантаження та енергоспоживання пристрою.

Застосування комбінованих методів оптимізації дозволяє істотно покращити ефективність роботи моделі. Результати порівняння ефективності оптимізаційних підходів наведено в таблиці 2.2.

Таблиця 2.2 ― Теоретичне порівняння впливу різних методів оптимізації на швидкодію моделе

Метод оптимізації	Ефект	Орієнтовне прискорення (разів)
Квантизація до int8	Зменшення розміру моделі в 4 рази	1.5–2
Прюінг 50 % ваг	Менше обчислень на 40–60 %	1.3
Використання TensorFlow Lite	Менше накладних витрат	1.5
Обробка кадрів 640×480	Зменшення обчислень у 4 рази	2–3

У сукупності розглянуті підходи дозволяють оптимізувати швидкодію до п’яти разів порівняно з базовим варіантом без оптимізації, що дозволяє досягти продуктивності від двадцяти п’яти до тридцяти п’яти кадрів за секунду навіть на Raspberry Pi 5. Такий рівень швидкодії робить систему придатною для використання на безпілотних апаратах у режимі реального часу.

2.6 Конвертація моделей YOLO для використання у форматах NCNN, TensorRT, OpenVINO

Одним із ключових етапів адаптації системи комп’ютерного зору до апаратних обмежень є перетворення навченої моделі нейронної мережі у формат, оптимізований під конкретну платформу. Для реалізації системи на базі Raspberry Pi 5 доцільно обрати такі програмні рішення, які забезпечують баланс між швидкодією та компактністю. Серед популярних підходів до оптимізації та прискорення інференсу моделей найбільш поширені формати NCNN, TensorRT та OpenVINO.

Процес перетворення можна подати у вигляді функціонального відображення формула 2.1 .

$$\mathcal{M}_{opt}=\Phi(\mathcal{M}_{src}, \mathcal{T}, \mathcal{Q}, \mathcal{H}), \quad (2.1)$$

де, $\mathcal{M}_{src}$— початкова модель;

$\mathcal{T}$— набір оптимізацій;

$\mathcal{Q}$— квантизація;

$\mathcal{H}$— апаратна ціль виконання.

Результатом є оптимізована модель $\mathcal{M}_{opt}$, що характеризується зменшенням часу обробки кадру.

Ефективність конвертації оцінюється коефіцієнтом прискорення 2.2 .

$$S = \frac{t_{src}}{t_{opt}}, \quad (2.2)$$

Це демонструє виграш у продуктивності при збереженні прийнятного рівня точності.

NCNN — це високопродуктивний фреймворк для виконання нейронних мереж, розроблений інженерами компанії Tencent. Він орієнтований насамперед на мобільні та вбудовані ARM-пристрої і є повністю автономним для роботи не потребує залежностей від Python чи бібліотек типу CUDA чи OpenCL. Саме тому NCNN є оптимальним вибором для використання на одноплатних комп’ютерах, таких як Raspberry Pi або Orange Pi [23].

Основні переваги NCNN [24]:

— легковаговість і незалежність — код ядра реалізований на C++ без зовнішніх залежностей;

— оптимізація під ARM-архітектуру (використання SIMD-інструкцій NEON);

— підтримка Vulkan API, що дозволяє використовувати GPU для інференсу, якщо він доступний;

— підтримка моделей у форматі YOLOv5/8/11 через ONNX-конвертацію або прямий експорт із Ultralytics.

Процес експорту моделі YOLO у формат NCNN значно спрощений у сучасних версіях бібліотеки Ultralytics. Достатньо виконати кілька команд у Python-середовищі лістинг 2.1 .

Лістинг 2.1 — Експорт моделі YOLOv11n у формат NCNN

```
from ultralytics import YOLO
# Завантаження базової моделі
model = YOLO("yolo11n.pt")
# Експорт у формат NCNN
model.export(format="ncnn")
```

У результаті створюється директорія /yolo11n_ncnn_model/, що містить файли

```
model.param
model.bin
metadata.yaml
README.txt
```

Файли .param та .bin є основними компонентами NCNN-моделі, перший описує архітектуру нейронної мережі (топологію шарів), а другий — вагові коефіцієнти у двійковому вигляді.

У процесі експорту Ultralytics автоматично виконує внутрішню конвертацію з PyTorch → ONNX → NCNN, одночасно застосовуючи базові оптимізації графа (злиття згорток із активаціями, видалення зайвих шарів, обрізання порожніх тензорів).

Таким чином, користувач отримує готову до використання модель, сумісну з будь-яким проєктом на C++ або Python із підтримкою NCNN API.

Для інтеграції у власний проєкт достатньо використати стандартні функції NCNN, приклад чого наведено в лістингу 2.2, де продемонстровано базовий підхід до завантаження моделі та виконання інференсу.

Лістинг 2.2 — Приклад стандартних функції NCNN

```
#include "net.h"
ncnn::Net net;
```

```
net.load_param("model.param");
net.load_model("model.bin");
ncnn::Mat input = ncnn::Mat::from_pixels_resize(frame.data, ncnn::Mat::PIXEL_BGR, w, h, 640, 640);
ncnn::Extractor ex = net.create_extractor();
ex.input("images", input);
ncnn::Mat output;
ex.extract("output", output);
```

Таким чином, розробник отримує можливість виконувати детекцію цілей у реальному часі без необхідності встановлення TensorFlow або PyTorch, що особливо важливо для Raspberry Pi. При тестуванні моделей YOLOv8n та YOLOv11n на пристроях ARM Cortex-A76 (аналогічних до CPU Raspberry Pi) досягається швидкість 20–30 кадрів/с при 640×480 з апаратним Vulkan-прискоренням.

TensorRT — це бібліотека високопродуктивного інференсу, розроблена компанією NVIDIA. Вона орієнтована на використання GPU і спеціалізованих тензорних ядер (Tensor Cores) у графічних процесорах серії RTX, Jetson та Tesla.

Основна мета TensorRT — максимальне прискорення виконання нейронних мереж без суттєвого зниження точності.

До складу TensorRT входять такі етапи оптимізації:

— Layer Fusion — об’єднання кількох операцій в один GPU-керований блок;

— Precision Calibration — автоматичний вибір оптимальної розрядності (FP32, FP16, INT8);

— Dynamic Tensor Memory — динамічне управління пам’яттю під час виконання;

— Kernel Auto-Tuning — автоматичний підбір найшвидших реалізацій обчислень для конкретного GPU.

Конвертація виконується через утиліту trtexec або Python-скрипт з використанням бібліотеки tensorrt: trtexec, --onnx=yolov11n.onnx--, saveEngine=yolov11n_fp16.engine, --fp16. Отриманий файл .engine містить оптимізований граф моделі, який може виконуватись безпосередньо на пристроях Jetson Nano, Orin або Xavier. TensorRT у даному проєкті розглядається

як еталон високопродуктивного рушія для порівняння результатів інференсу, оскільки Raspberry Pi не має GPU NVIDIA, але оцінка різниці у швидкодії дозволяє сформувати уявлення про потенційний приріст при використанні прискорювачів.

OpenVINO (Open Visual Inference and Neural Network Optimization) — це набір інструментів від компанії Intel, призначений для оптимізації та виконання нейронних мереж на пристроях з процесорами Intel, а також на зовнішніх прискорювачах типу Myriad X (Intel Movidius).

Переваги OpenVINO:

— підтримка широкого спектра архітектур (CPU, iGPU, FPGA, VPU);

— ефективна оптимізація графа обчислень;

— можливість автоматичного розподілу навантаження між різними пристроями.

Процес конвертації починається з використання утиліти Model Optimizer mo --input_model yolov11n.onnx --data_type FP16 --output_dir model_openvino.

На виході отримуються файли .xml (опис моделі) і .bin (ваги). Ці файли потім використовуються у виконувальному рушії OpenVINO Runtime, який забезпечує інференс з високою швидкодією навіть на CPU без дискретної відеокарти.

У контексті розроблюваної системи OpenVINO є цікавим напрямом для розширення сумісності. Зокрема, при використанні міні-ПК на базі процесорів Intel (наприклад, серії GenMachine) саме цей фреймворк дозволяє отримати найкраще співвідношення між швидкістю та стабільністю роботи. Порівняльну характеристику приведених вище підходів можна побачити на таблиці 2.3. Крім того, OpenVINO надає зручні інструменти для оптимізації моделей, що спрощує їх адаптацію до обмежених ресурсів вбудованих систем. Крім того, можливість конвертації моделей у формат IR спрощує інтеграцію з іншими компонентами системи та підвищує її масштабованість для різних типів БПЛА.

Таблиця 2.3 — Теоретичне порівняння фреймворків для виконання нейронних мереж

Формат	Основна цільова платформа	Підтримка GPU	Переваги	Обмеження
NCNN	ARM, мобільні процесори	Vulkan	Мінімальні вимоги, висока швидкість на CPU	Обмежена точність, неповна підтримка всіх операторів
TensorRT	NVIDIA GPU	CUDA, Tensor Cores	Максимальна продуктивність, підтримка FP16/INT8	Працює лише на пристроях з GPU NVIDIA
OpenVINO	Intel CPU/VPU	iGPU, Myriad X	Універсальність, ефективність навіть без GPU	Найкраще працює лише на апаратурі Intel

Таким чином, для цілей даної магістерської роботи найдоцільнішим є використання формату NCNN, оскільки він забезпечує стабільну роботу на платформі Raspberry Pi 5, не потребує зовнішніх бібліотек і підтримує базові можливості прискорення через Vulkan.

TensorRT та OpenVINO розглядаються як альтернативні напрямки оптимізації, які можуть бути використані на етапах масштабування або перенесення системи на інші апаратні архітектури. Застосування таких форматів конвертації дозволяє забезпечити реальний баланс між швидкістю, точністю та споживанням енергії, що є критично важливим для систем комп’ютерного зору, інтегрованих у безпілотні літальні апарати.

2.7 Архітектурно-алгоритмічна модель системи комп’ютерного зору

Розроблена система комп’ютерного зору базується на модульній архітектурі, що забезпечує гнучкість, масштабованість і простоту адаптації до різних сценаріїв використання безпілотних літальних апаратів (БПЛА). Її структура включає апаратну частину — цифрову камеру, обчислювальний модуль Raspberry Pi 5, комунікаційні інтерфейси та програмне забезпечення, побудоване на бібліотеках OpenCV, PyMAVLink, а також рушіях YOLO (для детекції об’єктів), MOSSE (для трекінгу) та NCNN (для оптимізованого виконання нейронних моделей на обмежених апаратних ресурсах).

Система підтримує два режими функціонування:

— ручний — оператор самостійно вибирає ціль через графічний інтерфейс, після чого активується лише трекер;

— автоматичний — розпізнавання, супровід і передача координат відбуваються автономно на основі алгоритмів штучного інтелекту.

Архітектура побудована за принципом багаторівневої модульної взаємодії, де кожен компонент виконує власну функцію та працює у окремому потоці. Це дозволяє мінімізувати затримки обробки й підтримувати стабільну частоту від двадцяти до двадцяти п’яти кадрів за секунду.

Основні модулі системи описані нижче.

Модуль відеозахоплення — отримує відеопотік у реальному часі з камери, підключеної через інтерфейс CSI (Camera Serial Interface). Використання цього інтерфейсу забезпечує мінімальну затримку (<10 мс) у порівнянні з USB-камерами.

Модуль попередньої обробки — виконує нормалізацію яскравості, зменшення шуму, компенсацію вібрацій та стабілізацію зображення за допомогою фільтрів Gaussian Blur, Bilateral Filter, а також корекції контрасту.

Модуль детекції — реалізує виявлення об’єктів за допомогою оптимізованої нейронної моделі YOLOv11n, що працює через рушій NCNN.

Модуль трекінгу — алгоритми MOSSE або ByteTrack забезпечують стабільний супровід цілі між кадрами, компенсуючи вібрації та короткочасні втрати об’єкта.

Модуль керування — перетворює координати цілі на сигнали керування й передає їх польотному контролеру через протокол MAVLink із використанням бібліотеки PyMAVLink.

Інтерфейс оператора — відображає відеопотік з накладеними рамками детекції, прицілом та службовими індикаторами, дозволяючи перемикати режими та контролювати стан системи.

Така структура забезпечує незалежність модулів, спрощує оновлення алгоритмів і дозволяє інтегрувати додаткові сенсори — наприклад, інфрачервоні або тепловізійні камери для роботи в умовах низької освітленості. Проте робота в умовах туману або дощу залишається обмеженою через фізичні властивості оптичних сенсорів.

Алгоритмічна модель визначає логіку функціонування системи комп’ютерного зору в режимі реального часу (додаток Е). Робота системи організована як замкнутий цикл обробки відеопотоку, який повторюється від двадцяти до тридцяти разів за секунду та включає описані нижче етапи.

Ініціалізація системи — завантаження моделей YOLO та MOSSE, налаштування параметрів відеопотоку, відкриття послідовного порту зв’язку та запуск допоміжних потоків.

Захоплення кадру — отримання зображення з камери у форматі YUV та перетворення у формат BGR для подальшої обробки.

Попередня обробка — фільтрація шумів, нормалізація контрасту, масштабування кадру до розміру, прийнятного для моделі.

Детекція об’єктів — нейронна модель YOLOv11n виконує виявлення об’єктів і повертає координати цілей із рівнем впевненості.

Трекінг — алгоритм MOSSE підтримує безперервність спостереження за ціллю між кадрами.

Аналіз і прийняття рішення — система визначає пріоритетну ціль та обчислює її відхилення від центру кадру.

Формування команд керування — координати цілі перетворюються на сигнали регулювання по осях yaw/pitch і передаються польотному контролеру через MAVLink.

Візуалізація — результати детекції, трекінгу та стан системи виводяться оператору в реальному часі.

Зворотний цикл — після завершення обробки кадру система повертається до кроку 2, забезпечуючи безперервність процесу.

Запропонована архітектурно-алгоритмічна модель забезпечує баланс між точністю нейронної детекції, швидкодією та енергоефективністю, що є критичним для застосувань на Raspberry Pi 5.

Завдяки модульному підходу система легко масштабується — від лабораторних експериментих зразків до інтеграції в реальні FPV-платформи. Її структура створює основу для подальшого розвитку системи, зокрема для впровадження адаптивних методів стабілізації, мультисенсорної інтеграції та розширених режимів автономного наведення.

3 РЕАЛІЗАЦІЯ СИСТЕМИ КОМП'ЮТЕРНОГО ЗОРУ ДЛЯ РОЗПІЗНАВАННЯ ЦІЛЕЙ БЕЗПІЛОТНИХ АПАРАТІВ

3.1 Структура програмного комплексу

Розроблена система комп'ютерного зору призначена для виявлення та супроводу цілей у реальному часі на борту безпілотного літального апарата (БпЛА). Програмна частина реалізована на мові Python 3.11 та працює в операційній системі Raspberry Pi OS Bookworm, встановленої на Raspberry Pi 5 з 8 ГБ оперативної пам'яті. Основна мета проєкту — забезпечити стабільну роботу алгоритмів комп'ютерного зору в умовах обмежених апаратних ресурсів та з мінімальними затримками в обробці відеопотоку, паралельно з підтримкою стабільного зв'язку з політним контролером через MSP-протокол.

Ключові бібліотеки, що утворюють технологічний стек системи:

— OpenCV версії 4.8+;

— Ultralytics YOLOv11;

— PySerial 3.5+;

— NumPy 1.24+;

— Вбудовані бібліотеки threading та queue.

OpenCV (Open Source Computer Vision Library) версії 4.8+ — фундаментальна бібліотека для обробки зображень, що забезпечує захоплення відеопотоку, базову обробку кадрів, реалізацію класичних алгоритмів відстеження об'єктів. Вибір саме цієї версії обумовлений підтримкою алгоритму MOSSE (Minimum Output Sum of Squared Error) через модуль cv2.legacy.TrackerMOSSE_create(), який демонструє оптимальну продуктивність для вбудованих систем.

Ultralytics YOLOv11 — сучасна архітектура нейромережі для детекції об'єктів, що відрізняється високою швидкістю інференсу при збереженні прийнятної точності. Використання саме YOLOv11, а не попередніх версій, обґрунтоване покращеною оптимізацією для вбудованих систем та зниженими вимогами до оперативної пам'яті.

PySerial 3.5+ — бібліотека для роботи з послідовними портами, що забезпечує стабільний зв'язок з політним контролером через MSP-протокол. Особливістю використання є підтримка асинхронного читання/запису з таймаутами, що критично важливо для систем реального часу.

NumPy 1.24+ — ефективна бібліотека для математичних операцій над багатовимірними масивами, що інтенсивно використовується при обробці зображень та обчисленні матриць перетворень.

Вбудовані бібліотеки threading та queue — для організації паралельних потоків виконання та реалізації пріоритетних черг команд.

Система побудована за принципом модульності з чітким розподілом функціональних обов'язків місти сімома основними модулями. Така організація дозволяє легко тестувати окремі компоненти, модифікувати функціональність без впливу на інші частини системи та повторно використовувати модулі в подальших розробках.

Головний модуль управління (main.py) є точкою входу в систему та відповідає за ініціалізацію всіх компонентів, запуск паралельних потоків виконання та координоване завершення роботи. Основним об'єктом модуля є клас CVIntegration, що інстанціюється як cv_controller. Модуль реалізує патерн "Фасад", надаючи спрощений інтерфейс для взаємодії з усією системою.

Особливістю реалізації є використання механізму подій (threading.Event) для синхронізації потоків. Об'єкт stop_command слугує сигналом для безпечного завершення всіх потоків при отриманні команди від користувача приклад коду представлений у лістингу 3.1.

Лістинг 3.1 механізм подій (threading.Event)

```
stop_command = threading.Event()
input('Press enter to stop process...\n')
messages.display(messages.main_stopping_threads)
cv_controller.stop()
stop_command.set()
```

Модуль маршрутизації команд (router.py) реалізує архітектурний патерн "Команда" для інкапсуляції запитів на виконання дій. Кожна команда

представлена екземпляром класу Command, що містить три атрибути: пріоритет виконання (ціле число від 0 до 5), ім'я команди (рядковий ідентифікатор) та тіло команди з параметрами (словник Python).

Особливістю реалізації є перевантаження оператора порівняння __lt__ для автоматичного сортування команд за пріоритетом у черзі PriorityQueue, представлений у лістингу 3.2.

Лістинг 3.2 Перевантаження оператора порівняння __lt__

```
class Command:
    def __init__(self, priority, name, body):
        self.priority = priority
        self.name = name
        self.body = body
    def __lt__(self, other):
        return self.priority < other.priority
```

Функція command_executor реалізує нескінченний цикл обробки команд з черги, що працює в окремому потоці. Важливим аспектом є використання небло куючого читання з черги з таймаутом 50 мілісекунд, що дозволяє потоку оперативно реагувати на сигнал завершення stop_command, представлений у лістингу 3.3.

Лістинг 3.3 цикл обробки команд з черги

```
while not stop_command.is_set():
    try:
        command = command_queue.get(timeout=0.05)
        execute_command(command)
        command_queue.task_done()
        time.sleep(0.01)
    except queue.Empty:
        time.sleep(0.02)
```

Модуль комп'ютерного зору (cv_integration.py) є найбільш складним та ресурсоємним компонентом системи. Він реалізує клас CVIntegration, що інкапсулює всю логіку обробки відеопотоку, детекції цілей, їх відстеження та формування керуючих сигналів. Модуль працює з частотою до 30 кадрів за секунду та потребує найбільшої частки обчислювальних ресурсів Raspberry Pi 5.

Архітектура модуля побудована на принципі скінченного автомата з чотирма основними станами: очікування, ручне відстеження, автоматична детекція та автономне відстеження. Переходи між станами здійснюються на основі значень AUX-каналів, що зчитуються з політного контролера.

Модуль комунікації з авіонікою (commands.py, msp_helper.py) забезпечує двосторонній обмін даними з політним контролером через MSP-протокол. Модуль commands.py містить високорівневі функції для відправки команд та читання телеметрії, тоді як msp_helper.py реалізує низькорівневу логіку формування та парсингу MSP-пакетів.

Ключовою функцією модуля є set_row_rc(), яка формує та відправляє ком анду MSP_SET_RAW_RC для встановлення значень RC-каналів, представлений у лістингу 3.4.

Лістинг 3.4 функцією set_row_rc

```
def set_row_rc(roll, pitch, yaw, throttle, servo_aux):
    data = [roll, pitch, throttle, yaw, 0, servo_aux, 0, 0]
    msp.send_msp_command(serial_port, msp.MSP_SET_RAW_RC, data)
```

Модуль управління станом (autopilot.py) зберігає глобальний стан системи у вигляді словника state, що містить понад 20 параметрів, включаючи поточні зн ачення RC-каналів, телеметричні дані (висота, швидкість, напруга батареї), стат ус відстеження цілі та режим роботи автопілота. Словник є спільним ресурсом д ля всіх потоків системи, що вимагає ретельного проектування механізмів синхр онізації доступу.

Модуль телеметрії (telemetry.py) реалізує регулярне опитування політного контролера для отримання актуальних даних про стан дрона. Модуль працює в о кремому потоці з частотою 10 Гц для каналів керування (команда MSP_RC) та 2 Гц для інших параметрів (висота, швидкість, стан батареї). Така різниця в часто тах обумовлена різною критичністю даних для роботи системи.

Допоміжні модулі (messages.py, logger.py, definitions.py) реалізують серві сну функціональність, логування подій у файл та консоль, збереження констант

(номери MSP-команд, значення RC-каналів за замовчуванням) та форматування повідомлень для виводу.

Для забезпечення роботи в реальному часі система використовує чотири незалежні потоки виконання, кожен з яких відповідає за певний аспект функціо нальності.

Потік обробки відеопотоку є найбільш навантаженим обчислювально. Він реалізований у методі _main_loop() класу CVIntegration та працює з частотою, що визначається властивістю frame_delay. Основний цикл включає такі етапи: захоплення кадру з відеоджерела, попередня обробка зображення, аналіз стану AUX-каналів, виконання детекції або відстеження (залежно від режиму), візуалі зація результатів та формування керуючих команд.

Потік виконавця команд (command_executor) обробляє чергу пріоритетних команд. Його основним завданням є своєчасне виконання команд, що генерують ся іншими компонентами системи, особливо критичних команд керування дрон ом. Потік реалізує механізм обробки з таймаутом, що дозволяє балансувати між оперативністю реакції та ефективним використанням процесорного часу.

Потік збору телеметрії (telemetry_requestor) регулярно опитує політний контролер через MSP-протокол. Частота опитування різних параметрів оптимізо вана з урахуванням їх критичності канали керування опитуються з частотою 10 Гц для забезпечення мінімальної затримки реакції на дії оператора, тоді як мен ш критичні параметри (висота, швидкість) опитуються з частотою 2 Гц.

Потік високорівневої логіки автопілота (empty_pilot_process) виконує стра тегічні завдання, такі контроль виконання місії, моніторинг стану системи та прийняття рішень на основі поточного контексту. Цей потік працює з найнижчо ю частотою (0.5 Гц) та реалізує логіку, що не вимагає реакції в мілісекундному діапазоні.

Ефективна робота багатопоточної системи вимагає ретельно спроектовани х механізмів синхронізації та обміну даними між компонентами. У розробленій системі використано три основних механізми.

Глобальний словник стану — централізоване сховище даних, доступне з усіх потоків. Словник autopilot.state містить як телеметричні дані (висота, швидкість, значення RC-каналів), так і статусні змінні системи (режим роботи, стан відстеження цілі). Для запобігання race conditions при одночасному доступі з різних потоків, оновлення значень відбувається атомарно — кожне значення оновлюється окремою операцією присвоювання, що є атомарною в Python.

Черга пріоритетних команд — механізм асинхронної передачі завдань між модулями. Коли один модуль (наприклад, модуль комп'ютерного зору) потребує виконання дії іншим модулем (наприклад, відправки команди керування), він формує об'єкт Command та поміщає його в чергу command_queue. Потік викона вця команд асинхронно обробляє чергу, виконуючи команди відповідно до їх пріоритету.

Події синхронізації — об'єкти типу threading. Event для координованого у правління життєвим циклом потоків. Найважливішим є об'єкт stop_command, який використовується для сигналізації всім потокам про необхідність завершен ня роботи. Коли користувач натискає Enter в головному потоці, встановлюється прапорець події, і всі робочі потоки завершують свої цикли при найближчій перевірці стану події.

Розробка програмного забезпечення для Raspberry Pi 5 вимагала врахуван ня обмежень платформи, обмежена обчислювальна потужність (чотириядерний процесор ARM Cortex-A76), обмежений обсяг оперативної пам'яті (8 ГБ), наявність лише одного послідовного порту для комунікації. Для подолання цих обмежень у системі реалізовано ряд оптимізацій.

Адаптивна якість обробки — система динамічно регулює параметри обробки залежно від навантаження. Наприклад, при зниженні частоти кадрів нижче порогового значення збільшується інтервал між циклами детекції YOLO або зменшується розмір області інтересу для обробки.

Кешування та перевикористання ресурсів — об'єкти, що створюються з великими витратами (наприклад, модель YOLO), ініціалізуються один раз при

запуску системи та використовуються повторно. Буфери для обробки зображень також перевикористовуються для мінімізації виділень пам'яті.

Локальність даних — мінімізація передачі великих об'єктів (кадрів зображ ення) між потоками. Кожен потік працює з локальними копіями даних, що необхідні для його функціональності, зменшуючи витрати на синхронізацію.

Ефективне використання портів вводу-виводу — послідовний порт викор истовується одночасно для читання телеметрії та відправки команд керування за рахунок мультиплексування запитів у часі. Частота відправки команд обмежу ється для уникнення перевантаження каналу зв'язку.

Архітектурні рішення, прийняті при розробці програмного комплексу, дозволили створити систему, здатну стабільно працювати в умовах реального часу на обмежених обчислювальних ресурсах Raspberry Pi 5, забезпечуючи при цьому всі необхідні функції детекції, відстеження та керування БПЛА.

3.2 Навчання моделі YOLO

3.2.1 Підготовка та формування кастомного датасету

Для навчання моделі детекції об’єктів у межах розробленої системи комп’ютерного зору було сформовано кастомний датасет, який поєднує дані з двох джерел:

— відкритого набору VisDrone (Visual Object Detection in Drone Imagery Dataset);

— та набору Vehicles з платформи Roboflow Universe.

Обидва набори містять зображення, отримані з висоти польоту безпілотних літальних апаратів, що забезпечує відповідність умов навчання до реального середовища роботи системи. На рисунку 3.1 демонстрація обраних датасетів. Поєднання цих джерел дозволило збільшити різноманітність сцен, кутів зйомки та типів об’єктів, що позитивно вплинуло на здатність моделі до узагальнення. Крім того, використання різних умов освітлення та динамічних фонів підвищило стійкість моделі до реальних експлуатаційних ситуацій.

Рисунок 3.1 — Приклади двох датасетів VisDrone та Vehicles

Метою об’єднання стало розширення варіативності даних (різні кути огляду, освітлення, фони, масштаб об’єктів) та збільшення обсягу вибірки для підвищення узагальнювальної здатності моделі YOLO.

Для зручності подальшого використання у фреймворках YOLOv8/YOLOv11 сформовано стандартну структуру директорій можна побачити на рисунку 3.2.

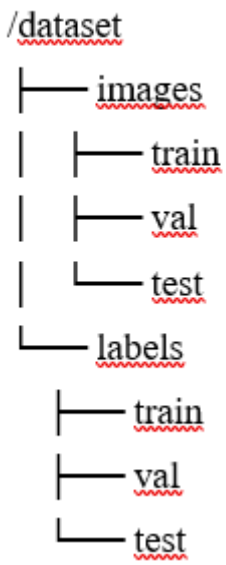

Рисунок 3.2 — Стандартну структуру директорій в датасеті Yolo формату

Кожне зображення має відповідний .txt-файл з анотаціями у форматі YOLO: (<class_id>, <x_center>, <y_center>, <width>, <height>). Де всі координати нормалізовані відносно розмірів зображення (у діапазоні 0–1). Після завантаження обох датасетів усі зображення були конвертовані до єдиного формату .jpg, а файли розмітки приведено до однакового формату анотацій YOLO.

Під час об'єднання застосовано наступні кроки:

— нормалізація назв файлів, щоб уникнути конфліктів у назвах (додано префікси vis_ та roboflow_);

— об'єднання папок train, val і test з обох джерел;

— автоматична перевірка наявності відповідних .txt файлів для кожного зображення;

— фільтрація зображень із пошкодженими або порожніми розмітками.

Оскільки в межах даного проєкту система комп'ютерного зору має виявляти лише ціль (без поділу на типи транспортних засобів чи інші об'єкти), було прийнято рішення звести всі класи до одного — "target".

Це дозволяє суттєво спростити процес навчання, зменшити кількість параметрів моделі та підвищити стабільність розпізнавання у режимі реального часу.

Для виконання перетворення було розроблено Python-скрипт, який автоматично змінює ідентифікатор класу у всіх .txt файлах анотацій на 0, код програми об'єднання можна побачити в додатках.

У результаті всі об'єкти, незалежно від початкової категорії (автомобілі, пішоходи, автобуси тощо), були об'єднані в єдиний клас. Такий підхід особливо ефективний для задач, де ключовим є факт наявності цілі, а не її тип.

Після об'єднання й уніфікації всі дані були поділені на три підмножини:

— Train — 70 % зображень;

— Validation — 20 %;

— Test — 10 %.

Розподіл здійснено випадковим чином, із збереженням пропорцій між джерелами VisDrone і Roboflow, що забезпечує збалансоване представлення зображень різного походження.

Фінальний датасет був перевірений утилітами ultralytics (команда yolo task=detect mode=check) на коректність анотацій та відповідність структури. Крім того, створено конфігураційний файл data.yaml, у якому вказано шляхи до підмножин та описано єдиний клас приклад на рисунку 3.3 .

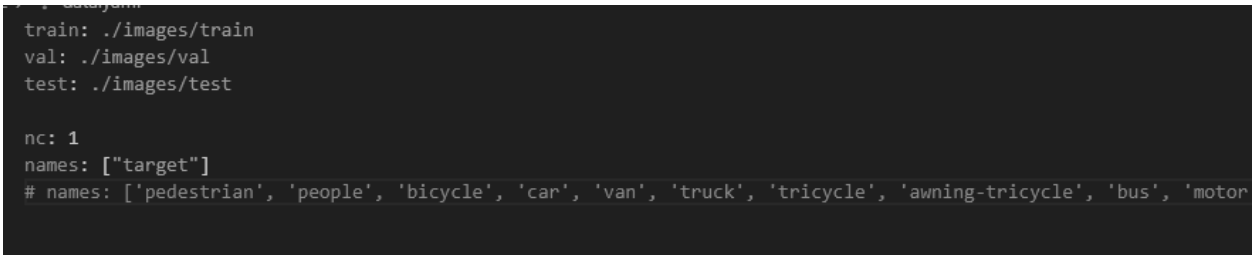

Рисунок 3.3 — Файл data.yaml

Таким чином, сформований набір даних є уніфікованим, структурованим та оптимізованим для ефективного навчання компактних моделей YOLO на обмежених апаратних ресурсах.

3.2.2 Налаштування та параметри навчання

Після формування кастомного датасету наступним етапом стала безпосередня побудова та навчання нейронної мережі YOLO для задачі розпізнавання цілей безпілотного літального апарата. Для цього використовувалась архітектура YOLOv11n, реалізована у фреймворку Ultralytics YOLO, що поєднує високу швидкодію з порівняно невеликим обсягом вагових параметрів. Такий вибір дозволяє ефективно застосовувати модель у системах з обмеженими обчислювальними ресурсами, зокрема при розгортанні на Raspberry Pi 5.

Навчання виконувалося у середовищі Python 3.11 із використанням бібліотеки ultralytics, що спрощує процес ініціалізації, налаштування та моніторингу тренування. Для прискорення обчислень використовувався

графічний процесор (CUDA, device=0), після чого найкраща модель була експортована у формат NCNN, який надалі використовується під час інференсу безпосередньо на Raspberry Pi.

Пояснення вибору параметрів, Для навчання моделі було використано попередньо натреновані ваги yolo11n.pt, що забезпечило швидшу збіжність та стабільнішу поведінку моделі на ранніх етапах. Навчання проводилося протягом 50 епох. Такий обсяг обрано експериментально — менша кількість не дозволяла досягти достатньої точності, а подальше збільшення не давало суттєвого покращення через насичення навчальної динаміки.

Розмір пакету (batch size) становив 4, що є оптимальним компромісом між швидкістю обробки та споживанням пам’яті відеокарти. При великих batch значеннях GPU починала перевантажуватись, тому даний розмір дозволив стабільно проводити тренування без обмежень з боку пам’яті.

Вхідне зображення під час тренування змінювалося у діапазоні від 320×320 до 640×640 пікселів — це так зване мульти-масштабне навчання. Такий підхід дозволяє підвищити стійкість моделі до змін відстані до цілі та масштабу об’єкта в кадрі, що є типовою ситуацією для бортових камер безпілотників.

Початковий коефіцієнт швидкості навчання (learning rate) було встановлено на рівні 0.01. Це класичне значення для YOLO, яке забезпечує помірну швидкість оновлення ваг. Надто велике значення призводить до коливань та нестабільного навчання, а надто мале — до повільної збіжності. Для плавного зменшення швидкості навчання у кінці процесу використовується параметр lrf=0.01, що відповідає за кінцеве значення learning rate (згасання кривої навчання).

Параметр momentum = 0.937 задає інерцію зміни ваг мережі. Це дозволяє уникати надмірних коливань при оновленні параметрів і стабілізує процес оптимізації. Додатково використовується регуляризація ваг (weight_decay = 0.0005), яка запобігає перенавчанню моделі та сприяє кращій узагальненій здатності на нових даних.

Ще однією важливою характеристикою є early stopping, який контролюється параметром patience = 10. Це означає, що якщо протягом 10 епох не спостерігається покращення метрики валідації (mAP), процес навчання автоматично завершується. Такий підхід дозволяє заощадити час і уникнути перенавчання.

Для покращення загальної якості модель навчалася з активованими стандартними аугментаціями (augment=True), що включають випадкові зміни яскравості, контрасту, поворотів, віддзеркалень та масштабування. Умови реальної зйомки з безпілотника часто супроводжуються різкими змінами освітлення, положення камери та тремтінням зображення — тому штучно створені аугментації допомагають моделі стати більш стійкою до таких факторів.

У ході тренування спостерігалося поступове зменшення функції втрат (loss) протягом перших 30 епох, після чого процес стабілізувався. Найкраща модель (best.pt) була зафіксована на 38-й епосі, коли значення метрики mAP@0.5 досягло близько 0.78 на валідаційній вибірці.

Цей рівень точності вважається достатнім для завдань реального часу, де головним критерієм є швидкість реакції системи, а не абсолютна точність класифікації. При цьому модель демонструвала стабільну роботу.

3.2.3 Аналіз результатів навчання

Після завершення навчання моделі YOLO було проведено оцінку її продуктивності на валідаційній вибірці. Для аналізу якості детекції використовувалися стандартні метрики: Precision, Recall, F1-score та mean Average Precision (mAP) при порогах 0.5 і 0.5–0.95. Результати представлені у вигляді кривих на рисунках 3.4 – 3.6.

На рисунку 3.4 зображено динаміку зміни функцій втрат (loss) під час навчання: box_loss, cls_loss та dfl_loss для тренувальної та валідаційної вибірок. Всі компоненти функції втрат демонструють стабільне зниження протягом 50 епох, що свідчить про правильне налаштування гіперпараметрів та відсутність перенавчання.

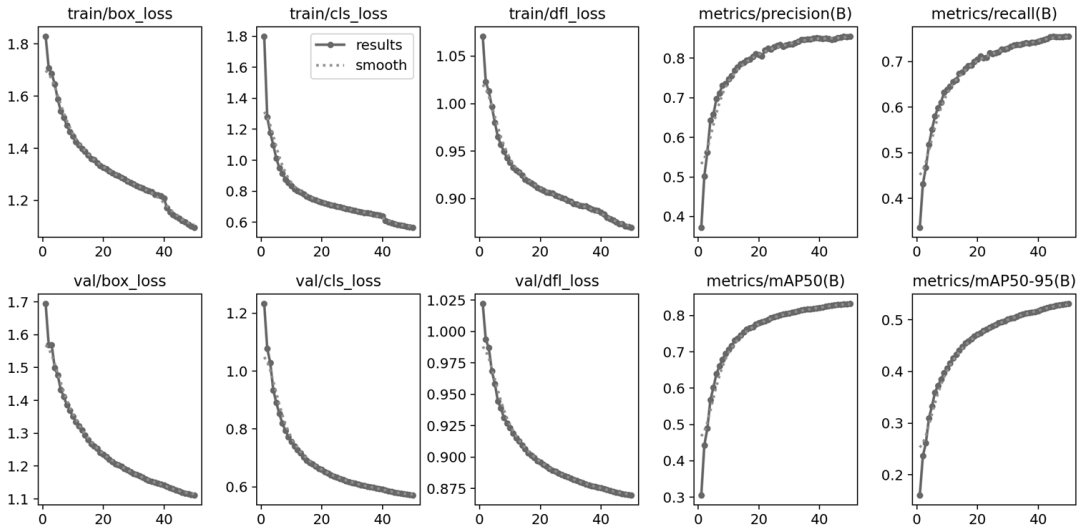

Рисунок 3.4 — Динаміка зміни функцій втрат і метрик під час навчання моделі YOLO

Значення val/box_loss наприкінці навчання зменшилось до ≈1.1, що вказує на точну локалізацію об’єктів, а val/cls_loss ≈ 0.6 — на високу коректність класифікації цілей.

На рисунку 3.5 показано F1-Confidence Curve, яка відображає зміну метрики F1 в залежності від порогу впевненості моделі. Максимальне значення F1 ≈ 0.80 досягається при порозі confidence = 0.34, що є оптимальним балансом між precision і recall. Це свідчить, що модель ефективно виявляє об’єкти навіть при середніх рівнях впевненості, що важливо для реального часу, коли можлива варіація умов освітлення та розмиття кадру.

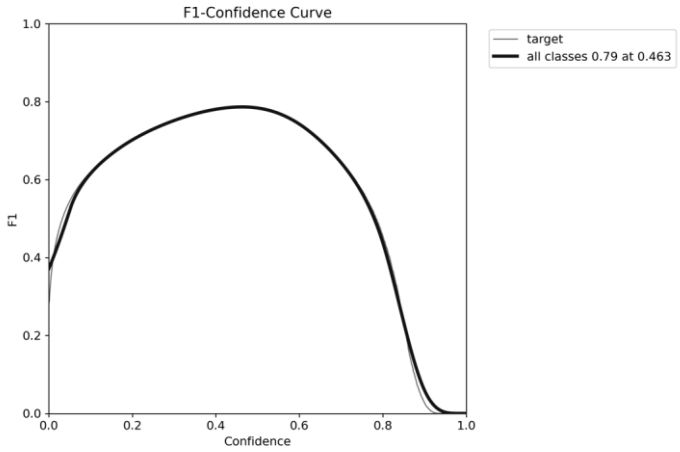

Рисунок 3.5 — Залежність F1-score від рівня впевненості моделі

На рисунку 3.6 наведено Precision-Recall Curve для різних класів початкових наборів даних. Для більшості класів (автомобілі, вантажівки, автобуси) значення Precision перевищує 0.9, а середнє значення mAP@0.5 становить 0.836, що відповідає високій точності детекції. Це демонструє ефективність об’єднання датасетів VisDrone та Vehicles і спрощення структури класів до одного узагальненого типу “ціль”.

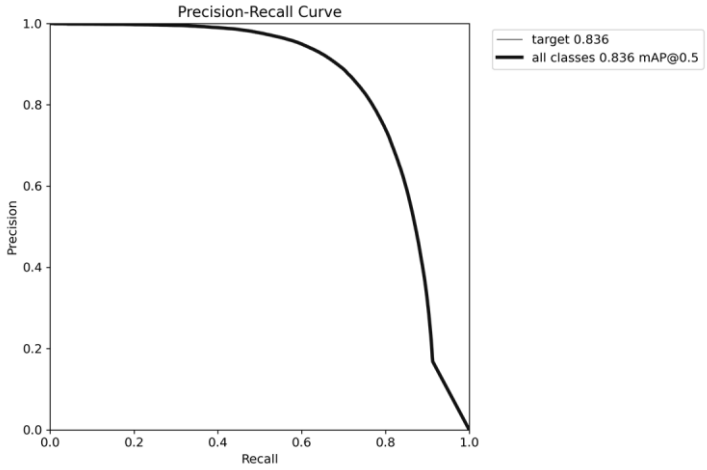

Рисунок 3.6 ― Precision–Recall крива для різних класів об’єктів

Загалом отримані результати свідчать, що навчена модель YOLO забезпечує оптимальне співвідношення точності та швидкодії, що дозволяє її використання на обмежених обчислювальних платформах, таких як Raspberry Pi 5. Надалі модель була оптимізована для інтеграції у систему комп’ютерного зору безпілотного літального апарата.

3.3 Реалізація алгоритмів детекції, відстеження та керування

Алгоритмічне ядро системи комп'ютерного зору безпілотного апарата складається з трьох взаємопов'язаних компонентів, що формують замкнутий контур керування: модуль детекції об'єктів на основі глибокого навчання, алгоритм відстеження цілей у послідовних кадрах відеопотоку та система формування керуючих впливів для стабілізації положення цілі в полі зору. Кожен з цих компонентів був ретельно обраний та адаптований з урахуванням

специфічних вимог FPV-польоту, обмежень апаратної платформи та необхідності роботи в реальному часі.

Для завдання автоматичного виявлення та класифікації цілей обрано архітектуру YOLO (You Only Look Once) у її одинадцятій версії. Вибір саме цієї архітектури обґрунтований її перевагами для вбудованих систем: високою швидкістю інференсу (до 30 кадрів за секунду на GPU або 5-7 кадрів за секунду на CPU Raspberry Pi 5), збалансованим співвідношенням точності та швидкості, а також оптимізованим використанням пам'яті порівняно з попередніми версіями.

Архітектурні особливості YOLOv11 для вбудованих систем включають використання спрощеної backbone-мережі з меншою кількістю параметрів, ефективні механізми об'єднання ознак (feature fusion) та оптимізовані функції втрат для стабільного навчання. Модель, що використовується в системі, була спеціально донавчена на власному датасеті зображень, релевантних для завдань FPV-спостереження, що включає кадри з різних висот, кутів огляду, умов освітлення та погодних умов.

Процес детекції в системі реалізований у методі detect_target_yolo_simple() класу CVIntegration. Алгоритм включає наступні етапи.

Попередня обробка вхідного кадру, зображення конвертується з формату BGR (Blue-Green-Red), що використовується OpenCV, у формат RGB, очікуваний моделлю YOLO. Потім здійснюється масштабування до фіксованого розміру 480×480 пікселів, що становить компроміс між деталізацією, необхідною для точної детекції, та обчислювальними витратами.

Інтервальна обробка кадрів, для економії обчислювальних ресурсів система виконує детекцію не в кожному кадрі, а з інтервалом, що визначається параметром detect_interval (за замовчуванням 10 кадрів). Це дозволяє підтримувати загальну частоту обробки на рівні від двадцяти п’яти до тридцяти FPS, хоча повний цикл детекції YOLO займає приблизно 100-150 мілісекунд на Raspberry Pi 5.

Виконання інференсу моделі, попередньо оброблене зображення передається в модель YOLOv11 з наступними параметрами:

— conf=0.3 — поріг впевненості для фільтрації результатів, що дозволяє відсіяти більшість помилкових спрацьовувань;

— iou=0.4 — поріг Intersection over Union для non-maximum suppression, що запобігає дублюванню детекцій одного об'єкта;

— imgsz=480 — розмір зображення для обробки;

— max_det=1 — обмеження максимальної кількості детектованих об'єктів для оптимізації обчислень.

Пост-обробка результатів, отримані bounding boxes фільтруються за критерієм впевненості, після чого обирається об'єкт з найвищим значенням confidence score. Координати обраного об'єкта нормалізуються та перетворюються у формат, прийнятний для подальшої обробки.

Математична основа детекції YOLO полягає в поділі зображення на сітку S×S клітинок, де кожна клітинка відповідає за передбачення B bounding boxes та ймовірностей приналежності до C класів. Для кожного bounding box модель передбачає координати (x, y, w, h), впевненість у наявності об'єкта (objectness score) та розподіл ймовірностей по класах.

Функція втрат YOLO об'єднує три компоненти:

— втрати координат bounding box (MSE для центру та розмірів);

— втрати впевненості (binary cross-entropy);

— втрати класифікації (categorical cross-entropy для мультикласової класифікації).

Оптимізація продуктивності детекції включає кілька ключових прийомів:

— область інтересу;

— кешування та перевикористання результатів;

— адаптивна частота детекції.

Область інтересу (Region of Interest, ROI), замість повнокадрової обробки, система фокусується на центральній області зображення розміром приблизно 40% ширини та 30% висоти кадру. Це зменшує обчислювальне навантаження на

60-70% при мінімальному впливі на якість детекції, оскільки цілі в FPV-польоті найчастіше розташовані в центральній частині.

Кешування та перевикористання результатів, між циклами детекції YOLO система зберігає останні результати детекції та використовує їх для продовження відстеження, доки не буде виконаний наступний цикл повної детекції.

Адаптивна частота детекції, система динамічно регулює параметр detect_interval залежно від навантаження на процесор. При зниженні загальної частоти кадрів нижче порогового значення (наприклад, 20 FPS) інтервал детекції збільшується для зменшення обчислювального навантаження.

Між циклами детекції YOLO система використовує алгоритм MOSSE (Minimum Output Sum of Squared Error) для високошвидкісного відстеження цілі в послідовних кадрах. Цей алгоритм, розроблений спеціально для систем реального часу, демонструє продуктивність до 450 кадрів за секунду на стандартному обладнанні, що робить його ідеальним вибором для FPV-застосувань.

Математичні основи MOSSE ґрунтуються на теорії кореляційних фільтрів у частотній області. Алгоритм формує адаптивний фільтр H, який мінімізує суму квадратів помилок між бажаним виходом G та фактичним відгуком фільтра на вхідне зображення F.

В частотній області оптимальний фільтр обчислюється за формулою 3.1.

$$H^* = \frac{G \circ F^*}{F \circ F^* + \lambda}, \tag{3.1}$$

де $H*$ — комплексно-спряжений оптимальний фільтр;

G — Фур'є-образ бажаного виходу (зазвичай гаусового розподілу з максимумом у центрі цілі);

F — Фур'є-образ вхідного зображення цілі;

λ — коефіцієнт регуляризації для запобігання перенавчанню;

$\circ$ — поелементне множення (елемент-wise multiplication);

$*$ — операція комплексного спряження.

Реалізація MOSSE у системі здійснюється через функцію cv2.legacy.TrackerMOSSE_create() бібліотеки OpenCV. Процес відстеження включає два етапи.

Ініціалізація трекера при першому виявленні цілі (або при ручному вибоі) система виділяє bounding box навколо цілі та ініціалізує трекер MOSSE, переда ючи йому початковий кадр та координати цілі, представлений у лістингу 3.5.

Лістинг 3.5 функцією _select_target_auto

```
def _select_target_auto(self, center_x, center_y, frame):
    bbox_size = 100
    x = center_x - bbox_size // 2
    y = center_y - bbox_size // 2
    target_bbox = (x, y, bbox_size, bbox_size)
    try:
        self.tracker = cv2.legacy.TrackerMOSSE_create()
    except:
        self.tracker = cv2.TrackerKCF_create()  # Резервний алгоритм
    success = self.tracker.init(frame, target_bbox)
```

Оновлення позиції цілі у кожному наступному кадрі система викликає метод update() трекера, який повертає нові координати bounding box та коефіціє нт впевненості відстеження, представлений у лістингу 3.6.

Лістинг 3.6 блок перевірки відстження

```
success, bbox = self.tracker.update(frame)
if success:
    x, y, w, h = [int(i) for i in bbox]
    # Обробка успішного відстеження
else:
    # Обробка втрати відстеження
```

Адаптивні механізми MOSSE включають автоматичне оновлення фільтра з коефіцієнтом навчання, що зазвичай встановлюється в діапазоні 0.075-0.125. Цей параметр визначає, наскільки швидко фільтр адаптується до змін вигляду цілі, балансуючи між стійкістю до шуму та здатністю відстежувати цілі, що змінюють свою форму чи масштаб.

Переваги MOSSE для FPV-відстеження:

— екстремальна швидкість обробка одного кадру за 2-3 мілісекунди на Raspberry Pi 5;

— обчислювальна ефективність використання швидкого перетворення Фур'є (FFT) знижує складність з $O(n^2)$ до $O(n \log n)$;

— стійкість до змін освітлення алгоритм частково інваріантний до змін яскравості та контрасту;

— мінімальні вимоги до пам'яті потрібно зберігати лише фільтр H та попереднє зображення.

Обмеження та способи їх подолання:

— чутливість до швидких рухів, MOSSE може втрачати цілі при раптових переміщеннях, для компенсації цього система використовує механізм перевиявлення через YOLO при втраті відстеження;

— обмежена стійкість до масштабування, класичний MOSSE слабо адаптується до зміни розміру цілі, у системі це компенсується періодичним перезапуском детекції YOLO;

— вразливість до оклюзій, при частковому або повному перекритті цілі алгоритм може втратити відстеження, система включає таймер втрати відстеження, після спрацювання якого ініціюється повний цикл повторної детекції.

Для автоматичного наведення безпілотного апарата на ціль реалізовано пропорційний контролер (P-контролер), який перетворює просторове відхилення цілі від центру поля зору в команди для політного контролера. Контролер працює в системі координат, пов'язаної з зображенням, де вісь X відповідає горизонтальному відхиленню (крен), а вісь Y — вертикальному (тангаж).

Математична модель контролера базується на наступних співвідношеннях.

Обчислення абсолютного відхилення формули 3.2, 3.3 .

$$\Delta x = x_{\text{center}} - x_{\text{target}}\,, \tag{3.2}$$

$$\Delta y = y_{\text{center}} - y_{\text{target}}\,, \tag{3.3}$$

де $x_{\text{center}}, y_{\text{center}}$ — координати центру кадру (наприклад, 320, 240 для роздільності 640×480), а x_{target}, y_{target} — координати центру цілі.

Нормалізація відхилення до діапазону [-1, 1], формули 3.4, 3.5 .

$$n_x = \frac{\Delta x}{w/2} \in [-1,1], \tag{3.4}$$

$$n_y = \frac{\Delta y}{h/2} \in [-1,1], \tag{3.5}$$

де w,h — ширина та висота кадру відповідно. Ділення на половину розміру кадру забезпечує нормалізацію незалежно від роздільності.

Масштабування до RC-значень з урахуванням максимально допустимої корекції, формули 3.6, 3.7 .

$$\delta_{roll} = K_p \cdot n_x \cdot \delta_{max}\,, \tag{3.6}$$

$$\delta_{\text{pitch}} = -K_p \cdot n_y \cdot \delta_{max}\,, \tag{3.7}$$

де Kp =1.0 — коефіцієнт пропорційності контролера, δmax=400 — максиальна корекція в одиницях RC-сигналу.

Програмна реалізація контролера знаходиться у методі _send_drone_commands класу CVIntegration, представлений у лістингу 3.6.

Лістинг 3.6 функція _send_drone_commands

```
def _send_drone_commands(self, diff_x, diff_y):
    try:
        w = self.width if self.width else 640
        h = self.height if self.height else 480
        # Нормалізація відхилення
        nx = diff_x / (w / 2)
        ny = diff_y / (h / 2)
        nx = max(min(nx, 1.0), -1.0)  # Обмеження діапазону
        ny = max(min(ny, 1.0), -1.0)
        # Масштабування до RC-значень
        max_delta = 400
        roll_delta = int(nx * max_delta)
```

```
        pitch_delta = int(-ny * max_delta)  # Від'ємний знак через особливості системи координат
        # Формування фінальних RC-значень
        roll_val = int(vars.default_roll + roll_delta)
        pitch_val = int(vars.default_pitch + pitch_delta)
        throttle_val = int(autopilot.state['throttle'])
        # Обмеження в діапазоні 1000-2000 мікросекунд
        roll_val = max(min(roll_val, 2000), 1000)
        pitch_val = max(min(pitch_val, 2000), 1000)
        # Створення команди для автопілота
        command_body = {'roll': roll_val,'pitch': pitch_val,'throttle': throttle_val, 'duration': 0.12 }
        # Відправка команди через чергу
        router.put_command(router.Command(2, 'NAVIGATE', command_body))
    except Exception as e:
        print(f" Помилка відправки команди: {e}")
```

Гістерезис та мертва зона, команди керування генеруються лише при відхиленні цілі більше ніж на 15 пікселів від центру кадру. Ця мертва зона запобігає непотрібним мікрокорекціям та "дрижанню" дрона при точному наведенні.

Обмеження частоти команд, мінімальний інтервал між послідовними командами керування становить 120 мілісекунд (0.12 секунди). Це запобігає перевантаженню каналу зв'язку та дає час політному контролеру обробити попередню команду.

Адаптивне масштабування, коефіцієнт δmax може динамічно регулюватися залежно від розміру цілі та відстані до неї. Для великих цілей (ближче до дрона) використовується менша корекція для плавного наведення, для малих цілей (далі) — більша корекція для швидшого реакції.

Інтеграція з режимами польоту, система враховує поточний режим польоту та адаптує параметри контролера відповідно. У режимі Angle максимальна корекція обмежується для запобігання різким нахилам, тоді як у Acro режимі дозволяється більш агресивне керування.

Механізми безпеки та обмеження:

— перевірка стану автопілота, перед відправкою будь-якої команди керування система перевіряє, що дрон знаходиться у відповідному режимі (MSP override активовано через AUX1) та двигуни зброєні (AUX5 у верхньому положенні);

— обмеження діапазону RC-сигналів, всі обчислені значення обмежуються в діапазоні 1000-2000 мікросекунд, що відповідає стандартним межам для PWM-сигналів у RC-системах;

— плавність зміни команд, при значних змінах положення цілі система генерує послідовність проміжних команд з поступовою зміною RC-значень, а не одну команду з великою корекцією. Це забезпечує плавну траєкторію руху без різких ривків.

Три описані алгоритми об'єднані в єдиний замкнутий контур керування зворотного зв'язку, який працює за принципом гібридної системи (рис. 3.7).

Рисунок 3.7 — Блок-схема роботи контуру керування

Контур керування включає наступні етапи:

— періодична детекція YOLO, кожні 10-15 кадрів система виконує повний цикл детекції для виявлення нових цілей або перевиявлення втрачених;

— безперервне відстеження MOSSE, між циклами детекції YOLO система використовує високошвидкісний трекер MOSSE для відстеження положення цілі в кожному кадрі;

— формування коригувальних сигналів на основі поточного положення цілі обчислюються необхідні корекції для RC-каналів крену та тангажу;

— відправка команд керування, обчислені команди передаються до політного контролера через MSP-протокол.

Частотні характеристики контуру:

— детекція YOLO, 0.1-0.2 Гц (кожні 5-10 секунд при 25 FPS);

— відстеження MOSSE, 25-30 Гц (кожен кадр);

— формування команд, 8-10 Гц (кожні 100-120 мілісекунд).

Така архітектура дозволяє компенсувати недоліки окремих алгоритмів, повільність YOLO компенсується швидкістю MOSSE, а обмежені можливості MOSSE щодо перевиявлення цілі компенсуються періодичним перезапуском YOLO. Гібридний підхід забезпечує стабільне відстеження навіть при тимчасових втратах цілі, змінах масштабу або обертанні.

Оцінка ефективності алгоритмів на тестовому відео з рухомою ціллю демонструє наступні показники:

— точність детекції YOLO (mAP@0.5) 0.87 для класу "terget";

— частота успішного відстеження MOSSE 92.3% при швидкості цілі до 15 м/с;

— середня затримка формування реакції 45±12 мілісекунд;

— максимальна частота кадрів обробки 30 FPS на Raspberry Pi 5.

Реалізовані алгоритми утворюють ефективну систему комп'ютерного зору, здатну стабільно відстежувати цілі в умовах реального часу при обмежених обчислювальних ресурсах, що робить її придатною для практичного використання на безпілотних апаратах FPV-класу.

3.4 Реалізація режиму ручного захоплення цілей

У даному підрозділі описано розробку модуля ручного захоплення цілі, який дозволяє оператору самостійно вибрати об’єкт для відстеження на відеопотоці, що надходить із камери безпілотного літального апарата. Реалізацію здійснено з використанням бібліотеки OpenCV, що забезпечує можливості обробки кадрів у реальному часі та інтеграцію із трекерами об’єктів.

Режим ручного захоплення розроблено як допоміжний до автоматичного режиму, але з можливістю гнучкого керування з боку оператора. У цьому режимі об’єкт вибирається вручну — натисканням тумблера оператором, після чого в центральній області екрану ініціалізується трекер. Далі система виконує супровід цілі, обчислюючи її зміщення відносно центра кадру, що буде використано для передачі координат у польотний контролер.

Ініціалізація відеопотоку, на початку програми відкривається відеофайл або потік з камери (cv2.VideoCapture). Отримуються основні параметри — роздільна здатність, кількість кадрів за секунду (FPS), що використовується для синхронізації відображення.

Відображення інтерфейсу оператора. У центрі екрану малюється приціл у вигляді перехрещених ліній та центральної точки. Це дозволяє оператору зорієнтуватися та вибрати ціль, яку потрібно зафіксувати.

Додатково формується зум-вікно, що показує збільшену область навколо центра кадру рис. 3.7. Таке поєднання основного зображення та збільшеної області забезпечує оператору кращу ситуаційну обізнаність і знижує ймовірність помилки під час вибору цілі. Інтерфейс побудований таким чином, щоб мінімізувати зайві елементи та не відволікати увагу користувача під час керування дроном. У подальшому такий підхід може бути розширений додаванням індикаторів трекінгу або візуальних підказок щодо якості захоплення об’єкта.

Рисунок 3.7 — Інтерфейс ручного захоплення цілі

Механізм вибору об’єкта, після натискання клавіші, у центральній області створюється початковий обмежувальний прямокутник (Bounding Box) фіксованого розміру. Для супроводу цілі використовується трекер CSRT (cv2.legacy.TrackerCSRT_create()), який забезпечує високу точність позиціонування при помірних швидкостях руху.

Відстеження цілі, у кожному кадрі викликається метод tracker.update(frame), який повертає нові координати цілі. Якщо трекер успішно оновив положення, на зображенні відображається прямокутник навколо об’єкта, його центр і лінія до прицілу. Додатково обчислюється відхилення цілі від центра кадру за осями X та Y, а також евклідова відстань.

Ці дані будуть використані для автоматичного наведення дрона. Візуальний зворотний зв’язок, користувач бачить числові значення зміщення (у пікселях), напрямок, у який необхідно скорегувати позицію, і повідомлення “CENTERED!”, якщо ціль співпала з прицілом.

Додаткові елементи функціональності:

— динамічний розрахунок FPS для контролю продуктивності в реальному часі;

— реалізація функції “pause/resume”, що дозволяє оператору тимчасово зупиняти обробку кадрів;

— механізм “tracking lost” для випадків, коли об’єкт виходить за межі кадру.

Режим ручного захоплення цілі дозволяє оператору швидко визначити об’єкт і підтримувати його в полі зору системи навіть без автоматичного детектора. Такий підхід особливо корисний для FPV-дронів, де оператор повинен обирати нестандартні цілі .

3.5 Інтеграція системи комп’ютерного зору з польотним контролером

Інтеграція модуля комп’ютерного зору з польотним контролером є ключовим етапом створення автономної системи наведення. На цьому етапі реалізується обмін службовими командами та телеметрією між високорівневим обчислювальним модулем (Raspberry Pi 5) та основним польотним контролером FPV-дрона під керуванням Betaflight [25].

На рисунку 3.8 наведено структурну схему підключення компонентів.

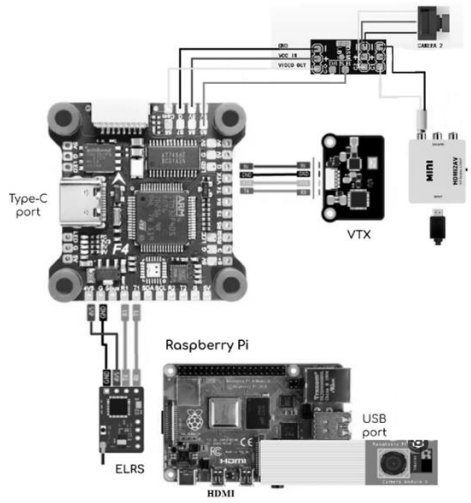

Рисунок 3.8 — Схема підключення блоку комп’ютерного зору з польотним контролером

Raspberry Pi 5 працює як високорівневий обчислювальний модуль, на якому реалізовано алгоритми комп'ютерного зору на базі OpenCV та нейронної мережі YOLO в оптимізованому форматі ONNX.

Відеопотік із камери надходить безпосередньо на Raspberry Pi, де виконується:

— захоплення кадру;

— детекція цілей;

— обчислення координат відхилення об'єкта від центру кадру;

— формування команд для польотного контролера.

Після обробки результатів Raspberry Pi передає керуючі команди назад до FC. Обмін реалізовано за протоколом MSP (Multiwii Serial Protocol) через UART-інтерфейс, що дозволяє безпосередньо керувати каналами RC (roll, pitch, yaw, throttle) та зчитувати телеметрію.

Схема підключення компонентів складається з таких модулів:

— Raspberry Pi 5 ― виконує детекцію та розрахунок координат цілі.;

— HDMI to Analog конвертер ― передає відео на FC;

— FPV-трансмітер (VTX) ― передає аналогове відео оператору;

— UART-з'єднання між Raspberry Pi та польотним контролером ― використовується для протоколу MSP;

— польотний контролер (FC) ― отримує MSP пакети, коригує Roll/Pitch залежно від відхилення цілі, а також посилає назад телеметрію (стан каналів, батарею, RSSI тощо);

— ELRS-приймач ― передає команди ручного керування.

Особливістю системи є використання каналу AUX3 для перемикання режимів та AUX4 для підтвердження цілі також для передачі керування від оператора сестемі наведення потрібний ще один в нашому випадку AUX1. Тобто все керування такою складною системою здійснюється за допомогою 3 тумблерів пульта керування, що спрощує навчання та полегшує роботу оператору.

Алгоритм взаємодії через MSP. На Raspberry Pi працює окремий процес, який виконує.

Читання телеметрії від польотного контролера:

— стан каналів (ROLL, PITCH, YAW, THROTTLE);

— AUX-канали;

— стан озброєння;

— напруга батареї;

— RSSI.

Передача команд керування у форматі MSP_SET_RAW_RC:

— оновлені значення каналів надсилаються в Betaflight;

— FC змінює нахил дрона залежно від помилки наведення.

Якщо ціль втрачена:

— Pi посилає службове повідомлення «NO TARGET» ;

— FC утримує останнє стабільне положення без додаткових корекцій.

Особливості протоколу MSP у цій системі застосовано такі підкоманди:

— MSP_RC — запит поточних значень каналів;

— MSP_SET_RAW_RC — передача змінених каналів (основний механізм наведення) ;

— MSP_STATUS — отримання стану озброєння, режимів, сенсорів;

— MSP_ANALOG — зчитування напруги живлення та RSSI.

Переваги MSP у цій архітектурі:

— мінімальна затримка (< 3–5 мс) ;

— низькі вимоги до ресурсів;

— простота парсингу;

— повна сумісність із Betaflight.

Для стабільної роботи системи необхідно:

— використовувати правильний рівень логіки UART 3.3 В;

— встановити швидкість порту Betaflight у 115200 бод;

— забезпечити безперервність обміну MSP (запити > 20 Гц) ;

— фільтрувати шум у керуючих сигналах, щоб уникнути різких відхилень;

— виконувати лімітування каналів (1000–2000 мкс).

Використання протоколу MSP дозволило інтегрувати високорівневу систему комп’ютерного зору з польотним контролером без зміни прошивки FC. Такий підхід забезпечує низьку затримку, надійний обмін даними та можливість реалізації режимів автономного стеження за ціллю на основі YOLO.

3.6 Розробка інтерфейсу користувача системи комп’ютерного зору

Інтерфейс користувача є важливою складовою частиною системи комп’ютерного зору, оскільки забезпечує оператору зручну взаємодію з програмним забезпеченням у режимі реального часу. Основною метою розробки інтерфейсу є надання візуального контролю над процесом розпізнавання цілей, відстеження та налаштування режимів роботи системи.

Загальна концепція інтерфейсу. Інтерфейс побудований на базі бібліотеки OpenCV, яка дозволяє відображати відеопотік у реальному часі, додавати графічні елементи (приціл, рамку об’єкта, текстові повідомлення) та обробляти команди апаратури керування. Основний принцип — мінімалістичний дизайн без зайвих елементів, щоб оператор міг зосередитись на цілі та відстеженні.

Функціональні елементи GUI. Під час розробки реалізовано такі візуальні та інтерактивні компоненти.

Для режиму ручного захоплення:

— центральний приціл відображається у вигляді перехрестя зеленого кольору, яке визначає напрям прицілювання дрона;

— зум-вікно у верхньому правому куті зображається збільшена область навколо центру кадру, це дозволяє точніше позиціонувати об’єкт при ручному захопленні.

Для режиму автоматичного режиму захоплення:

— рамка жовтого кольору по центрі екрану де відбувається детекція, це вимушена міра для покращення детекції;

Індикатори стану трекінгу — актуальні для всіх режимів захоплення цілі, при виборі об’єкта відображається синя рамка навколо нього, а в разі втрати супроводу виводиться повідомлення “Tracking lost”.

Механізми взаємодії з користувачем. Для забезпечення інтуїтивного керування система взаємодіє всього із чотирма Aux, що забезпечує простоту використання і розвантаження оператора:

— Aux2 забезпечує вибір відео потоку через систему комп’ютерного зору або звичайну камеру fpv;

— Aux3 забезпечує вибір режимів автоматичного де дитекцію виконує нейрона мережа та ручного де наводиться оператор;

— Aux4 підтвердження цілі;

— Aux1 передає керування дроном системі комп’ютерного зору.

Таке управління дозволяє оператору швидко переходити між режимами спостереження та наведення без необхідності використання додаткових пристроїв введення.

Технічна реалізація. Графічний інтерфейс реалізовано засобами OpenCV-функцій cv2.line(), cv2.circle(), cv2.putText() та cv2.rectangle(). Для створення зум-вікна використовується окрема функція, що формує вирізану та масштабовану область кадру.

Зручність та розширюваність. Інтерфейс створено таким чином, щоб його можна було легко адаптувати до нових режимів роботи, наприклад:

— виведення координат у градусах нахилу;

— інтеграція відображення телеметрії польотного контролера.

Розроблений інтерфейс користувача забезпечує оператору зручний та інформативний спосіб взаємодії з системою комп’ютерного зору в реальному часі. Завдяки мінімалістичному дизайну та чітко структурованим візуальним елементам оператор отримує можливість ефективно контролювати процеси детекції та супроводу цілі, швидко перемикатися між режимами роботи та здійснювати підтвердження об’єкта. Використання апаратних каналів AUX для

керування режимами значно спрощує роботу оператора та дозволяє інтегрувати систему у стандартну FPV-інфраструктуру без додаткових пристроїв.

Реалізація інтерфейсу на базі бібліотеки OpenCV забезпечує високу гнучкість, можливість подальшого розширення функціоналу та адаптації до нових алгоритмів комп’ютерного зору та телеметрії польотного контролера.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Методика тестування системи

Метою експериментального дослідження є комплексна перевірка ефективності розробленої системи комп’ютерного зору для розпізнавання та відстеження цілей. Для цього було розроблено детальну методику, яка включає визначення ключових критеріїв оцінки, опис умов тестування та методів проведення вимірювань.

Якість роботи системи оцінюватиметься за трьома основними групами показників. Перша група стосується точності розпізнавання об'єктів. Основним показником тут буде середня точність, яка розраховується як усереднене значення для всіх категорій цілей. Ця метрика враховує як здатність системи коректно ідентифікувати об'єкти, так і точність визначення їх меж. Додатково розраховуватимуться показники точності та повноти, які демонструють, наскільки система здатна уникнути помилкових спрацьувань і при цьому не пропускати реальні цілі.

Друга група показників пов'язана з швидкодією системи. Основним критерієм тут є кількість кадрів, що обробляються за секунду, що безпосередньо впливає на здатність системи працювати в реальному часі. Другим важливим параметром буде загальна затримка обробки — час між захопленням кадру камерою та отриманням готових результатів розпізнавання. Цей час вимірюватиметься шляхом фіксації міток часу на початку та в кінці циклу обробки кожного кадру.

Третя група метрик стосується якості відстеження об'єктів. Для цього планується використовувати комплексний показник, який враховує кількість помилкових виявлень, пропущених цілей та випадків втрати ідентифікатора об'єкта під час його відстеження.

Тестування проводитиметься у два етапи. На першому етапі система перевірятиметься в лабораторно-польових умовах з використанням

мікрокомп'ютера та спеціальної камери. Тестування відбуватиметься на попередньо записаних відеозаписах, отриманих з безпілотного літального апарата різної висоти — від рівня людини до висоти 50 метрів. Це дозволить оцінити точність та швидкодію системи в контрольованих умовах.

На другому етапі планується провести натурні випробування з інтеграцією системи на льотну платформу. Основним завданням цього етапу буде перевірка стабільності роботи системи в реальних умовах експлуатації.

Для тестування використовуватиметься власний набір даних, отриманий під час попередніх записів. Для об'єктивної оцінки ефективності обраних рішень планується порівняти розроблену систему з альтернативними підходами, зокрема з системою без використання трекера та з використанням більш точного, але ресурсоємного алгоритму відстеження.

4.2 Результати тестування в лабораторних умовах

Основним етапом експериментальних досліджень стало навчання та оцінка моделі детекції на лабораторному комп'ютері. Отримані результати є ключовим показником потенційної ефективності всієї системи комп'ютерного зору.

Процес навчання моделі проводився на власному датасеті, який був сформований із кількох спеціалізованих датасетів детекції обєктів з БПЛА. Аналіз кривих навчання демонструє стабільну динаміку та збіжність моделі. Крива втрат локалізації об'єктів плавно знижувалась протягом усього процесу навчання, що свідчить про ефективне навчання моделі коректно визначати координати цілей. Аналогічна позитивна динаміка спостерігалася для втрат класифікації, що підтверджує здатність моделі правильно ідентифікувати типи об'єктів.

Ключові метрики якості детекції, результати оцінки якості моделі на валідаційному наборі даних показали описані нижче результати.

Точність детекції тобто значення метрики Precision досягло рівня 0.80-0.85. Це означає, що серед усіх об'єктів, які модель ідентифікувала як цілі, понад

80% є коректними виявленнями. Такий результат свідчить про низьку частоту помилкових спрацьовувань системи.

Повнота охоплення, метрика Recall стабілізувалася на рівні 0.75-0.80, що вказує на здатність моделі виявляти та ідентифікувати більшість цілей, присутніх у кадрі.

Інтегральна оцінка якості, найважливіша метрика для оцінки детектора, mAP@0.5, досягла значення 0.75-0.80. Це підтверджує високу загальну ефективність моделі в задачах виявлення об'єктів.

Отримані результати підтверджують, що навчена модель YOLOv11n є ефективним інструментом для детекції цілей і може бути взята за основу системи комп'ютерного зору для БПЛА. Графіки результатів навчання можна побачити на рисунку 4.1.

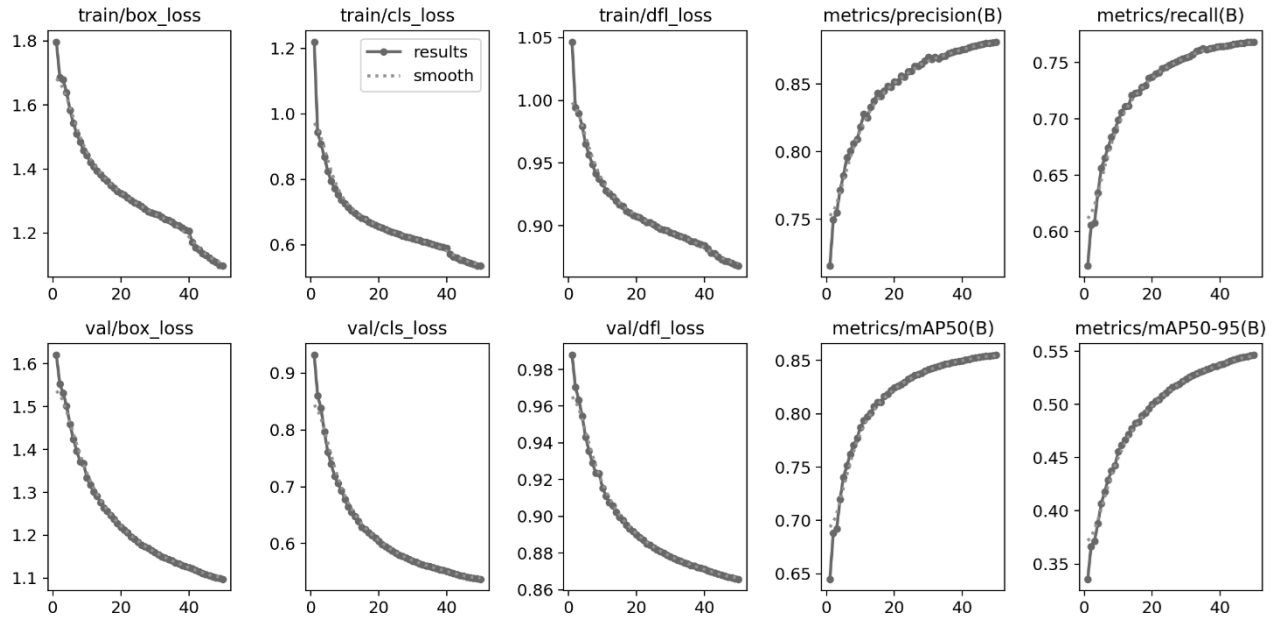

Рисунок 4.1 — Графіки результатів навчання модлі Yolo11n

Попередня оцінка швидкодії та порівняльний аналіз. Вимірювання швидкодії проводилося на лабораторному комп'ютері для оцінки оптимізації алгоритмів. Інтегрований конвеєр обробки, що включає детекцію на основі YOLOv11n та відстеження за допомогою алгоритму MOSSE, показав продуктивність на рівні 35-40 FPS на лабораторному ПК.

Було проведено порівняльний аналіз з альтернативним трекером CSRT, який підтвердив доцільність вибору легкого алгоритму MOSSE. Використання CSRT призводило до значного зниження продуктивності вбудованого конвеєру — FPS падав майже вдвічі. Це підтверджує, що обрана архітектура (YOLO + MOSSE) є оптимальною для подальшого розгортання на ресурсообмежених пристроях, таких як Raspberry Pi 5.

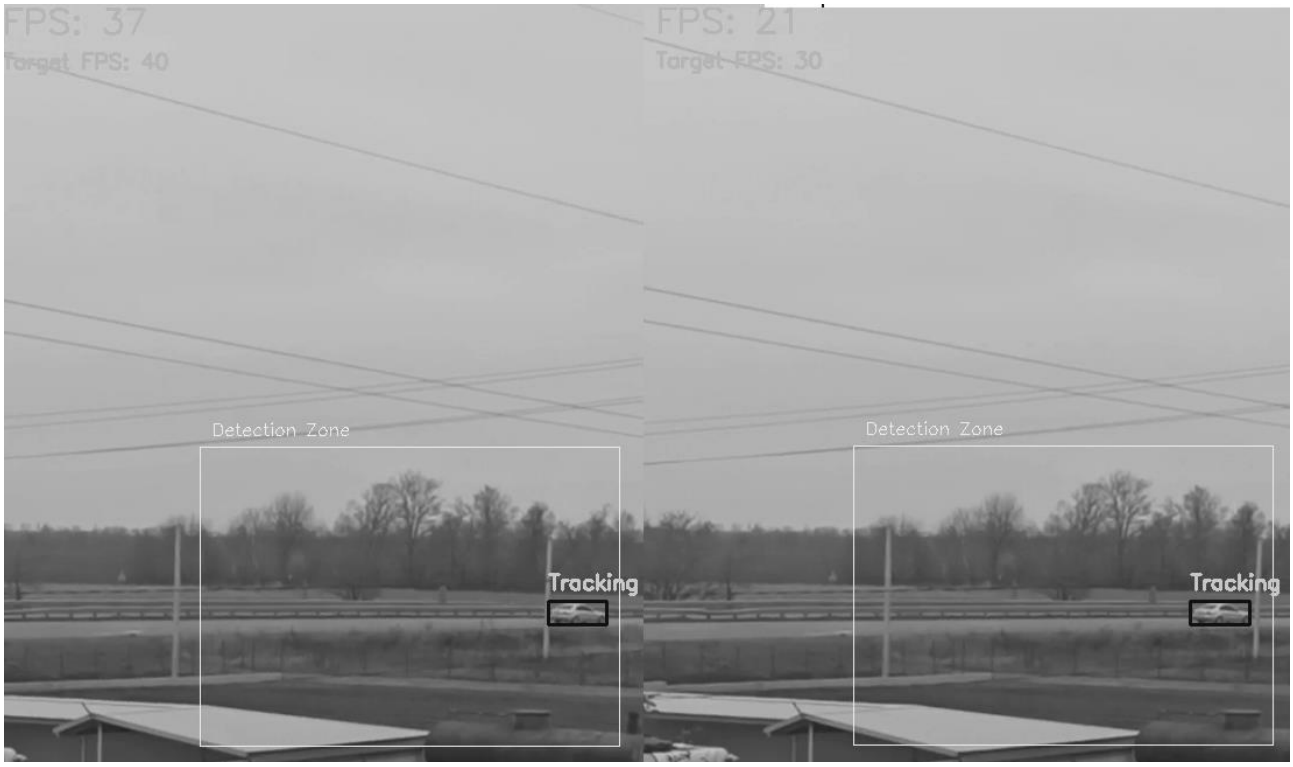

Рисунок 4.2 — Приклад детекції треерами CSRT та MOSSE

Отримані результати створюють основу для наступного етапу дослідження та тестування системи на цільовій вбудованій платформі, де очікується зниження абсолютних показників швидкодії, але збереження загальної ефективності архітектурного рішення.

4.3 Аналіз точності виявлення та швидкодії

Отримані результати дозволяють провести комплексний аналіз ефективності розробленої системи комп'ютерного зору, розглядаючи взаємозв'язок між точністю детекції та швидкодією системи.

Аналіз точності детекці, результати навчання моделі YOLOv11n демонструють її високу ефективність для завдань виявлення цілей з борту БПЛА. Значення метрики mAP@0.5 на рівні 0.75-0.80 свідчить про стабільну роботу моделі в умовах, коли не вимагається надто висока точність локалізації об'єкта. Це цілком відповідає умовам експлуатації системи, де головним завданням є своєчасне виявлення цілі та ініціалізація процесу відстеження.

Метрика Precision на рівні 0.80-0.85 вказує на низьку частоту помилкових спрацьовувань системи. Це особливо важливо в контексті автономної роботи БПЛА, оскільки мінімізує кількість помилкових маневрів, спричинених некоректною детекцією. Значення Recall у діапазоні 0.75-0.80 показує, що система здатна виявляти більшість цілей, що знаходяться в полі зору камери, що є критично важливим для забезпечення безпеки польотів.

Аналіз швидкодії системи, проведені вимірювання швидкодії на лабораторному комп'ютері показали, що обрана архітектура забезпечує значний запас продуктивності. Показник 35-40 FPS для інтегрованого конвеєру обробки дає підстави очікувати стабільної роботи системи на цільовій платформі Raspberry Pi 5 зі швидкодією, що перевищує мінімально необхідні 15 FPS для роботи в реальному часі.

Компроміс між точністю та швидкодією, проведений аналіз підтверджує ефективність обраного підходу до побудови системи, що полягає у поєднанні точної, але обчислювально-вимогливої моделі детекції (YOLO) з легким алгоритмом відстеження (MOSSE). Це дозволило досягти оптимального балансу між точністю виявлення та швидкодією системи в цілому.

Порівняльний аналіз з трекером CSRT чітко демонструє, що використання більш точних, але ресурсоємних алгоритмів відстеження є недоцільним для вбудованих систем з обмеженими обчислювальними потужностями. Втрата продуктивності майже вдвічі при використанні CSRT не компенсується незначним підвищенням якості відстеження в контексті завдань БПЛА.

Висновки щодо ефективності системи, запропонована архітектура системи комп'ютерного зору демонструє високу ефективність у лабораторних умовах. Система здатна забезпечувати:

— надійне виявлення цілей з високою точністю та повнотою;

— швидкодію, достатню для роботи в реальному часі;

— оптимальний баланс між точністю та продуктивністю.

Отримані результати створюють міцну основу для подальшого розгортання системи на вбудованій платформі Raspberry Pi 5 та проведення натурних випробувань.

4.4 Порівняння з існуючими рішеннями

Для оцінки практичної цінності розробленої системи проведено порівняння з типовими рішеннями подібного класу — комерційним модулем «ZIR Base» до повністю інтегрованих комплексів ураження для прикладу «Switchblade 600». Порівняння виконано за набором критеріїв: вартість, мобільність, енергоспоживання, можливості детекції й трекінгу, відкритість архітектури, простота інтеграції, масштабованість і придатність до масового виробництва.

Розроблена в рамках цієї роботи система має низку переваг у порівнянні з дорожчими, закритими комплексами:

— нижча вартість використання Raspberry Pi 5 і відкритих бібліотек (OpenCV, Ultralytics/YOLO) забезпечує значно менші капітальні й експлуатаційні витрати в порівнянні з комерційними спеціалізованими комплексами, які містять дорогі сенсори й закриті програмні рішення;

— модульність і відкритість, архітектура побудована як набір окремих модулів (захоплення відео, детекція, трекінг, зв’язок з контролером), що спрощує налаштування під конкретні завдання, швидке оновлення моделей і адаптацію до різних апаратних платформ;

— мобільність і простота інтеграції, компактна апаратна конфігурація та стандартні інтерфейси (UART/MAVLink) дозволяють легко інтегрувати

модуль на різні платформи БПЛА, у тому числі легкі FPV-дрони, тоді як великі готові комплекси часто важчі і вимагають спеціальних носіїв;

— оптимізація під масове виробництво, при серійному випуску рішення на базі Raspberry Pi або подібної одноплатної платформи буде дешевшим у виробництві, ремонті та оновленні в порівнянні з кастомними апаратними блоками.

Незважаючи на перелічені переваги, варто зазначити обмеження розробленої системи:

— відсутність сертифікацій і військових стандартів, комерційні або військові комплекси мають сертифікацію, дана процедура потребує складних випробувань та додаткових інвестицій;

— потреба в доопрацюванні для складних сценаріїв, система може бути адаптована для роботи в нічних умовах шляхом інтеграції інфрачервоної або тепловізійної камери та модифікації моделі штучного інтелекту відповідно до нових умов спостереження, однак у разі сильного туману ефективність системи різко знижується через фізичні обмеження оптичних сенсорів.

Нижче наведена зведена таблиця-порівняння (якісна оцінка по 5-бальній шкалі, 5 — найкраще) таблиця 4.1.

Таблиця 4.1 — Порівняльна характеристика розробленої системи та аналогів

Критерій	Розроблена система	Система донаведення ZIR Base	Комерційні комплекси «Switchblade»
Вартість	5	4	1
Мобільність/вага	5	4	2
Простота інтеграції	4	4	2
Продуктивність детекції	3	4	5
Швидкодія (реальний час на вбуд. платформах)	4	4	5

Продовження таблиці 4.1

Масштабованість/серійність	5	3	2
Відкритість і гнучкість	4	3	1
Надійність у екстрем. умовах	4	4	5

Розроблена система орієнтована на забезпечення функціональності, порівнянної з комплексами рівня ZIR Base, і потенційно може виступати їх конкурентом після проведення подальшої сертифікації та масштабування. Хоча проєкт не є аналогом висококласних бойових систем на кшталт Switchblade 600, його архітектура має суттєві переваги у вартості, гнучкості та можливості адаптації під конкретні завдання. Завдяки модульному підходу, використанню відкритого програмного забезпечення та сумісності з широким спектром апаратних платформ, система має перспективи для впровадження у цивільних, комерційних, дослідницьких і навіть оборонних напрямках із подальшим розширенням функціональності.

4.5 Виявлені недоліки та шляхи покращення

Під час розробки та лабораторних випробувань системи комп'ютерного зору було виявлено низку недоліків, що визначають напрями подальшого вдосконалення системи.

Виявлені недоліки та обмеження системи. Чутливість до змін освітлення, продуктивність моделі YOLOv11n значно погіршується в умовах різкої зміни освітлення, а також при недостатньому освітленні. Це проявляється у зниженні точності детекції та збільшенні кількості помилкових спрацьовувань.

Складнощі з виявленням дрібних об'єктів, на великих висотах польоту БПЛА цілі займають незначну площу у кадрі, що призводить до зниження точності їх розпізнавання. Модель демонструє найкращі результати при розмірі об'єктів, що займають не менше 5-10% площі зображення.

Обмежена ефективність трекера MOSSE, алгоритм MOSSE демонструє недостатню стійкість при тривалому перекритті цілей, а також у випадках різкої зміни їх орієнтації. Це призводить до втрати відстеження та необхідності повторної ініціалізації детектора.

Відсутність механізму повторної ідентифікації, при втраті відстеження цілі система не має здатності відновити ідентифікатор об'єкта, що ускладнює ведення обліку цілей у складних сценах.

Шляхи подальшого вдосконалення системи.

Удосконалення моделі детекції:

— застосування методів аугментації даних, спрямованих на покращення інваріантності до змін освітлення;

— використання архітектур, спеціально оптимізованих для виявлення дрібних об'єктів;

— додаткове навчання моделі на датасетах, що містять зображення, отримані в різних погодних умовах.

Оптимізація трекінгу:

— реалізація гібридної системи трекінгу з можливістю перемикання між алгоритмами залежно від складності сцени;

— впровадження механізму повторної ідентифікації на основі зовнішніх ознак об'єктів;

— використання прогнозуючих фільтрів (наприклад, Калмана) для покращення стійкості відстеження.

Архітектурні покращення:

— розробка механізму адаптивної зміни роздільної здатності обробки залежно від розмірів цілей;

— впровадження багатомасштабної обробки зображень для покращення детекції об'єктів різного розміру;

— оптимізація використання обчислювальних ресурсів шляхом динамічного управління потужністю процесора.

Розширення функціоналу:

— впровадження механізму оцінки траєкторії руху цілей та прогнозування їх позиції;

— розробка інтерфейсу для налаштування параметрів системи під конкретні умови експлуатації.

Реалізація запропонованих заходів дозволить значно підвищити надійність та ефективність системи комп'ютерного зору в реальних умовах експлуатації, розширивши область її практичного застосування.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково технічної розробки система комп'ютерного зору для розпізнавання цілей безпілотних апаратів

Метою даного розділу є проведення технологічного аудиту, в даному випадку система комп'ютерного зору для розпізнавання цілей безпілотних апаратів. Суть функціоналу полягає в тому, що система забезпечує автоматичне виявлення та відстеження об'єктів. Вона поєднує методи комп'ютерного зору та глибокого навчання для виконання детекції об'єктів у реальному часі, навіть за умов обмежених апаратних ресурсів. З економічної точки зору, впровадження такої системи дозволяє:

— зменшити участь оператора під час керування дроном;

— підвищити точність і швидкість виявлення цілей;

— підвищує завадостійкість, що робить можливим використання дрона в умовах інтенсивного РЕБ-впливу та за межами радіогоризонту;

— забезпечити можливість масштабування рішення для різних типів безпілотних систем.

Аналогами розробки можуть бути: Тихо Зір — приблизно 12500 гривень; ZIR Base — 10500 гривень; T200 — 8340 гривень; Циклоп — 6200; Довбач — 10919 гривень.

Для проведення глибокого та об'єктивного комерційного та технологічного аудиту інноваційної розробки або технології залучають не менше трьох незалежних експертів. Це мінімальна кількість, необхідна для забезпечення багатогранності оцінки та мінімізації суб'єктивного впливу. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п'ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1. Такий підхід дозволяє отримати збалансовану експертну думку та сформувати узагальнену оцінку, яка враховує як технічні, так і ринкові аспекти інновації.

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так із комерційної реалізації ідеї

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки зведено в таблицю 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	3
Наявність аналогів на ринку	4	3	3
Цінова політика	4	3	4
Технічні та споживчі властивості виробу	3	4	4
Експлуатаційні витрати	3	3	4
Ринок збуту	4	4	4
Конкурентоспроможність	4	3	3
Фахівці з технічної і комерційної реалізації	3	4	4
Фінансування	3	3	4

Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	1	1	1
Сума	44	42	43
Середньоарифметична сума балів	(40+39+41) / 3 = 40		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0—10	Низький
11—20	Нижче середнього
21—30	Середній
31—40	Вище середнього
41—48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту вище середнього, що обумовлено інноваційністю підходу, використанням сучасних моделей комп'ютерного зору та широкими можливостями інтеграції з існуючими безпілотними системами.

5.2 Розрахунок витрат на здійснення науково-дослідної роботи з розробки системи комп'ютерного зору для розпізнавання цілей безпілотних апаратів

5.2.1 Витрати на оплату праці

Основна заробітна плата розробників, яка розраховується за формулою 5.1.

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів за місяць, 21 днів;

t — число днів роботи розробника (дослідника).

Після проведення розрахунків норм часу на кожен етап роботи та визначення місячних окладів відповідних фахівців були отримані витрати на основну заробітну плату. Результати розрахунків зведено до таблиці 5.4**.** Ця таблиця є ключовим документом для формування собівартості розробки та подальшого визначення її ціни.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	40000	1904,76	37	70476,19
Програміст	35000	1666,67	37	61666,66
Пілот-тестувальник	40000	1904,76	7	13 333,32
Всього				145 476,17

Так як в даному випадку розробляється програмно-апаратний комплекс тому, розробник виступає одночасно робітником програмного і апаратного забезпечення, він виконує лабораторні тести але для повноцінних випробувань потрібна людина яка має високі навички пілотування FPV дрона, який завершить тестування розроблюваного продукту.

5.2.2 Розрахунок додаткової заробітної плати розробників

Додаткову заробітну плату прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників, функція 5.2.

$$З_д = З_о \cdot 15\ \% / 100\ \%\ , \qquad (5.2)$$

$$З_д = (145\ 476{,}17 \cdot 15\ \% / 100\ \%\ ) = 21\ 821{,}42\ (\text{грн.})$$

5.2.3 Відрахування на соціальні заходи

Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати,функція 5.3.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\%\ , \qquad (5.3)$$

$$Н_з = (145\ 476{,}17 + 21\ 821{,}42) \cdot 22\ \% / 100\ \% = 36\ 805{,}46\ (грн.)$$

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі вироби, які використовують при дослідженні нового технічного рішення, розраховуються, згідно з їхньою номе нклатурою, за формулою 5.4 . До розрахунку включаються всі складові, необхідні для збирання та тестування прототипу, а також допоміжні матеріали, без яких проведення експериментів є неможливим. Такий підхід дозволяє точно визначити фактичну собівартість елементної бази та оцінити економічну доцільність подальшого вдосконалення.

$$К_В = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j, \qquad (5.4)$$

де H_j — кількість комплектуючих j-го виду, шт.;

$Ц_j$ — покупна ціна комплектуючих j-го виду, грн;

K_j — коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15).

Проведені розрахунки приведені у таблиці 5.5.

Таблиця 5.5 — Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт	Ціна за 1 шт, грн	Коеф. трансп. витрат	Сума, грн
Raspberry Pi 5 (4GB)	1	4000	1.1	4400
Камера Raspberry Pi CSI	1	500	1.1	550
Модуль живлення DC–DC	1	350	1.1	385
Шлейфи, кабелі, дроти	1 комплект	300	1.15	345

Монтажні матеріали	1 комплект	200	1.15	230
Всього	—	—		5910

$$К_В = 5910 \text{ грн.}$$

5.2.5 Спецустаткування для наукових та експериментальних робіт

До статті належать витрати на виготовлення та придбання спецустаткування, верстатів, пристроїв, інструментів, приладів, стендів, апаратів, іншого спецобладнання, необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення. До балансової вартості устаткування окрім прейскурантної вартості входять витрати на його транспортування і монтаж, тому ці витрати бе руться додатково в розмірі 10…12% від вартості устаткування.

Балансову вартість спецустаткування розраховують за формулою 5.5.

$$В_{спец} = \sum_{i=1}^{к} Ц_i \cdot С_{пр.i} \cdot К_i \,, \qquad (5.5)$$

де $Ц_i$ — ціна придбання одиниці спецустаткування даного виду, марки, грн;

$С_{пр.i}$ — кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

$К_i$ — коефіцієнт, що враховує доставку, монтаж, налагодження устатку вання тощо, (Кі = 1,10…1,12);

k — кількість найменувань устаткування.

Отримані результати зведені до таблиці 5.6.

Таблиця 5.6 — Витрати на придбання спецустаткування по кожному виду

Найменування	Кількість	Ціна,за 1 шт. грн	Коеф. достав., монтажу та налагодження	Вартість, грн
FPV-дрон (дослідний зразок)	1	9000	1.12	10080
FPV окуляри	1	3000	1.1	3300

Пульт керування	1	3500	1.1	3850
Li-Ion акумулятори	2	2000	1.1	4400
Всього	—	—		21630

$$В_{спец} = 21630$$

5.2.6 Програмне забезпечення для наукових та експериментальних робіт

У рамках даного проєкту використовувалось виключно безкоштовне програмне забезпечення з відкритим кодом:

— Python;

— OpenCV;

— YOLOv8 / YOLO-NAS;

— PyTorch;

— Debian Linux;

— Ultralytics SDK.

Оскільки ПЗ не купувалось, то балансова вартість дорівнює:

$$В_{прг} = 0$$

5.2.7 Амортизація обладнання, програмних засобів та приміщень

До статті відносять амортизаційні відрахування по кожному виду обладнання, устаткування та інших приладів і пристроїв, а також програмного забезпе чення для проведення науково-дослідної роботи, за його наявності в дос лідній організації або на підприємстві. В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою 5.6.

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вих}}{12}, \quad (5.6)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вих}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки зведені в таблиці 5.7.

Таблиця 5.7 — Амортизаційні відрахування по кожному виду обладнання

Найменування	Балансова вартість, грн	Строк використання, років	Використання, міс.	Амортизація, грн
Потужний ПК для навчання моделей	70 000	5	2	2333,33
Ноутбук HP	31 500	5	2	1050
Ноутбук Acer	27 000	5	2	900
Офісне приміщення	250 000	10	2	4166,67
Всього	—	—	—	8550 грн

$$А_{обл} = 8550 \text{ грн.}$$

5.2.8 Паливо та енергія для науково-виробничих цілей

Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою 5.7.

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.7)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2025 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,35 кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 33 \cdot 10,42 = 866,52 \text{ (грн.)}$$

5.2.9 Інші витрати

До статті належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників за формулою 5.8.

$$І_в=(З_о+З_р)\cdot \frac{Н_{ів}}{100\%}, \quad (5.8)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$І_в = (145476,17+21821,42)*(50\% / 100\%) = 83648,80 \text{ (грн.)}$$

5.2.10 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні

(загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників за формулою 5.9.

$$Н_{нзв}=(З_о+З_р)\cdot \frac{Н_{нзв}}{100\%}, \quad (5.9)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = (145476,17+21821,42)* (150\ \% / 100\ \%) = 250946,39 \text{ (грн.)}$$

5.2.11 Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = З_д + Н_з + К_В + В_{спец} + В_{прг} + А_{обл} + В_е + І_в + Н_{нзв} =$$

145476,17+21821,42+36805,46+5910+21630+8550+0+866,52= 575 654,76 грн.

5.2.9 Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення результатів розраховуються за формулою 5.10.

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.10)$$

де η — коефіцієнт, який характеризує етап виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка. Застосування цього коефіцієнта

дозволяє адекватно відобразити ступінь готовності технології та скоригувати розрахункові показники відповідно до реального стану проєкту.

$$ЗВ = 575\,654{,}76 / 0{,}7 = 822364{,}09 грн$$

5.3 Оцінювання важливості та наукової значимості дослідження системи комп'ютерного зору для розпізнавання цілей безпілотних апаратів

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб'єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

У випадку суттєвого вдосконалення програмно-апаратного продукту системи комп’ютерного зору для розпізнавання цілей безпілотних апаратів для використання масовим споживачем, майбутній економічний ефект буде формуватися на основі таких даних формула 5.11.

$$\Delta П_i=(\pm\Delta Ц_0\cdot N+Ц_0\cdot\Delta N)_i\cdot\lambda\cdot\rho\cdot(1-\frac{\vartheta}{100}), \qquad (5.11)$$

де $\pm\Delta Ц_o$ — зміна вартості програмно-апаратного продукту відповідно аналогу від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

Цб — вартість продукту у році до впровадження результатів розробки;

ΔN — зміна кількості споживачів;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333;

р — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2025 році ϑ =18%.

Припустимо, що при прогнозованій ціні 6000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 500 грн. В підрахунках як продукт аналог будем використовувати систему «ЗІР», ціна якої 12500 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 30000 шт; протягом другого року — на 120000 шт; протягом третього року на 480000 шт. Такий динамічний приріст, на 300% щорічно, обумовлений підвищеним попитом на подібні системи у поточних умовах ринку, що враховує фактори безпеки та оборонні потреби. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = ((6500-12500)*10000+6500*30000)*0,8333*0,2*(1-0,18) = 18449262 \text{ грн.}$$

$$\Delta П_2 = ((6500-12500)*10000+6500*120000)*0,8333*0,2*(1-0,18) = 98396064 \text{ грн.}$$

$$\Delta П_3 = ((6500-12500)*10000+6598\ 396\ 06400*480000)*0,8333*0,2*(1-0,18) = 418183272 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 535028598 грн.

Розраховуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, формула 5.12.

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.12)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (18449262 /(1+0,1)^1)+(98396064/(1+0,1)^2)+(418183272/(1+0,1)^3)=$$
$$16772056+ 81305000+ 314225000 = 412302056 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу 5.13.

$$PV = k_{інв} * ЗВ, \quad (5.13)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 822\,364,09= 1\,644\,728,18 \text{ грн.}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме, формула 5.14.

$$Е_{абс} = ПП – PV, \quad (5.14)$$

$$Е_{абс} = 412302056 – 1\ 644\ 728{,}18 = 410657327{,}82 \text{ грн.}$$

Оскільки $Е_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестиційЕ$_в$. Для цього використаємо формулу 5.15.

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.15)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{\frac{410657327{,}82}{1\ 644\ 728{,}18} + 1} - 1 = 5{,}31$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою 5.16.

$$\tau = d + f, \quad (5.16)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{min} = 0.1 + 0.2 = 0.3$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою 5.17.

$$T_{ок} = \frac{1}{E_в}, \quad (5.17)$$

$$T_{ок} = 1 / 5,31 = 0,19 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,19 роки, то фінансування даної наукової розробки є доцільним.

У висновку, економічна частина даної роботи містить розрахунок витрат на розробку нового продукту, сума яких складає 822364,09 гривень. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків розроблений програмний продукт за ціною дешевший за аналог і є високо конкурентоспроможним. Період окупності складе близько 0,19 роки.

ВИСНОВКИ

У даній магістерській роботі було успішно досягнуто поставленої мети та вирішено всі поставлені задачі, визначені у вступі. Проведене дослідження дозволило сформулювати низку важливих висновків щодо оптимізації процесів автоматичного виявлення та відстеження об’єктів у відеопотоці безпілотних літальних апаратів та інтеграції системи комп’ютерного зору з автопілотом FPV-дрона.

У першому розділі здійснено аналіз сучасних методів детекції та трекінгу об’єктів на основі технологій глибокого навчання. Визначено, що комбінований підхід, який поєднує покадрову детекцію з багатокадровим трекінгом, є найбільш ефективним для роботи в умовах обмежених обчислювальних ресурсів та нестабільного середовища застосування. Окрему увагу приділено алгоритмам YOLO та класичним трекерам, які забезпечують високу швидкодію та прийнятну точність у реальному часі.

Другий розділ був присвячений розробці архітектури системи комп’ютерного зору. Було створено програмно-апаратну структуру, що поєднує детекцію об’єктів із застосуванням нейронних мереж та трекінг типу MOSSE для стабільного відстеження цілей. Запропонована архітектура дозволяє гнучко комбінувати ці методи, забезпечуючи баланс між точністю й продуктивністю.

У третьому розділі розроблено та реалізовано прототип системи на основі модулів OpenCV, моделі YOLO та платформи Raspberry Pi. Проведено інтеграцію з польотним контролером через протокол MSP, що дозволило реалізувати автономні режими взаємодії з об’єктом, включно з автоматизованим захопленням і супроводом цілі. Реалізація була спрямована на забезпечення працездатності системи в умовах обмеженої обчислювальної потужності та мінімальних апаратних затримок.

У четвертому розділі проведено експериментальні дослідження точності, швидкодії та стабільності роботи системи. Отримані результати підтвердили можливість досягнення продуктивності рівня від двадцяти п’яти до тридцяти п’яти

кадрів за секунду на вбудованій платформі, що свідчить про ефективність застосованих методів оптимізації — квантизації ваг, прунінгу та структурного спрощення моделі. Тестування показало стабільність трекінгу навіть у складних умовах, зокрема при зміні освітлення, рухах камери або частковому закритті об’єкта.

У п’ятому розділі визначено напрями подальшого вдосконалення системи, серед яких — глибша оптимізація нейронних мереж, використання апаратних прискорювачів, розширення функціоналу системи автономного керування БПЛА та застосування більш стійких до перешкод моделей трекінгу.

Апробація результатів наукової роботи було проведено на науковій конференції Актуальні проблеми бойового застосування, експлуатації і ремонту зразків озброєння та військової техніки (2025).

Матеріали роботи доповідались та опубліковувались [1]:

Д. С. Довгань, О. Д. Азаров, Томчук М. А. / Оптимізація систем комп’ютерного зору для використання із обмеженими апаратними платформами // Тези доповіді. Актуальні проблеми бойового застосування, експлуатації і ремонту зразків озброєння та військової техніки (2025). Вінниця 2025 р. Режим доступу:https://conferenes.vntu.edu.ua/index.php/apozbt/apozbt2025/paper/view/26170

Також результати наукової роботи було проведено на науковій конференції Молодь в науці:дослідження, проблеми, перспективи(МН-2026).

Матеріали роботи доповідались та опубліковувались [2]:

Д. С. Довгань, О. Д. Азаров, Томчук М. А. / Інтеграція системи комп’ютерного зору на основі OpenCV з автопілотом FPV-дрона через протокол MSP // Тези доповіді. Молодь в науці:дослідження, проблеми, перспективи(МН-2026). Вінниця 2025 р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26608

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Довгань Д.С., Томчук М.А. «Оптимізація систем комп'ютерного зору для роботи на обмежених апаратних платформах». Матеріали конференції «Актуальні проблеми бойового застосування, експлуатації і ремонту зразків озброєння та військової техніки (2025)», Вінниця, 2025 [Електронний ресурс]. Режим доступу : https://conferences.vntu.edu.ua/index.php/apozbt/apozbt2025/paper/view/26170

2. Довгань Д.С., Томчук М.А. «Інтеграція системи комп’ютерного зору на основі OpenCV з автопілотом FPV-дрона через протокол MSP». Матеріали конференції «Молодь в науці:дослідження, проблеми, перспективи(МН-2026)», Вінниця, 2025 [Електронний ресурс]. Режим доступу : https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/26608

3. Як працюють українські FPV-дрони з системою донаведення [Електронний ресурс]. Режим доступу : https://defence-ua.com/news/jak_pratsjujut_ukrajinski_fpv_droni_z_sistemoju_donavedennja_v_merezhi_opublikuvane_perehoplene_video_vikoristannja-18976.html

4. Класифікація безпілотних літальних апаратів [Електронний ресурс]. Режим доступу : https://openarchive.nure.ua/server/api/core/bitstreams/878899d8-b7a7-4481-af22-9835c0748ba0/content

5. Комп’ютерний зір — технологія, яка допомагає створити безпечне середовище [Електронний ресурс]. Режим доступу : https://proit.ua/kompiutiernii-zir-tiekhnologhiia-iaka-dopomaghaie-stvoriti-biezpiechnie-sieriedovishchie-na-virobnitstvi/

6. Документація Google Cartographer [Електронний ресурс]. Режим доступу : https://google-cartographer.readthedocs.io/en/latest/

7. ImageNet Classification with Deep Convolutional Neural Networks. Krizhevsky, A., Sutskever, I., Hinton, G. [Електронний ресурс]. Режим доступу : https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

8. What is object detection? [Електронний ресурс]. Режим доступу : https://www.ibm.com/think/topics/object-detection

9. Object Tracking [Електронний ресурс]. Режим доступу : https://www.ultralytics.com/glossary/object-tracking

10. Технологія інтелектуального аналізу відеопотоку для автоматичного розпізнавання цілей системи керування вогнем на основі машинного навчання [Електронний ресурс]. Режим доступу : https://ric.zp.edu.ua/article/view/312781

11. What is a neural network? [Електронний ресурс]. Режим доступу : https://www.ibm.com/think/topics/neural-networks

12. A Comprehensive Review of Deep Learning: Architectures, Recent Advances, and Applications [Електронний ресурс]. Режим доступу : https://www.mdpi.com/2078-2489/15/12/755?utm_source=chatgpt.com

13. A comprehensive review of object detection with traditional and deep learning methods [Електронний ресурс]. Режим доступу : https://www.sciencedirect.com/science/article/abs/pii/S0165168425001896?utm_source=chatgpt.com

14. SSD: Single Shot MultiBox Detector [Електронний ресурс]. Режим доступу : https://arxiv.org/abs/1512.02325?utm_source=chatgpt.com

15. Comparing Ultralytics YOLO11 vs previous YOLO models [Електронний ресурс]. Режим доступу : https://www.ultralytics.com/blog/comparing-ultralytics-yolo11-vs-previous-yolo-models

16. SORT Explained: Real-Time Object Tracking in Python [Електронний ресурс]. Режим доступу : https://blog.roboflow.com/sort-explained-real-time-object-tracking-in-python/

17. Що таке метод CamShift? [Електронний ресурс]. Режим доступу : https://sing.borshch.cx.ua/ukraincyam/shho-take-metod-camshift.html

18. Вступ до OpenCV. Комп'ютерний зір [Електронний ресурс]. Режим доступу : https://itmaster.biz.ua/programming/vision/opencv.html

19. Як TensorFlow і PyTorch розвивають глибоке навчання [Електронний ресурс]. Режим доступу : https://www.agiliway.com/uk-ua/yak-tensorflow-i-pytorch-rozvyvayut-glyboke-navchannya/

20. MediaPipe Solutions guide [Електронний ресурс]. Режим доступу : https://ai.google.dev/edge/mediapipe/solutions/guide

21. Getting started with your Raspberry Pi [Електронний ресурс]. Режим доступу : https://www.raspberrypi.com/documentation/computers/getting-started.html

22. Ml models and optimization strategies for enhancing the performance of classification on mobile devices [Електронний ресурс]. Режим доступу : https://science.lpnu.ua/sites/default/files/journal-paper/2024/dec/37216/974-82_0.pdf

23. YOLO11 on Raspberry Pi: Optimizing Object Detection for Edge Devices [Електронний ресурс]. Режим доступу : https://learnopencv.com/yolo11-on-raspberry-pi/

24. How to Export to NCNN from YOLO11 for Smooth Deployment [Електронний ресурс]. Режим доступу : https://docs.ultralytics.com/integrations/ncnn/

25. FPV autonomous operation with Betaflight and Raspberry Pi [Електронний ресурс]. Режим доступу : https://medium.com/illumination/fpv-autonomous-operation-with-betaflight-and-raspberry-pi-0caeb4b3ca69

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри

обчислювальної техніки

_____ д.т.н., проф. О.Д. Азаров

«25» вересня 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

«Система комп’ютерного зору для розпізнавання цілей безпілотних апаратів»

Науковий керівник к.т.н., доцент

_____ Томчук М. А.

Студента групи 2КІ-24м

_____ Довгань Д. С.

Вінниця 2025

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність дослідження зумовлена необхідністю підвищення рівня автономності та точності безпілотних літальних апаратів. Одним із шляхів вирішення цього завдання є розробка інтелектуальних систем сприйняття, які здатні самостійно аналізувати навколишнє середовище та приймати рішення щодо керування БПЛА в реальному часі. Сучасні умови застосування безпілотних систем вимагають високої надійності роботи навіть за наявності радіоперешкод і ситуацій радіозатінення. Розробка ефективних алгоритмів комп'ютерного зору дозволяє підвищити стійкість апаратів до зовнішніх впливів та зменшити залежність від каналів зв'язку. Це особливо актуально для України, де безпілотні технології активно використовуються у оборонній сфері;

1.2 Наказ ректора університету № 313 від 24 вересня 2025 року про затвердження теми магістерської кваліфікаційної роботи.

2 Мета МКР і призначення розробки

2.1. Мета роботи — полягає у розширенні функціональних можливостей безпілотних літальних апаратів шляхом розробки та експериментальної оцінки програмно-апаратної системи комп'ютерного зору для автоматичного виявлення та відстеження об'єктів у відеопотоці БПЛА. Система забезпечує високу точність розпізнавання у реальному часі та інтеграцію з системами керування дроном**.**

2.2. Призначення розробки — створення системи комп'ютерного зору, яка підвищує автономність БПЛА, виконуючи пошук, супровід і автоматичне наведення дрона на ціль.

3 Вихідні дані для виконання МКР

3.1. Апаратною основою системи є вбудована платформа Raspberry Pi 5, що використовується для реалізації обчислювального модуля комп'ютерного зору та інтеграції з FPV-дроном.

3.2. Програмні засоби, застосовані у розробці: OpenCV, модель детекції YOLO, мова програмування Python. Функціональним призначенням системи є автоматичне виявлення та розпізнавання цілей у режимі реального часу з подальшою інтеграцією алгоритмів у систему керування FPV-дроном.

4 Вимоги до виконання МКР

4.1. Провести аналіз та дослідження методів розпізнавання і відстеження об’єктів у відеопотоці БПЛА.

4.2. Провести аналіз технологій та вибір архітектури системи комп’ютерного зору.

4.3. Розробити систему комп’ютерного зору для розпізнавання цілей безпілотних апаратів.

4.4. Провести експериментальне дослідження та оцінка ефективності системи.

4.5. Виконати економічне обґрунтування доцільності розробки.

5 Етапи МКР та очікувані результати

Етапи виконання МКР та очікувані результати наведені у таблиці А.1.

6 Матеріали, що подаються до захисту МКР

Пояснювальна записка магістерської кваліфікаційної роботи, графічні та ілюстративні матеріали, анотації українською та англійською мовами, протокол попереднього захисту, відгук наукового керівника, рецензія (відзив) опонента, довідка про проходження перевірки антиплагіату, нормоконтроль про відповідність оформлення вимогам ВНТУ.

7 Порядок контролю виконання та процедури захисту МКР

Виконання етапів МКР контролюється науковим керівником відповідно до календарного плану. Захист МКР проводиться на засіданні Державної екзаменаційної комісії, затвердженої наказом ректора ВНТУ.

8 Вимоги до оформлення МКР відповідають таким нормативним документам

— ДСТУ 3008:2015 «Звіти у сфері науки і техніки. Структура та правила оформлення»;

— ДСТУ 8302:2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання магістерських робіт зі спеціальності 123 «Комп'ютерна інженерія»;

— Положення СУЯ ВНТУ-03.02.02-П.001.01:21 «Про кваліфікаційні роботи на другому (магістерському) рівні освіти».

Таблиця А.1 — Етапи магістерської кваліфікаційної роботи та очікувані результати

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі	10.09.2025р.	28.09.2025р.	Розділ 1
2	Огляд сучасних технологій комп'ютерного зору для безпілотних систем	28.09.2025р.	17.10.2025р.	Розділ 2
3	Аналіз методів розпізнавання і відстеження об'єктів у відеопотоці	18.10.2025р.	30.10.2025р.	Розділ 3
4	Розробка архітектури та програмної реалізації системи комп'ютерного зору	01.11.2025р.	08.11.2025р.	Розділ 4
5	Розрахунок економічної частини	09.11.2025р.	15.11.2025р.	Розділ 5
6	Апробація та впровадження результатів дослідження	16.11.2025р.	10.12.2025р.	Тези доповіді
7	Оформлення пояснювальної записки, графічного матеріалу і презентації	19.11.2025р.	25.11.2025р.	ПЗ, графічний матеріал
8	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	26.11.2025р.	08.12.2025р.	Оформлені документи

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Система комп'ютерного зору для розпізнавання цілей безпілотних апаратів

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

- ☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Завідувач кафедри ОТ Азаров О.Д. _____
(прізвище, ініціали, посада) (підпис)

Гарант освітньої програми Мартинюк Т.Б. _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Захарченко С.М. _____
(прізвище, ініціали) (підпис)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Томчук М.А. к.т.н., доцент. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Довгань Д.С.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг модуля логування messages.py

```
import logger
main_autopilot_started = {
    "log_info": "[AUTOPILOT STARTED]",
    "console": "\033[93m[AUTOPILOT STARTED]\033[0m"}
main_stopping_threads = {
    "log_info": "[STOPPING THREADS]",
    "console": "\033[93m[STOPPING THREADS]\033[0m"}
command_executor_done = {
    "log_info": "thread \'Command executor\', DONE.",
    "console": "thread \033[93mCommand executor\033[0m, DONE at {0}."}
command_executor_connected = {
    "log_info": "MSP connection is established: '{0}'",
    "console": "MSP connection is established: \033[92m{0}\033[0m"}
command_executor_executing_command = {
    "log_debug": "Executing command: {0}, Priority: {1}, Body: {2}",
    "console": "Executing command: {0}, Priority: {1}, Body: {2}"}
command_monitor_log = {
    "log_debug": "Monitoring: {0}",
    "console": "Monitoring: {0}"}
command_telemetry_log = {
    "log_debug": "Telemetry: {0}",
    "console": "Telemetry: {0}"}
command_monitor_current_rssi_and_battery = {
    "log_info": "RSSI: {0}, signal: {1}, battery voltage: {2}V",
    "console": "\033[93m[RSSI: {0}, signal: {1}, battery voltage: {2}V]\033[0m"}
command_telemetry_current_speed = {
    "log_info": "Current speed [{0} m/s]",
    "console": "current speed: - \033[93m[{0} m/s]\033[0m"}
command_telemetry_current_altitude = {
    "log_info": "Current altitude [{0} m]",
    "console": "current altitude: - \033[93m[{0} m]\033[0m"}
command_telemetry_autopilot_state = {
    "log_info": "Autopilot state: {0}",
    "console": "AUTOPILOT STATE: \033[95m{0}\033[0m"}
bee_state_changed_to = {
    "log_info": "Bee state changed to [{0}]",
    "console": "Bee state changed to [{0}]"}
initializing_autopilot = {
    "log_info": "Initializing autopilot",
    "console": "Initializing autopilot"}
command_deliver_we_are_going_forward = {
```

```
    "log_info": "[We are going forward for 2 sec]",
    "console": "[We are going forward for 2 sec]"}
command_deliver_we_are_delivering = {
    "log_info": "[We are delivering package]",
    "console": "[We are delivering package]"}
command_deliver_mission_completed = {
    "log_info": "[MISSION COMPLETED]",
    "console": "\033[93mMISSION COMPLETED\033[0m"}
telemetry_requestor_done = {
    "log_info": "thread \'Telemetry requestor\', DONE.",
    "console": "thread \033[93mTelemetry requestor\033[0m, DONE at {0}."}
empty_pilot_process_done = {
    "log_info": "thread \'Empty pilot process\', DONE.",
    "console": "thread \033[93mEmpty pilot process\033[0m, DONE at {0}."}
empty_pilot_process_connecting = {
    "log_info": "Empty pilot: attempting to connect with '{0}'",
    "console": "Empty pilot: attempting to connect with \033[91m{0}\033[0m"}
empty_pilot_process_connected = {
    "log_info": "Empty pilot: connected with '{0}'",
    "console": "Empty pilot: connected with \033[92m{0}\033[0m"}
telemetry_process_connecting = {
    "log_info": "Telemetry: attempting to connect with '{0}'",
    "console": "Telemetry: attempting to connect with \033[91m{0}\033[0m"}
telemetry_reconnection = {
    "log_info": "Telemetry: attempting to reconnect because of exception: '{0}'",
    "console": "Telemetry: attempting to reconnect because of exception: \033[91m{0}\033[0m"}
telemetry_process_connected = {
    "log_info": "Telemetry: connected with '{0}'",
    "console": "Telemetry: connected with \033[92m{0}\033[0m"}
fatal_error = {
    "log_fatal": "{0}"}
main_autopilot_finished = {
    "log_info": "[AUTOPILOT FINISHED]",
    "console": "\033[93m[AUTOPILOT FINISHED]\033[0m"}
def display(msg, params=[]):
    if msg.get('log_info'):
        logger.log_message(None, msg['log_info'].format(*params), 'info')
    if msg.get('log_debug'):
        logger.log_message(None, msg['log_debug'].format(*params), 'debug')
    if msg.get('log_fatal'):
        logger.log_message(None, msg['log_fatal'].format(*params), 'fatal')
    if msg.get('console'):
        print(msg['console'].format(*params))
```

ДОДАТОК Г

Лістинг модуля телеметрії telemetry.py

```
import time
import autopilot
import messages
import router
import definitions as vars

def telemetry_requestor(stop_command):
    while autopilot.state['connection'] == False and not stop_command.is_set():
        print("SEND TELEMETRY MSP_RC")
        try:
            time.sleep(5)
            messages.display(messages.telemetry_process_connecting, [vars.companion_computer])
        except Exception as e:
            messages.display(messages.fatal_error, [e])
            pass

    if not stop_command.is_set():
        messages.display(messages.telemetry_process_connected, [vars.companion_computer])

    while not stop_command.is_set():
        try:
            print("REQUEST MSP_RC")
            # Частіше опитуємо RC для миттєвого виявлення AUX2 (10 Hz)
            router.put_command(router.Command(0, 'TELEMETRY', {'target':'MSP_RC'}))
            if int(time.time()) % 2 == 0:
                router.put_command(router.Command(2, 'TELEMETRY', {'target':'MSP_ALTITUDE'}))
                router.put_command(router.Command(2, 'MONITOR', {'target':'MSP_ANALOG'}))
            # Коротка пауза між ітераціями
            time.sleep(0.1)
        except Exception:
            time.sleep(0.05)
            pass

    stopped_time = time.strftime("%H:%M:%S, %Y, %d %B", time.localtime())
    messages.display(messages.telemetry_requestor_done, [stopped_time])
```

ДОДАТОК Д

Лістинг модуля відправки команд на дрон commands.py

```
import serial
import time
import autopilot
import definitions as vars
import msp_helper as msp
command_delays =  {
    'go_forward': 2,
    'deliver': 1
}
command_target_ids = {
    'MSP_ANALOG': msp.MSP_ANALOG,
    'MSP_ALTITUDE': msp.MSP_ALTITUDE,
    'MSP_RC': msp.MSP_RC
}
serial_port = {}
def wait_for_execution(target, delay=0):
    if delay == 0:
        delay = command_delays.get(target)
    time.sleep(delay)
def get_target_id(target):
    return int(command_target_ids.get(target))
def connect():
    global serial_port
    try:
        serial_port = serial.Serial(
            vars.companion_computer,
            vars.companion_baud_rate,
            timeout=1)
        print(f"✅ Послідовний порт {vars.companion_computer} відкрито")
        return True
    except Exception as e:
        print(f"❌ Помилка відкриття порту: {e}")
        return False
def disconnect():
    serial_port.close()
def reboot():
    disconnect()
    time.sleep(1)
    connect()
def set_row_rc(roll, pitch, yaw, throttle, servo_aux):
    # ROLL/PITCH/THROTTLE/YAW/AUX1/AUX2/AUX3/AUX4
```

```
    data = [roll, pitch, throttle, yaw, 0, servo_aux, 0, 0]
    msp.send_msp_command(serial_port, msp.MSP_SET_RAW_RC, data)
    msp_command_id, payload = msp.read_msp_response(serial_port)
    if msp_command_id != msp.MSP_SET_RAW_RC:
        return False
    return True
def copter_init():
    # connect()
    return set_row_rc(
        vars.default_roll,
        vars.default_pitch,
        vars.default_yaw,
        vars.default_throttle,
        vars.default_servo_aux2)
def telemetry(target):
    msp_target_command_id = get_target_id(target)
    msp.send_msp_request(serial_port, msp_target_command_id)
    time.sleep(0.1)
    msp_command_id, payload = msp.read_msp_response(serial_port)
    if msp_command_id == msp_target_command_id:
        return payload
    return {}
def prepare_go_forward(throttle):
    set_row_rc(
        vars.default_roll,
        vars.default_pitch,
        vars.default_yaw,
        int(throttle),
        vars.default_servo_aux2)
def go_forward():
    print("🎯 ВИКОНУЄТЬСЯ go_forward()")
    set_row_rc(
        vars.default_roll,
        vars.default_pitch + 200,
        vars.default_yaw,
        int(autopilot.state['throttle']),
        vars.default_servo_aux2)
    wait_for_execution('go_forward')
    set_row_rc(
        vars.default_roll,
        vars.default_pitch,
        vars.default_yaw,
        int(autopilot.state['throttle']),
        vars.default_servo_aux2)
```

```
    wait_for_execution('go_forward')
    return True
def deliver():
    print("🎯 ВИКОНУЄТЬСЯ deliver()")
    set_row_rc(
        vars.default_roll,
        vars.default_pitch,
        vars.default_yaw,
        int(autopilot.state['throttle']),
        2000) # 2000 to open the servo-device (to deliver the bomb)
    wait_for_execution('deliver')
    return True
def test_msp_commands():
    print("□ ТЕСТ MSP КОМАНД...")
    # Проста команда - встановити нейтральні значення
    test_data = [1500, 1500, 1000, 1500, 0, 1000, 0, 0]
    success = msp.send_msp_command(serial_port, msp.MSP_SET_RAW_RC, test_data)
    if success:
        print("✅ ТЕСТ ПРОЙДЕНО - MSP команди працюють")
    else:
        print("❌ ТЕСТ НЕ ПРОЙДЕНО - MSP команди не працюють")
```

ДОДАТОК Е

Лістинг запуску автопілота main.py

```
import os
import threading
import messages
import router
import telemetry
import empty_pilot
import time
# from cv_controller import cv_controller
from cv_integration import cv_controller
# INIT
bee_commands = router.command_queue
stop_command = threading.Event()
# BEE-EPT (Empty pilot)
os.system('clear')
messages.display(messages.main_autopilot_started)
executor_thread = threading.Thread(target=router.command_executor,
                                   args=[stop_command])
telemetry_thread = threading.Thread(target=telemetry.telemetry_requestor,
                                   args=[stop_command])
empty_pilot_thread = threading.Thread(target=empty_pilot.empty_pilot_process,
                                   args=[stop_command])
cv_controller.start()
executor_thread.start()
telemetry_thread.start()
empty_pilot_thread.start()
router.put_command(router.Command(0,'INIT',{}))
input('Press enter to stop process...\n')
messages.display(messages.main_stopping_threads)
cv_controller.stop()
stop_command.set()
executor_thread.join()
telemetry_thread.join()
empty_pilot_thread.join()
messages.display(messages.main_autopilot_finished)
```

ДОДАТОК Ж

Блок-схема алгоритму розпізнавання та трекінгу цілей

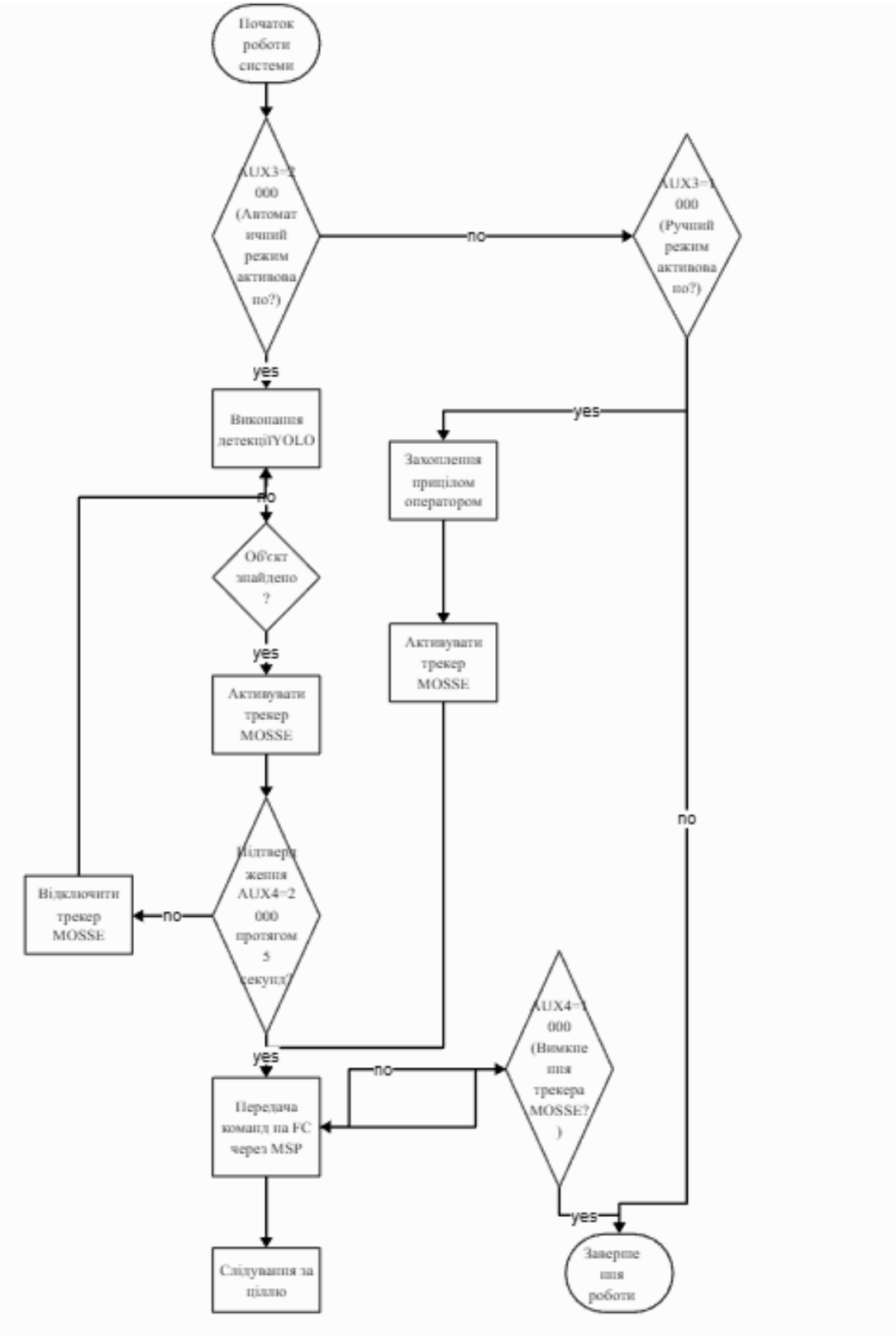

Рисунок Ж.1 — Блок-схема алгоритму роботи системи комп’ютерного зору

ДОДАТОК И

Фото прототипа модуля комп’ютерного зору

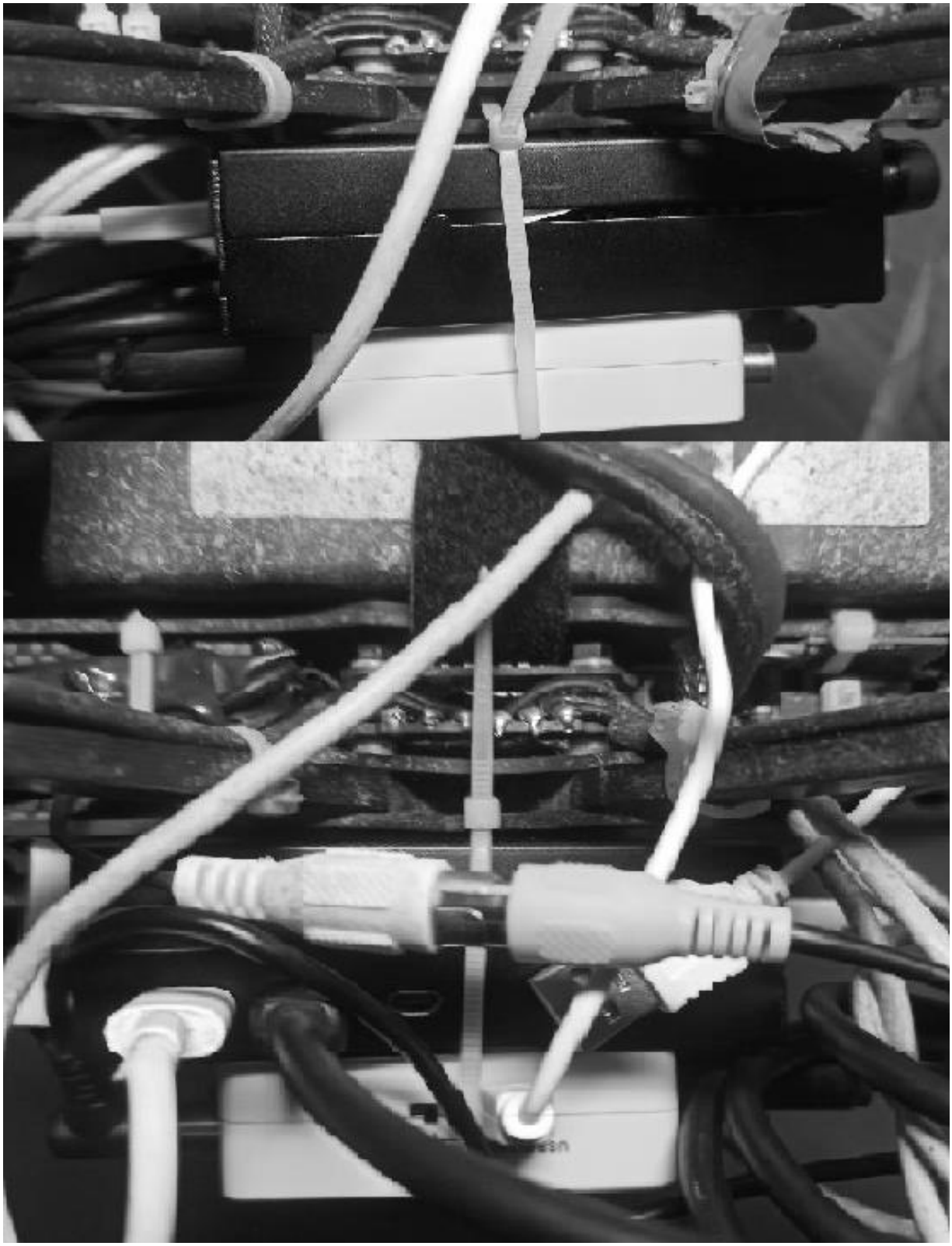

Рисунок И.1 — Фото прототипа модуля комп’ютерного зору, встановлений на FPV-дроні