Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка інтелектуальної автоматичної системи виявлення та стеження на базі поворотної платформи
(тема)

Виконав:
здобувач четвертого року навчання,
групи ІТШІ-21-4
Ян-Даніїл Зіновєєв
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Штучний інтелект
(повна назва освітньої програми)
Керівник ас. Микола Черненко
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ _____ Олег ЗОЛОТУХІН
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Штучного інтелекту

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Штучний інтелект
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Зіновєєву Яну-Даніїлу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтелектуальної автоматичної системи виявлення та стеження на базі поворотної платформи

затверджена наказом університету від 19 травня 20 25 р. № 378Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 червня 20 25 р.

3. Вихідні дані до роботи Документація мови програмування C++, документація бібліотеки OpenCV, документація бібліотеки TensorFlow Lite, документація Arduino

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі

2) Проектування системи та обґрунтування вибору технологій

3) Реалізація проекту автономної системи стеження

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	19.05.2025	виконано
2	Аналіз предметної галузі	22.05.2025	виконано
3	Аналіз актуальних існуючих рішень	25.05.2025	виконано
4	Проектування системи та обґрунтування вибору	28.05.2025	виконано
	технологій		
5	Реалізація проекту автономної системи стеження	02.06.2025	виконано
6	Захист перед ЕК	18.06.2025	виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Микола Черненко
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 72 с., 20 рис., 1 дод., 36 джерел.

ВІДЕОПОТІК, КОМП'ЮТЕРНИЙ ЗІР, МОДЕЛЬ, МОДУЛЬ, НЕЙРОННА МЕРЕЖА, ОБ'ЄКТ, ПЛАТФОРМА, ТРЕКІНГ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єкт дослідження – процес автоматичного стеження за об'єктами в режимі реального часу.

Предмет дослідження – автоматичне стеження за визначеним об'єктом із застосуванням методів комп'ютерного зору та машинного навчання. Основну увагу приділено розробці програмно-апаратного рішення з модульним підходом, а саме: алгоритмам обробки відеопотоку, трекінгу об'єктів, стабілізації зображення та розпізнавання, а також розробці поворотної платформи на базі контролера Arduino.

Мета дослідження – вирішення описаної проблеми за рахунок створення автоматичної системи, здатної здійснювати виявлення та стеження за об'єктами в режимі реального часу.

Методи дослідження – у ході дослідження застосовувалися методи комп'ютерного зору відповідно до сучасних підходів, реалізованих у бібліотеці OpenCV та стандартах ISO/IEC 15938 (MPEG-7). Для виявлення об'єктів застосовано модель YOLOv5, яка забезпечує розпізнавання зображень в режимі реального часу. Обчислення оптимізовано з використанням TensorFlow Lite для забезпечення ефективності на вбудованих системах. Також використовувалися методи трекінгу об'єктів, стабілізації зображення, експериментальні методи та методи програмної оптимізації.

ABSTRACT

Bachelor’s thesis contains: 72 pp., 20 fig., 1 ann., 36 references.

ARTIFICIAL INTELLIGENCE, COMPUTER VISION, MODEL, NEURAL NETWORK, OBJECT, MODULE, PLATFORM, TRACKING, VIDEO STREAM.

Object of the research – the process of automatic real-time object tracking.

Subject of the research – automatic tracking of a specified object using computer vision and machine learning methods. The primary focus is on developing a hardware-software solution based on a modular approach, including: video stream processing algorithms, object tracking, image stabilization and recognition, as well as the development of a pan-tilt platform based on an Arduino controller.

Purpose of the research – to address the described problem by creating an automatic system capable of detecting and tracking objects in real time.

Research methods – during the research, computer vision methods were applied in accordance with modern approaches implemented in the OpenCV library and the ISO/IEC 15938 (MPEG-7) standards. The YOLOv5 model was used for object detection, providing real-time image recognition. Computations were optimized using TensorFlow Lite to ensure efficiency on embedded systems. Additionally, methods for object tracking, image stabilization, experimental approaches, and software optimization techniques were employed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Автономно – виконання дії без втручання людини або без постійного підключення до зовнішніх систем керування;

Відеопотік – безперервна послідовність кадрів, що передається або записується в реальному часі, наприклад, з камери спостереження;

Нейронна мережа – математична модель, що імітує роботу біологічного мозку, складається з штучних нейронів, організованих у шари. Використовується для розпізнавання образів, класифікації, прогнозування та інших задач машинного навчання;

Оптичний потік – векторне поле, що описує зміщення пікселів між двома послідовними кадрами відео, використовується для аналізу руху об’єктів;

Поворотна платформа – механізм, що забезпечує обертання об’єкта навколо осі, часто використовується в системах спостереження, робототехніці та автоматизованих механізмах;

Програмний комплекс – формальна назва програмного забезпечення (software), тобто програм, які керують комп’ютерними системами або пристроями;

Трекінг – процес відстеження руху об’єкта в просторі або на зображенні з використанням алгоритмів комп’ютерного зору;

Arduino – програмована мікроконтролерна платформа, яка використовується для створення електронних пристроїв, автоматизації, робототехніки та інших інтерактивних систем;

AI – Artificial Intelligence – штучний інтелект.

ВСТУП

У сучасному світі автоматизація стає невід'ємною частиною нашого життя. Від систем «розумного будинку» до складних військових рішень – технології розпізнавання, стеження та аналізу зображень активно впроваджуються в різні сфери діяльності. Зростаючий обсяг інформації, потреба в оперативному реагуванні та мінімізація участі людини у рутинних або небезпечних процесах зумовлюють розвиток інтелектуальних систем обробки візуальних даних. Серед іншого, одним із перспективних напрямів є створення систем автоматичного виявлення та стеження, які можуть працювати без безпосереднього втручання людини, знижуючи ризик помилок та підвищуючи ефективність виконання завдань.

Автоматичні системи стеження застосовуються у великій кількості галузей. Наприклад, у сфері безпеки вони можуть визначати «небажаних гостей» на території приватних або промислових об'єктів, запобігаючи можливим загрозам та/або небажаним наслідкам. У виробництві такі системи можуть контролювати якість продукції, виявляючи зіпсовані або дефектні вироби, які є результатом виробництва, або продукти, які є сировиною, а при використанні маніпуляторів – навіть самостійно їх сортувати. У військовій сфері, що останнім часом набуває окремої важливості, подібні рішення здатні виконувати функції автоматичних систем наведення для безпілотних літальних апаратів, таких як дрони-камікадзе, що значно підвищує точність і швидкість виконання бойових завдань та не залежать від людей.

Технічна сторона подібних рішень є доволі складною та вимагає використання передових технологій. Обгрунтовано вибір технологій в кваліфікаційній роботі, що застосовуються для реалізації, а саме C++ у поєднанні з бібліотекою OpenCV для обробки відеопотоку та бібліотекою TFLite як інструменту роботи з нейронними мережами. Визначення об'єкта для стеження здійснюється за допомогою нейромереж, які запускаються на

TFLite – одній із популярних платформ для роботи з моделями машинного навчання на вбудованих пристроях. Оскільки рішення реалізується на фізичному обладнанні, використовується платформа Arduino з двома сервоприводами та камерою, що дозволяє забезпечити точне позиціонування камери в просторі.

Одним із ключових викликів у реалізації рішення, що запропоновано, є робота з обмеженнями на формат моделей та, насамперед, фізичні ресурси. Оскільки моделі, які планується використовувати, повинні бути у форматі .tflite uint8, це значно ускладнює вибір нейромереж, а також ускладнює процес їхньої оптимізації або потребує вирішення додаткових задач з трансформування моделей з одного формату в інший, що може негативно вплинути на якість розпізнавання. Крім того, збірка всіх необхідних бібліотек для C++ вручну може бути доволі трудомістким завданням, особливо з урахуванням необхідності підтримки специфічного обладнання. Проте відкритість архітектури та модульний підхід у розробці дозволяють адаптувати систему до різних апаратних платформ, поступово оновлювати її компоненти без потреби у повній перебудові всього проєкту та використовувати її (систему) в інших сценаріях та предметних галузях, що мають споріднену природу проблематики.

Специфікою цього рішення є те, що воно працює повністю автономно. Автономність досягається за рахунок спроможності системи самостійно обробляти позаштатні ситуації на кшталт втрати об’єкту трекером або переривання відеопотоку. Це не класичний застосунок для взаємодії з користувачем, а система, яка після налаштування параметрів запускається та працює автоматично. Користувач лише задає стартові параметри, а далі система автоматично визначає та супроводжує об’єкти в межах свого поля зору. У перспективі можливе розширення функціоналу за рахунок інтеграції додаткових сенсорів, що дозволить розширити можливості системи, наприклад, шляхом комбінування відеоаналітики з іншими типами даних

або додавання інтерфейсу користувача, що дозволить управляти платформою варіативно (автоматично або мануально).

Таким чином, розробка автоматичної системи виявлення та стеження відкриває широкі можливості для її застосування у сфері безпеки, промисловості та військової техніки. Завдяки поєднанню сучасних алгоритмів комп’ютерного зору, вбудованих рішень та глибокого навчання, подібні системи дозволяють значно підвищити ефективність виконання складних завдань, автоматизуючи процеси, які раніше вимагали участі людини. У майбутньому розвиток обчислювальних платформ, нейромережевих алгоритмів та вбудованих рішень ще більше розширить можливості таких систем, зробивши їх ще більш автономними, точними та ефективними.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

В умовах зростаючої потреби в автоматизованому моніторингу навколишнього середовища, зокрема у складних або небезпечних для людини умовах, актуальним постає питання створення систем, здатних самостійно виявляти, ідентифікувати та супроводжувати об’єкти.

У прикладних сферах – таких як охорона стратегічних об’єктів, прикордонний контроль, а також мобільні платформи військового призначення – виникає необхідність у компактних, енергоефективних рішеннях, здатних функціонувати в реальному часі та з мінімальним втручанням оператора. Водночас такі системи мають бути достатньо гнучкими для адаптації до різних сценаріїв застосування, що вимагає модульної побудови та відкритої архітектури. Таким чином, розробка подібних рішень потребує врахування обмежень апаратного забезпечення, особливостей середовища, а також забезпечення сумісності програмних компонентів у межах єдиної системи.

Мета даної роботи та її предметна галузь полягає у вирішенні задачі автоматичного виявлення та стеження за об’єктами. Основна ціль розроблюваної системи – автоматизувати процеси, що потребують аналізу візуальної інформації в режимі реального часу, зменшуючи вплив людського фактора та підвищуючи надійність і точність функціонування.

Такий підхід може бути корисним у різних сферах – від промислового контролю якості до реалізації військових задач, зокрема автономного керування дронами чи периметрової охорони.

Загальна структура системи, що розроблюється, поділяється на функціональні блоки (рисунок 1.1).

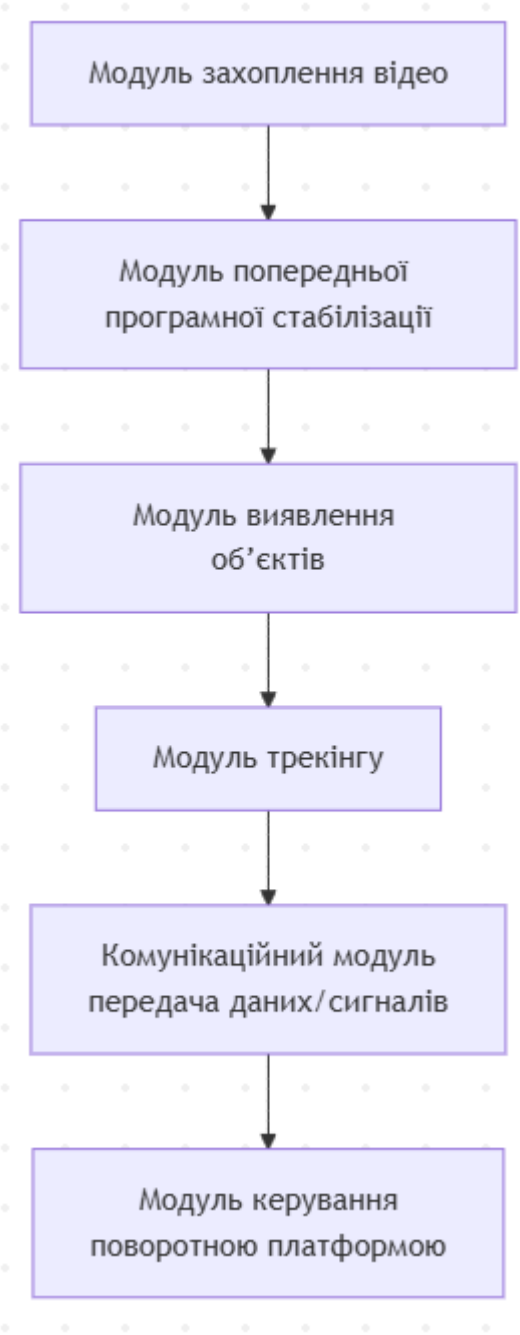

Рисунок 1.1 – Структура системи, що розроблюється

Модульний підхід дозволяє забезпечити масштабованість та адаптивність рішення до конкретних умов використання.

Для вирішення даної проблематики потрібно вирішити кілька задач:

– задача розпізнавання;

– задача стабілізації зображення;

– задача стеження.

Задача розпізнавання включає в себе використання нейронної мережі для отримання стартового положення об’єкта.

Нейронна мережа – це математична модель та обчислювальна система, що імітує принципи роботи біологічного мозку для обробки та аналізу даних. Вона складається з великої кількості штучних нейронів, організованих у шари (вхідний, приховані та вихідний), які взаємодіють між собою через вагові коефіцієнти.

Нейрон – основний структурний елемент нейронної мережі, що приймає один або кілька вхідних сигналів, обробляє їх за допомогою вагових коефіцієнтів і передає вихідний сигнал через активаційну функцію [1].

Шар НМ (нейронної мережі) – сукупність нейронів, які функціонують на одному рівні обчислень у нейронній мережі. Всі нейрони одного шару зазвичай мають однаковий тип активаційної функції та зв'язані з нейронами інших шарів [1].

Вхідний шар – перший шар нейронної мережі, який отримує початкові дані (вхідні ознаки) та передає їх у мережу без змін або з попередньою обробкою [1].

Приховані шари – шари нейронів, розташовані між вхідним і вихідним шарами, які виконують проміжні обчислення, перетворення та витяг основних характеристик даних. Чим більше прихованих шарів, тим складнішою є модель (глибока нейронна мережа) [1].

Вихідний шар – останній шар нейронної мережі, що формує остаточний прогноз або класифікацію, залежно від задачі (наприклад, ймовірності класів у класифікації або числове значення в регресії) [1].

Задача стабілізації зображення включає в себе створення спеціального алгоритму, який буде зменшувати вплив механічного руху камери для кращої роботи стеження. Це досягається шляхом аналізу зміщень зображення між послідовними кадрами та застосування коригувальних трансформацій, таких як афінні або перспективні перетворення.

Стабілізація зображення – технологія, що використовується для зменшення розмиття зображення, спричиненого тремтінням камери під час фотографування або відеозйомки. Система стабілізації компенсує незначні горизонтальні зміщення та кутові нахили камери або іншого фото- чи відеопристрою [2].

Коригувальні трансформації зображення – це процеси або операції, спрямовані на виправлення або покращення певних параметрів зображення, таких як колір, яскравість, контрастність тощо, без безпосереднього змінення вихідних пікселів [3].

Афінне перетворення (Affine transformation) – взаємооднозначне лінійне відображення площини або простору на себе, при якому паралельні прямі залишаються паралельними, а відношення довжин відрізків на одній прямій зберігається [4].

Стабілізація може базуватися на оптичному потоці, ключових точках або інерціальних сенсорах, що дозволяє компенсувати випадкові коливання та підвищити точність системи стеження.

Оптичний потік – векторне поле, яке описує видимий рух яскравих об'єктів у зображенні, спричинений рухом об'єктів або камери. Використовується для аналізу руху в комп'ютерному зорі [5].

Ключові точки – характерні точки в зображенні, які можна надійно ідентифікувати та використовувати для порівняння різних зображень або відстеження об'єктів. Вони часто використовуються в алгоритмах комп'ютерного зору для визначення особливостей зображення [6].

Задача стеження має на увазі використання так званого механізму трекінгу. Трекінг об’єктів – це алгоритмічний підхід до відстеження об’єктів у послідовності кадрів відео. Він дозволяє визначати положення, швидкість і траєкторію руху цільового об'єкта, зберігаючи його ідентифікацію навіть при зміні масштабу, освітлення або частковому перекритті іншими об'єктами [7].

За останні роки люди все більше і більше прагнуть автоматизувати всі системи для полегшення життя і процеси визначення та стеження не є виключенням. Наприклад, з ростом доходу на душу населення та можливості людей робити своє життя кращим питання захисту майна або навіть самих себе стає все більш гострим через втручання сторонніх людей. Для цього люди наймають сторожів, які б охороняли власність господаря, але є непоодинокі випадки, коли через сильну втому, неуважність або якісь інші фактори охоронці не виконували або виконували не якісно свою роботу. Тому створення системи яка буде автоматично визначати порушення приватних кордонів, класифікувати загрози та відстежувати їх (загрози) в реальному часі є питанням актуальним у громадських цілях.

З іншого боку не менш актуальним питанням є військова тематика, яка також є виразним прикладом використання даної системи, наприклад у дронах-камікадзе. Буває так, що на «останній милі» оператор втрачає зв’язок з дроном через те, що для попадання в ціль потрібна дуже низька висота, яка може бути проблемою для радіосигналу, тому створення автоматичної системи яка б не потребувала ані оператора, ані зв’язку з зовнішнім ресурсом є нагальним питанням.

Демонстраційна система складатиметься з двох модулів: фізичної поворотної платформи та обчислювального програмного модуля. Фізична платформа відповідатиме за надання програмному модулю відеопотоку та виконання корекції положення, тоді як програмний модуль оброблятиме отримане відео, здійснюватиме виявлення та стеження за об'єктами, а також генеруватиме команди для корекції положення платформи. Хоча ця архітектура не є класичним клієнт-серверним підходом, умовно можна розглядати фізичну платформу як постачальника даних і виконавця команд, а програмний модуль – як обчислювальний центр, що приймає, аналізує та видає результати обробки.

Ключовими вимогами до отриманої системи є швидкість роботи алгоритму, точність виявлення, стійкість алгоритму стеження до

непередбачуваних рухів та адаптація алгоритму у разі втрати об’єкту. Швидкість роботи зумовлена розділенням задач на різні обчислювальні модулі. Тобто задача контролю платформи лежить на одному незалежному модулі, а задача обробки на іншому. Точність виявлення забезпечується за рахунок використання згорткових нейронних мереж (Convolutional neural network – CNN). Стійкість алгоритму стеження до непередбачуваних рухів та адаптація алгоритму у разі втрати об’єкту забезпечується універсальним алгоритмом, який зможе обробляти подібні ситуації.

Згорткова нейронна мережа – це клас глибоких штучних нейронних мереж, спеціально розроблених для обробки даних із сітчастою структурою, таких як зображення. Вони ефективно застосовуються в задачах комп'ютерного зору, зокрема для розпізнавання та класифікації об'єктів на зображеннях [8].

Платформа складатиметься з модуля контролю Arduino, двох сервоприводів, що забезпечать достатній огляд та камери, яка буде надавати відеопотік до модуля обробки. Система повинна бути стійка до різних траєкторій руху об’єктів та забезпечувати безперервний відеопотік.

Усі розглянуті аспекти слід враховувати під час розробки системи виявлення та стеження на основі поворотної платформи. Важливо забезпечити безперервне вдосконалення системи, оновлення моделей і покращення алгоритму стеження та обробки в цілому, що сприятиме підвищенню точності та ефективності роботи системи.

Аналізуючи ринок відеокамер та різноманітних готових автоматизованих систем можна побачити, що вже існують готові рішення, які б виконували поставлену задачу. Однак, якщо провести детальне вивчення то можна зробити висновки, що ці рішення обмежені конкретною предметною галуззю з точки зору виявлення та є достатньо дорогими. Реалізація, що пропонується, є універсальним рішенням, завдяки модульній та відкритій структурі та яке буде незалежне від предметної галузі та буде

більш дешевим в реалізації, однак зазначимо, що створення подібної системи потребує чималих знань різних аспектів проектування та розробки.

1.2 Огляд актуальних існуючих рішень

Автоматичні системи виявлення та стеження набули широкого застосування як у сфері безпеки, так і в споживчих технологіях. Вони використовуються в камерах відеоспостереження, дронах, потокових відеосервісах і навіть у різних проектах на базі мікроконтролерів. Основна ідея таких систем – автоматично знаходити об'єкти, розпізнавати їх і стежити за ними за допомогою поворотних платформ або алгоритмів комп’ютерного зору. Пропоную розглянути найбільш доступні рішення, які можна придбати у вільному доступі.

Почнемо з камер серії Hikvision (рисунок 1.2) [9].

Рисунок 1.2 – IP камера Hikvision DS-2CD2T25FHWD-I8 (4.0)

Багато сучасних поворотних камер мають вбудовану функцію автотрекінгу, що дозволяє автоматично визначати рухомі об'єкти та стежити за ними. Такі камери можуть використовувати алгоритми комп'ютерного зору для розпізнавання людей, автомобілів або інших об'єктів.

Плюси:

– легко інтегруються у системи відеоспостереження;

– висока якість зображення (Full HD, 4K);

– інфрачервоне підсвічування для роботи вночі;

– широкий діапазон повороту та масштабування.

Мінуси:

– обмежена швидкість повороту;

– автотрекінг може втрачати об'єкт, якщо він зникає з кадру;

– висока вартість професійних моделей.

Далі пропоную розглянути споживчі камери з AI-стеженням. Досить відомим прикладом буде камера Insta360 Link (рисунок 1.3) [10].

Рисунок 1.3 – Веб-камера Insta360 Link

Це невеликі камери з вбудованими алгоритмами штучного інтелекту для автоматичного відстеження обличчя або людини. Вони зазвичай використовуються для відеоконференцій, блогінгу або стрімінгу.

Плюси:

– компактні та прості у використанні;

– швидке реагування на рухи;

– доступна ціна порівняно з професійними системами.

Мінуси:

– обмеженість у використанні поза приміщенням;

– невеликий радіус огляду та дальність дії;

– можуть помилятися в умовах поганого освітлення.

Як було відмічено рішення, що мною розроблюється, може використовуватись як у цивільних, так і у воєнних цілях. Пропоную розглянути одне з таких воєнних рішень як iFlight iGimbal, яке використовується як трекер для FPV-дронів (рисунок 1.4) [11].

Рисунок 1.4 – IFLIGHT 3-Axis CNC gimbal

Ці рішення часто використовуються в FPV-дронерів для автоматичного стеження за об'єктом або оператором. Вони можуть кріпитися на платформу або триногу та використовувати інерціальні сенсори або GPS для стеження.

Плюси:

– дуже швидкий відгук на зміну положення;

– можливість використання в динамічних умовах;

– гнучкість у налаштуванні параметрів.

Мінуси:

– вимагають досвіду для налаштування;

– дорожчі за прості камери з автотрекінгом;

– часто не мають вбудованої аналітики, що потребує додаткового софту.

Як бачимо, існує багато подібних рішень різного призначення, характеристик та, що не менш важливо, обмежень. Також в рамках цього розділу пропонується зробити порівняння готових рішень, які наведені вище, з тим, що пропонується як результат цієї роботи.

Отож, відповідно, пропоную розглянути саморобне рішення на базі Arduino (Arduino + сервоприводи + AI + OpenCV)

Плюси:

– низька вартість компонентів;

– гнучкість у налаштуванні алгоритмів та функціоналу;

– можливість інтеграції зі штучним інтелектом.

Мінуси:

– потребує глибоких знань програмування та електроніки;

– менша швидкість та точність у порівнянні з готовими рішеннями.

Вочевидь, саморобне рішення буде програвати продуктам великих багатомільйонних корпорації у точності та швидкості, але відкрита архітектура, модульність та в деяких випадках низька вартість це нівелює та навіть є плюсом. Не треба далеко ходити за прикладами, а візьмемо очевидне та нагальне: дрон-камікадзе. Найголовніше в цьому випадку є баланс між найнижчою собівартістю з можливих та ефективності роботи тому, що техніка буде знищена відповідно до специфіки роботи, тому використання набагато дорожчих, хоч і точніших аналогів тут невиправдана, бо це призведе до великих фінансових витрат, мінімізація яких є одним з основних цілей мого проекту.

Окремо варто додати, що у багатьох готових рішень нема програмної стабілізації кадру, є тільки механічна. Це також є дуже важливим плюсом двеного рішення тому, що якщо його застосувати з пристроєм, який має механічну стабілізацію, то буде отримано ще краще рішення.

1.3 Технологічні особливості розробки системи

Розробка вище зазначеної системи передбачає низку технічних та алгоритмічних викликів. Основні труднощі пов’язані з апаратною частиною, програмним забезпеченням, точністю виявлення та трекінгу об’єктів, а також стабільністю роботи системи в різних умовах експлуатації.

Необхідно розібратись детальніше та покроково:

– апаратні обмеження. Система використовує камеру, яка забезпечує зображення, з яким можна працювати, проте її продуктивність дуже обмежена та залежить від умов освітлення та динамічного діапазону сцени. В умовах недостатнього освітлення або високої контрастності можливе погіршення якості виявлення об’єктів. Крім того, використання двох сервоприводів на мікроконтролері Arduino накладає обмеження на швидкість та точність повороту камери, що може впливати на ефективність трекінгу, тому першим викликом в розробці подібного рішення є пошук балансу між швидкістю та точністю задля забезпечення найкращого результату;

– обчислювальна продуктивність. Основна обробка відеопотоку на даний момент виконується на зовнішньому комп’ютері, що дозволяє застосовувати складні алгоритми глибинного навчання. Однак, як буде описано пізніше, система розробляється з перспективою повного запуску на одній платі для забезпечення повної автономності та ізоляції від навколишнього обчислювального модуля та взагалі контролю. Саме тому був обраний формат .tflite, який дає змогу запуску моделей на мобільних пристроях, але через це знижується точність розпізнавання через

квантизацію моделі до uint8 (бо саме така розмірність моделі підтримується нейросопроцессорами). Це може призводити до втрати деталей та зниження якості класифікації. Класифікація – це багаторівневий, послідовний поділ обсягу поняття з метою систематизації, поглиблення та отримання нових знань стосовно членів поділу. Вона передбачає розподіл предметів, явищ або понять на класи чи групи за спільними ознаками або властивостями, що сприяє впорядкуванню та кращому розумінню різноманіття об'єктів [12]. Крім того, алгоритм KCF, що використовується для трекінгу, має обмеження в розпізнаванні швидких рухів об’єкта та змін його форми, що впливає на стабільність відстеження. Тому, як можна зрозуміти, детальне налаштування системи є другим викликом через умову дуже обмежених ресурсів;

– синхронізація роботи модулів. Для ефективної роботи системи необхідно синхронізувати відеопотік, розпізнавання об’єктів та рух платформи. Затримки у передачі даних через USB-інтерфейс можуть спричинити асинхронну реакцію системи, що ускладнює плавне стеження за об’єктами;

– алгоритмічні виклики. Використання TensorFlow Lite для розпізнавання об’єктів та OpenCV для трекінгу вимагає оптимізації роботи нейронної мережі під обмежені ресурси. Важливим аспектом є вибір ефективного механізму обробки кадрів у реальному часі, щоб забезпечити високу швидкість обробки без значного споживання ресурсів. Враховуючи це розробка ведеться на мовах C та C++, що є ще одним викликом тому, що треба власноруч налаштовувати та збирати всі бібліотеки;

– стабілізація зображення. Незначні вібрації та зміщення камери можуть негативно впливати на точність трекінгу, особливо при використанні сервоприводів. Необхідно розробити механізм цифрової або механічної стабілізації, який мінімізує вплив таких коливань та покращує якість відстеження. Також необхідно погодити конфігурації трекінгу,

алгоритму стабілізації та контроль сервоприводів для забезпечення найкращих результатів.

1.4 Постановка задачі

Місією даної роботи є розробка багатофункціонального адаптивного рішення для вирішення задач визначення та стеження за об’єктами, що передбачає використання готових рішень для розпізнавання та трекінгу об’єктів, оптимізацію їх роботи на обмежених обчислювальних ресурсах та інтеграцію з апаратною частиною системи, що дозволить вберегти життя, здоров’я та сили людей. Для розв'язання поставленої задачі треба виконати наступні пункти:

– реалізувати модуль виявлення об’єктів. Програмна частина повинна адаптуватись до моделі, яка буде подаватись на вхід, підлаштовуючи всі необхідні параметри системи без необхідності зміни програмного коду для будь-яких модулів;

– розробити модуль трекінгу об’єктів. Система повинна правильно реагувати на переміщення об’єкту, незважаючи на його траєкторію, та на втрату об’єкта трекером;

– розробити програмний модуль стабілізації зображення. Реалізувати алгоритми компенсації вібрацій за допомогою цифрової стабілізації;

– розробити фізичну поворотну платформу. Оптимізувати матеріали та компоненти для забезпечення міцності, малої ваги та дешевизни розробки, покращити алгоритми управління сервоприводами для більш плавного руху;

– забезпечити стабільне функціонування апаратної та програмної частин. Реалізувати механізм автоматичного перезапуску у разі збою окремих компонентів, зменшити затримки у передачі даних між модулями.

Таким чином, основна задача полягає у створенні ефективної системи детекції та трекінгу об’єктів, що забезпечує високу точність і стабільність роботи в умовах експлуатації з обмеженими ресурсами.

1.5 Висновки за розділом

Проведене дослідження демонструє актуальність розробки доступної, гнучкої та модульної системи автоматичного виявлення й стеження за об’єктами, яка може бути адаптована до широкого спектра задач – від цивільного відеоспостереження до військового застосування в автономних системах.

Основна мета полягала у створенні універсального рішення, здатного працювати в режимі реального часу, мінімізуючи людський фактор і знижуючи витрати на впровадження.

Запропонована система має чітко структуровану архітектуру, яка включає модулі розпізнавання, стабілізації зображення, трекінгу, програмного керування та фізичну поворотну платформу. Для реалізації цих функцій застосовуються сучасні алгоритми комп’ютерного зору, згорткові нейронні мережі (YOLOv5), трекери (KCF), методи цифрової стабілізації зображення (оптичний потік, ключові точки) та платформа на базі Arduino.

Перевагами розробленого рішення є універсальність, низька вартість, відкритість до адаптації, підтримка програмної стабілізації та можливість глибокого налаштування під специфічні вимоги. У порівнянні з комерційними або військовими системами, дана розробка поступається в деяких аспектах точності й готовності до негайного використання, проте виграє завдяки своїй доступності та технологічній гнучкості.

Разом з тим, у ході реалізації виявлено низку технологічних викликів, пов’язаних з апаратними обмеженнями (обмежена швидкість Arduino, якість відео), потребою в оптимізації алгоритмів під слабші обчислювальні платформи, складністю синхронізації між модулями та необхідністю

точного управління реальним обладнанням. Окрему увагу було приділено забезпеченню стабільної роботи всієї системи, в тому числі реалізації алгоритмів перезапуску та зменшення затримок у передаванні даних.

Таким чином, розділ підтверджує життєздатність обраної концепції та визначає чіткий вектор подальшої роботи в напрямку оптимізації, масштабування та впровадження розробленої системи в прикладні задачі.

2 ПРОЕКТУВАННЯ СИСТЕМИ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1 Схема роботи системи

Дана система не передбачає взаємодію з людиною (оператором) після запуску, тому дуже важливо перед початком розробки спроектувати систему, яка буде мати можливість працювати повністю автономно, незважаючи на помилки.

Для побудови схем для зручності будемо використовувати мову Mermaid. Mermaid – це мова розмітки для створення діаграм та графіків у текстовому вигляді. Її часто використовують у документації, особливо в GitHub, Notion, Obsidian, Typora, MkDocs, Docusaurus тощо. Вона проста, читається як псевдокод і дуже зручна для спільної роботи або швидкої візуалізації.

Для того, щоб розкрити сутність того, як працює система розглянути наступні схеми (рисунки 2.1, 2.2, 2.3, 2.4).

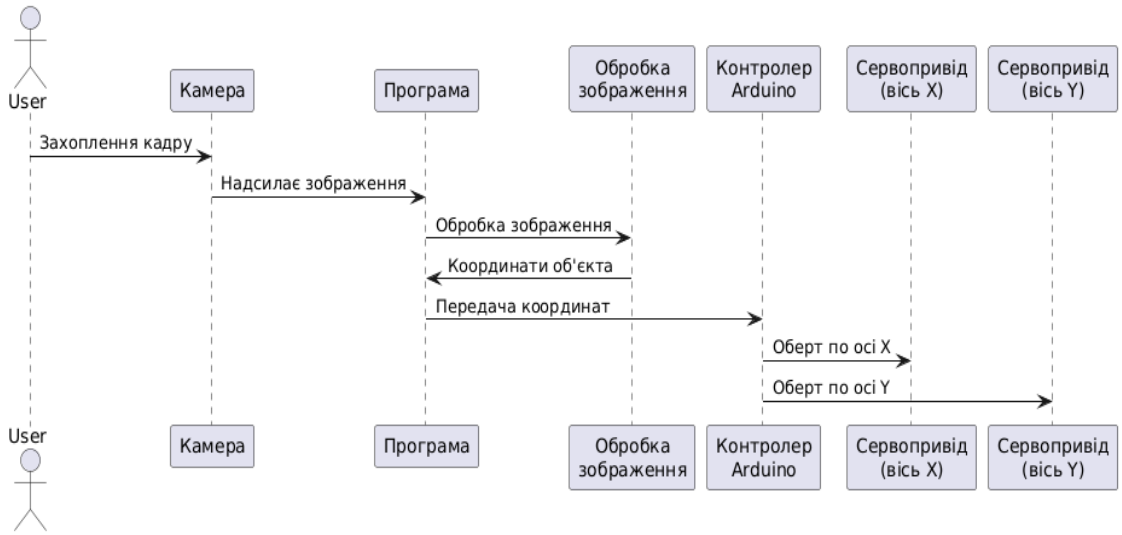

Рисунок 2.1 – Схема роботи апаратної частини (Sequence diagram)

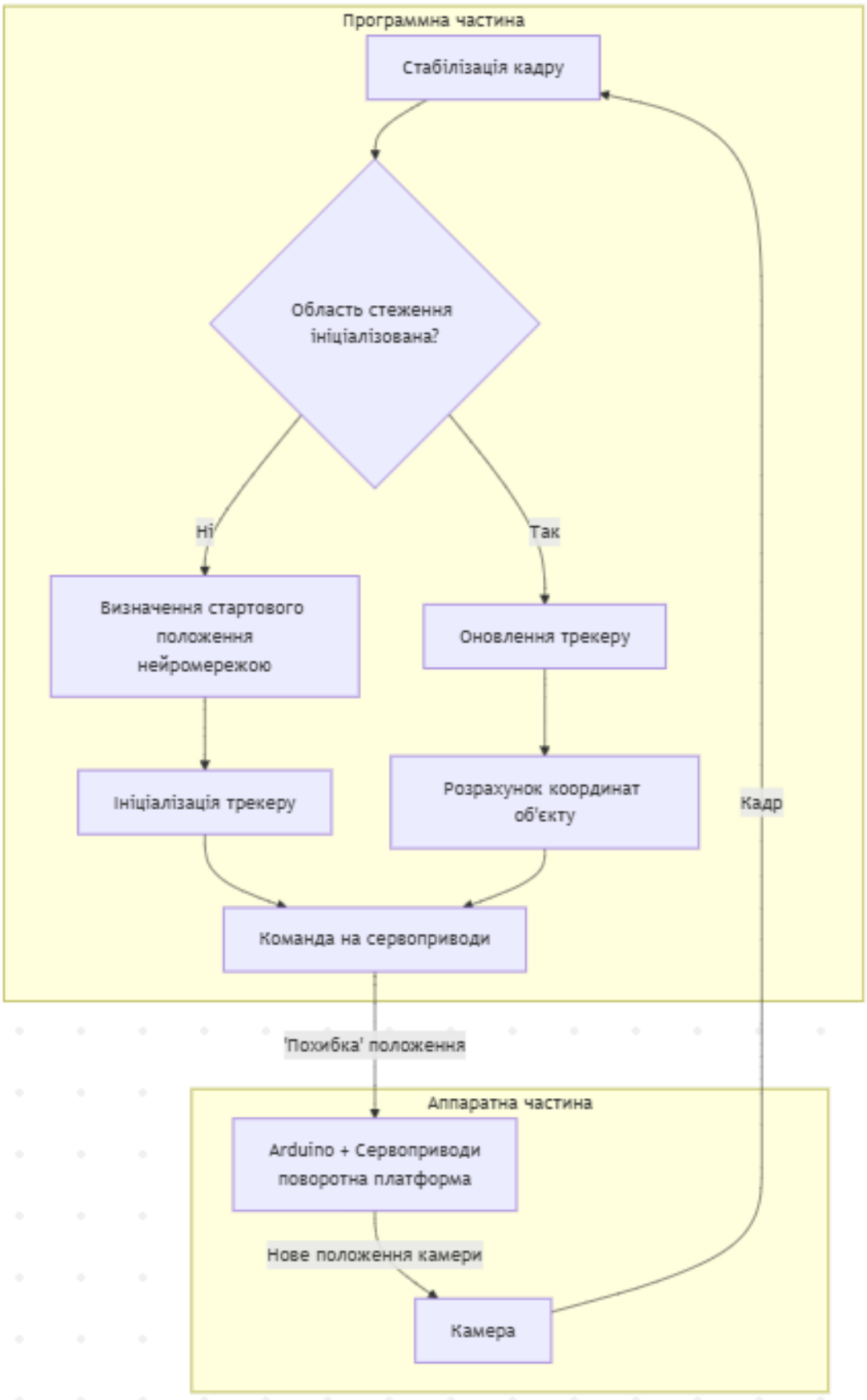

Рисунок 2.2 – Функціональна діаграма системи

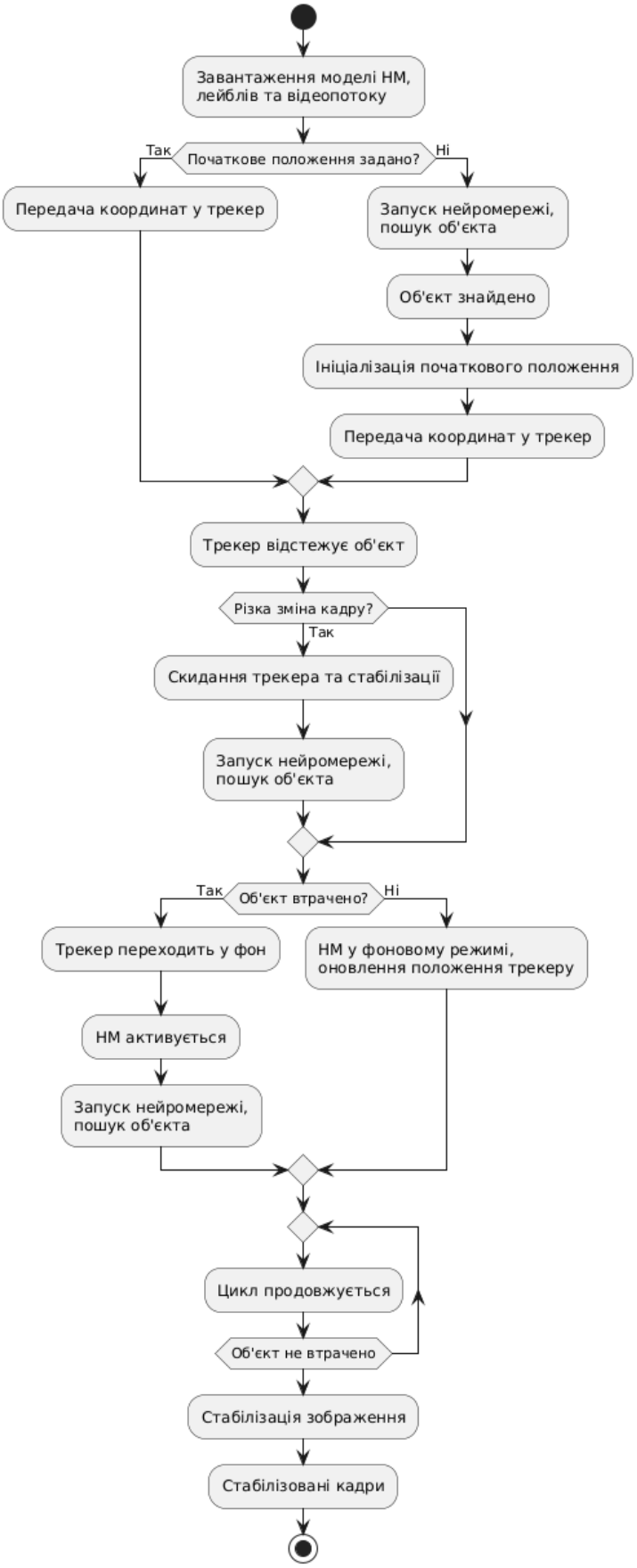

Рисунок 2.3 – Схема роботи програмної частини (Activity diagram)

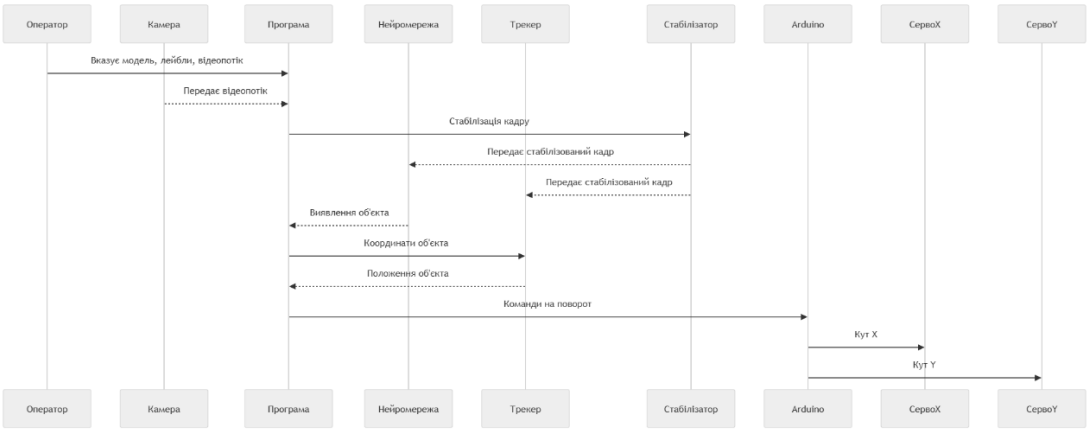

Рисунок 2.4 – Схема взаємодії апаратної та програмної частин (Sequence diagram)

Очевидно, для забезпечення автономності системи був розроблений алгоритм, який покриватиме втрату об’єкта та його знаходження. Також розроблене рішення стабілізації забезпечить кращу роботу трекеру за рахунок зменшення «шуму» та будь-яких фізичних взаємодій з апаратною частиною, які не є критичними. Водночас запровадження програмної стабілізації створює кілька проблем для відеопотоку та сервоприводів.

Схеми є повними, подальші зміни не будуть мати жодного ефекту на архітектуру, тому вважатимемо їх остаточними та на які буде спиратись вся розробка.

2.2 Вибір та обґрунтування компонентів програмної частини системи

Для розробки програмної частини було обрано наступні технології для реалізації такі компонентів різних рівнів:

- C++;
- OpenCV;
- TensorFlow Lite;
- Моделі YOLOv5 у форматі TFLite;
- Трекер KCF.

C++ – це універсальна мова програмування, розроблена Б'ярне Страуструпом як розширення мови C з додаванням об'єктно-орієнтованого підходу. Вперше випущена в 1985 році, вона стала основою для багатьох сучасних технологій, таких як ігрові рушії, веб-браузери, операційні системи та фінансові системи. Основні характеристики C++ включають простоту, незалежність від платформи, низькорівневий доступ до ресурсів та високу швидкість виконання [13].

OpenCV (Open Source Computer Vision Library) – це кросплатформна бібліотека з відкритим кодом для комп'ютерного зору та машинного навчання, яка використовується для розробки реальних програм з обробки зображень, комп'ютерного зору та розпізнавання образів.. Спочатку розроблена компанією Intel, вона тепер підтримується спільнотою розробників під егідою OpenCV Foundation. OpenCV використовується для розробки реальних програм з обробки зображень, комп'ютерного зору та розпізнавання образів [14].

TensorFlow Lite – це полегшена версія фреймворку машинного навчання TensorFlow, оптимізована для виконання на мобільних та вбудованих пристроях. Вона надає інструменти та бібліотеки для розробки та розгортання моделей машинного навчання на пристроях з обмеженими ресурсами, таких як мобільні телефони та пристрої Інтернету речей (IoT) [15].

Вибір інструментів для розробки системи був обумовлений обмеженими ресурсами. Оскільки метою є інтеграція всіх компонентів – апаратних і програмних – на одну плату для створення автономної та незалежної системи, першочерговим завданням є мінімізація використання ресурсів.

Вибір мови програмування C++ зумовлений її здатністю ефективно контролювати пам'ять та забезпечувати високу швидкість виконання програм. На відміну від інтерпретованих мов, таких як Python, яку б скоріш обрали для виконання подібних до поставлених нами задач, C++ є

компільованою мовою, що дозволяє перетворювати код безпосередньо в машинні інструкції, оптимізовані для конкретного апаратного забезпечення.

Це забезпечує швидше виконання програм та зменшує затримки, що є критично важливим для вбудованих систем з обмеженими ресурсами. Практичні експерименти підтверджують значну різницю в продуктивності між C++ та Python. У одному з таких досліджень програма на C++, яка інкрементувала значення змінної 1 000 000 000 разів, виконувалася приблизно за 1,7 секунди, тоді як аналогічна програма на Python потребувала близько 82 секунд для виконання тієї ж задачі [16]. До того ж враховуючи те, що програмна реалізація передбачатиме зв’язок з мікроконтролером, то використання С подібних мов є найбільш доречним. З огляду на наведені фактори, використання мови C++ є оптимальним вибором для розробки системи, оскільки вона забезпечує ефективне управління пам'яттю та високу швидкість виконання програм, що є критично важливими для вбудованих систем з обмеженими ресурсами.

Використання бібліотеки OpenCV обумовлене необхідністю ефективної роботи із зображеннями та відео. OpenCV (Open Source Computer Vision Library) є однією з найпопулярніших бібліотек комп'ютерного зору, що підтримує широкий спектр мов програмування, включаючи C++, Python, Java та MATLAB, і сумісна з різними операційними системами, такими як Windows, Linux, macOS, Android та iOS [17].

Незважаючи на наявність альтернатив, OpenCV залишається вибором багатьох розробників завдяки своїй гнучкості, широкій підтримці спільноти та великому набору функцій для обробки зображень та відео. Крім того, OpenCV оптимізована для реального часу та підтримує апаратне прискорення через CUDA та OpenCL, що робить її ефективною для застосувань, які вимагають високої продуктивності. Це й робить її найбільш підходящим варіантом для використання у цій роботі.

Використання TensorFlow Lite (нині відомого як LiteRT) обумовлено необхідністю ефективної роботи з моделями нейронних мереж засобами C++ та мінімізацією використання апаратних ресурсів.

TensorFlow Lite є високопродуктивним середовищем виконання для розгортання моделей машинного навчання на мобільних та вбудованих пристроях, зокрема на мікроконтролерах з обмеженими обчислювальними можливостями. Його компактний розмір та оптимізація для пристроїв з обмеженими ресурсами дозволяють виконувати інференс моделей з низькою затримкою та зменшеним споживанням енергії.

TensorFlow Lite підтримує кілька мов програмування, включаючи C++, що забезпечує гнучкість у розробці та інтеграції моделей машинного навчання в існуючі системи [19]. Крім того, TensorFlow Lite може використовувати делегати для апаратного прискорення, що дозволяє ефективно задіяти нейронні процесорні блоки (NPU) для прискорення інференсу моделей. З огляду на плани розгортання системи в ізольованому автономному середовищі в майбутньому, було прийнято рішення розробляти рішення з урахуванням цих умов.

Використання NPU значно підвищує продуктивність та знижує енергоспоживання порівняно з виконанням на CPU або GPU [20]. Тому використання TensorFlow Lite є доречним.

Використання YOLOv5 у форматі TFLite uint8 є вигідним для застосувань на мобільних та вбудованих пристроях завдяки зменшеному розміру моделі, швидшому виконанню, високій енергоефективності та використанню апаратного прискорення. YOLOv5 (You Only Look Once, версія 5) є однією з найпопулярніших моделей для задач виявлення об’єктів завдяки високій точності (mAP, mean Average Precision), швидкості обробки, компактності (можна адаптувати під edge-пристрої), підтримці експорту до різних форматів, включно з TFLite [21].

Це дозволяє досягти високої продуктивності та ефективності при розпізнаванні об'єктів на зображеннях, що є критично важливим для

реальних додатків, таких як мобільне відеоспостереження, безпека та автономні системи.

Формат .tflite забезпечує можливість використання моделі бібліотекою TensorFlow Lite, яка дозволить запускати модель на мобільних пристроях, тобто забезпечить використання найменшої кількості ресурсів.

Формат uint8 – обов'язкова вимога для апаратного прискорення на NPU. NPU (наприклад, Edge TPU, APU у MediaTek, або NPU у Huawei Kirin) працюють з quantized (квантизованими) моделями, які представлені у форматі uint8. Це значно зменшує обсяг пам'яті моделі (в 4 рази в порівнянні з float32), дозволяє виконувати операції швидше на NPU (апаратна підтримка int8 / uint8), зменшує споживання енергії [22].

Алгоритм KCF (Kernelized Correlation Filters) належить до кореляційних фільтрів, які використовують FFT (швидке перетворення Фур'є) для швидкої обробки відеокадрів. Завдяки цьому він працює в реальному часі навіть на пристроях із обмеженими ресурсами (мобільні процесори, edge-пристрої), не потребує попереднього навчання, не використовує нейронні мережі, що знижує споживання оперативної пам'яті та обчислювальні вимоги [23].

На відміну від глибоких трекерів (наприклад, Deep SORT, GOTURN), KCF не залежить від GPU/NPU, тому стабільно працює навіть на бюджетних смартфонах, не вимагає тензорних бібліотек (TFLite/NNAPI/OpenVINO), добре поєднується з TFLite-моделями для детекції (наприклад, YOLOv5), де KCF бере на себе відстеження між кадрами. Саме тому був обраний саме цей трекер для використання у нашій системі.

Підсумовуючи можна зробити наступний висновок: використання всіх компонентів системи зумовлене кількома важливими факторами:

- обмежені ресурси мобільних систем;
- швидкість роботи;
- робота в реальному часі.

Саме тому, що вищезазначені компоненти відповідають вимогам та здатні працювати разом, вони були обрані для виконання поставленої задачі.

2.3 Вибір типу моделі нейронної мережі для ідентифікації об’єктів

Вибір конкретної моделі для детекції об'єктів на зображенні є важливим кроком, що визначає точність, швидкість та ефективність системи.

Не зважаючи на те, що система, що розроблюється, не обмежена в предметній області з точки зору об’єкту ідентифікації, вибір самого типу моделі грає велику роль як з точки зору продуктивності так і з точки зору інтерпретації результатів.

У багатьох випадках саме YOLOv5 стає оптимальним варіантом завдяки своїм численним перевагам.

Розглянемо детальніше причини такого вибору, порівнявши YOLOv5 з іншими популярними архітектурами, проаналізувавши його продуктивність та надавши відповідні посилання на джерела.

YOLOv5 належить до сімейства алгоритмів YOLO (You Only Look Once), які відрізняються підходом до детекції об'єктів, як до задачі прямого прогнозування.

На відміну від двоступеневих детекторів, таких як Faster R-CNN, які спочатку генерують регіони-кандидати, а потім класифікують їх, YOLOv5 виконує обидва етапи одночасно, що значно підвищує швидкість обробки.

Порівнюючи з іншими одноступеневими детекторами, такими як SSD (Single Shot MultiBox Detector) та попередні версії YOLO (v3, v4), YOLOv5 пропонує ряд значних покращень:

– висока точність (mAP – mean Average Precision): Завдяки вдосконаленій архітектурі, яка включає такі нововведення, як Cross Stage Partial (CSP) блоки в backbone та neck, а також поліпшену стратегію аугментації даних та функцію втрат, YOLOv5 досягає кращих показників

точності на багатьох бенчмарках порівняно з попередніми версіями YOLO та SSD при зіставній або кращій швидкості [21];

– гнучкість та масштабованість: YOLOv5 пропонує різні варіанти моделі (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x), які відрізняються складністю та кількістю параметрів. Це дозволяє підібрати оптимальну модель під конкретні вимоги до обчислювальних ресурсів та необхідної точності. Найменші моделі (наприклад, YOLOv5n) можуть бути використані на вбудованих пристроях з обмеженими ресурсами, тоді як більші моделі (наприклад, YOLOv5x) забезпечують найвищу точність за рахунок більшої кількості обчислень [21];

– проста структура та легкість у використанні: Кодова база YOLOv5, написана на PyTorch, є добре структурованою та зрозумілою. Це полегшує процес навчання, розгортання та подальшої модифікації моделі. Наявність готових скриптів для навчання, валідації та інференсу робить YOLOv5 зручним інструментом для дослідників та розробників;

– активна спільнота та постійний розвиток: Проєкт YOLOv5 підтримується великою та активною спільнотою, що сприяє швидкому виправленню помилок, додаванню нових функцій та публікації попередньо навчених моделей.

Численні дослідження та порівняльні аналізи демонструють високу продуктивність YOLOv5. Наприклад, у порівнянні з YOLOv4, YOLOv5 часто показує кращі результати за метриками mAP при вищій швидкості обробки кадрів в секунду (FPS) [21], [25].

Важливо зазначити, що фактична продуктивність може варіюватися залежно від використовуваного обладнання, розміру вхідного зображення, складності сцени та оптимізації програмного забезпечення. Для тестування розробленої системи було обрано моделі YOLOv5s, YOLOv5n, YOLOv5s та кастомну модель, яка навчена на базі YOLOv5s.

Також згадаймо про те, що існує новіша версія YOLO – YOLOv8. Причин того, що була обрана саме YOLOv5 декілька:

– зрілість та стабільність. YOLOv5 існує довше, має більш зрілу кодову базу та ширшу спільноту. Це означає кращу документацію, більше прикладів використання та більшу ймовірність знайти вирішення поширених проблем. Стабільність коду є критично важливою при розгортанні на мобільних пристроях, де налагодження може бути складнішим. YOLOv8, хоча швидко розвивається, вона є відносно новішою, і її екосистема все ще формується. Можливі частіші оновлення та зміни, що може вимагати більше зусиль для підтримки [26];

– оптимізація для мобільних пристроїв та .tflite. Розробники YOLOv5 приділяли значну увагу оптимізації моделей для різних платформ, включаючи мобільні пристрої. Існують добре налагоджені процеси експорту в .tflite, а також приклади оптимізації моделей для зменшення розміру та підвищення швидкості інференсу на мобільних GPU та NPU (наприклад, використання квантизації). Хоча YOLOv8 також підтримує експорт в .tflite, рівень оптимізації та наявність готових рішень для мобільних платформ може бути меншим порівняно з YOLOv5 на даний момент. Процес оптимізації може вимагати більше ручної роботи та експериментів [27];

– розмір моделі та швидкість інференсу. Інференс (inference) нейронної мережі – це процес використання вже натренованої моделі для обробки нових (невідомих) даних з метою отримання результату, наприклад: класифікації зображення, розпізнавання об'єкта, перекладу тексту тощо. На відміну від етапу навчання (training), під час інференсу модель не змінює свої ваги – вона просто застосовує вже вивчені закономірності до нових прикладів [28]. YOLOv5 пропонує різні варіанти моделей (n, s, m, l, x), включаючи дуже маленькі та швидкі моделі (наприклад, YOLOv5n, YOLOv5s), які добре підходять для мобільних пристроїв з обмеженими ресурсами. Легше знайти компроміс між точністю та швидкістю, обравши відповідний варіант.YOLOv8 хоча також має різні розміри моделей, навіть найменші з них можуть бути

більшими та повільнішими за еквівалентні за точністю моделі YOLOv5 при конвертації в .tflite та запуску на мобільних пристроях. Це може бути критичним фактором для забезпечення плавної роботи застосунку [28].

2.4 Вибір та обґрунтування компонентів апаратної частини системи

Для розробки апаратної частини було обрано:

- контролер Arduino;
- камера FullHD;
- 2 сервоприводи.

Використання Arduino зумовлено тим, що він є мікроконтролером із відкритим вихідним кодом, який ідеально підходить для керування простими електронними системами, зокрема сервоприводами.

Популярність платформ Arduino пояснюється легкістю у програмуванні, великою спільнотою та широким вибором бібліотек для роботи з різними пристроями.

Для обертальної платформи Arduino виступає основним керуючим елементом, який отримує сигнали з комп’ютера (де виконується комп’ютерне обробка відеопотоку) і передає відповідні команди сервоприводам. Завдяки доступним ШІМ-виходам, він дозволяє точно контролювати положення сервомоторів [24].

Використання саме камери з роздільною здатністю FullHD (1920x1080) забезпечує високу якість зображення, що є критично важливим для точного виявлення об'єктів за допомогою алгоритмів комп’ютерного зору, зокрема OpenCV та TensorFlow Lite.

Висока роздільна здатність дозволяє краще ідентифікувати деталі об'єкта, що підвищує точність як виявлення, так і трекінгу.

Крім того, FullHD-камери зазвичай мають достатню частоту кадрів (30 fps і більше), що забезпечує плавне відео, необхідне для реального часу стеження.

Використання двох сервоприводів зумовлене забезпеченням управління за двома осями платформою: один – для повороту по горизонтальній осі (азимут), другий – по вертикальній (кут підйому).

Сервомотори забезпечують точне позиціонування завдяки вбудованим потенціометрам і замкненій системі керування. Вони швидко реагують на сигнали керування і ідеально підходять для систем, де необхідне позиціонування в межах обмеженого кута (зазвичай до 270°). Це дозволяє платформі плавно й точно слідкувати за об’єктом у реальному часі.

Взаємодія між апаратним та програмним забезпеченням відбуватиметься за допомогою кабелю USB з можливістю перенесення на бездротовий зв’язок.

Також використання обраних компонентів аргументоване фінансовими витратами. Всі обрані компоненти не є дорогими, що дозволяє в деяких сферах використання нехтувати їх цілісністю. Наприклад для використання на дронах-камікадзе першочерговим є баланс між ефективністю та вартістю, що є плюсом нашої системи.

Треба зазначити, що програмна частина не залежить від фізичного використання або предметної області. Вона може бути використана разом із зовсім іншим фізичним обладнанням ніж представлено у цій роботі та не потребуватиме глобальних змін.

2.5 Висновки за розділом

У цьому розділі було детально обґрунтовано архітектурні рішення та вибір технологій, що стали основою для створення автономної системи виявлення та стеження за об’єктами. Ключовим принципом проектування стала повна автономність системи після запуску, що досягнуто завдяки розробці чіткої структурної архітектури, інтеграції механізмів самовідновлення після втрати об’єкта та взаємодії між усіма модулями.

Програмні компоненти були підібрані з урахуванням високої продуктивності та сумісності з вбудованими системами. Мова C++ обрана як базова через її ефективність і низький рівень абстракції, що критично для роботи з мікроконтролерами. OpenCV забезпечує потужні засоби обробки зображень, а TensorFlow Lite дозволяє запускати моделі машинного навчання на ресурсно обмежених пристроях з підтримкою апаратного прискорення. Модель YOLOv5 у форматі TFLite (uint8) стала оптимальним вибором для детекції об’єктів, завдяки своїй збалансованості між точністю та швидкодією. Алгоритм KCF був обраний для реалізації трекінгу через його швидкодію, незалежність від попереднього навчання та низькі вимоги до ресурсів.

Апаратні складові також ретельно підібрані. Використання Arduino як контролера обґрунтоване простотою, відкритістю та широкою підтримкою. Платформа оснащена двома сервоприводами для забезпечення точного керування за двома осями, а камера гарантує належну якість відеопотоку. При цьому враховувалась необхідність зберігати низьку вартість рішення, що особливо важливо для ризикованих сценаріїв, наприклад, у дронах одноразового використання.

Окрему увагу приділено гнучкості програмної реалізації, яка не залежить жорстко від конкретної апаратної платформи, що дозволяє адаптувати систему до різних фізичних конфігурацій без значних змін у кодовій базі. Водночас у процесі проектування було виявлено низку викликів, зокрема вплив програмної стабілізації на синхронність відеопотоку й керування приводами, що потребуватиме подальшої оптимізації.

Загалом, розділ підтверджує технічну доцільність обраних рішень і формує надійну базу для реалізації ефективної, гнучкої та масштабованої системи, здатної працювати в умовах обмежених ресурсів.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ АВТОНОМНОЇ СИСТЕМИ СТЕЖЕННЯ

3.1 Збірка необхідних бібліотек

Після того, як був затверджений остаточний варіант архітектури системи, була почата безпосередня реалізація цієї системи.

Беручи до уваги те, що було вирішено розробляти систему засобами мови C++, то першочерговим завданням було зібрати всі необхідні інструменти. Розробку було вирішено вести на системі Linux Ubuntu 20.04 для досягнення максимально можливого контролю над процесом розробки.

3.1.1 Збірка бібліотеки OpenCV

Розробка була розпочата з того, що треба було зібрати бібліотеку OpenCV з source коду. Після кількох невдалих спроб все ж таки вдалось зібрати робочу версію за допомогою CMake, яка б мала усі необхідні модулі для подальшої роботи. Загальний вигляд налаштувань збірки має наступний вигляд (лістинг 3.1).

Лістинг 3.1 – Конфігурації для збірки бібліотеки OpenCV

```
cmake
-D CMAKE_BUILD_TYPE=Release
-D CMAKE_INSTALL_PREFIX=/usr/local
-D INSTALL_C_EXAMPLES=ON
-D OPENCV_GENERATE_PKGCONFIG=ON
-D BUILD_EXAMPLES=ON
-D
OPENCV_EXTRA_MODULES_PATH=~/opencv_build/opencv_contrib/module
s
-D WITH_GTK=ON
-D WITH_GTK_2_X=ON ..
```

Наведемо опис обраної конфігурації:

«-D CMAKE_BUILD_TYPE=Release» – збирає OpenCV в режимі релізу (Release), тобто з оптимізацією продуктивності, без відлагоджувальної інформації. Це робить бібліотеку швидшою.

«-D CMAKE_INSTALL_PREFIX=/usr/local» – вказує, куди встановити OpenCV після збірки (make install). /usr/local – типове місце для встановлення сторонніх бібліотек.

«-D INSTALL_C_EXAMPLES=ON» – додає C++ приклади до збірки. Це корисно для навчання та тестування функціональності.

«-D BUILD_EXAMPLES=ON» – активує збірку всіх прикладів, які включено у репозиторії OpenCV.

«-D OPENCV_EXTRA_MODULES_PATH= ~/opencv_build/ opencv_contrib/ modules» – підключає модулі з репозиторію opencv_contrib, які не входять у стандартну збірку. Це критично необхідно для використання розширених механізмів трекінгу.

«-D WITH_GTK=ON» – вмикає підтримку GTK (GIMP Toolkit) – кросплатформеного інтерфейсу для GUI. Це потрібно, щоб працювали такі функції, як cv::imshow, cv::waitKey тощо.

«-D WITH_GTK_2_X=ON» – вказує на використання GTK версії 2, що необхідно для сумісності з певними системами або застосунками.

«-D OPENCV_GENERATE_PKGCONFIG=ON» – створює .pc-файл (pkg-config) для OpenCV. Це спрощує використання OpenCV в інших C/C++ проектах – автоматично додаються include, бібліотеки тощо.

Після успішної збірки бібліотеки переходимо до збірки наступного, не менш важливого модуля нашої системи – збірки бібліотеки TensorFlow.

3.1.2 Збірка бібліотеки TensorFlow

У ході вивчення даної бібліотеки виявилось, що TensorFlow не передбачає збірку та використання засобами чистого CMake або Make (має

таку можливість, але потребуватиме набагато більше ресурсів), а має для цього окремо створений інструмент – Bazel, який в свою чергу встановлюється за допомогою інструменту Bazelisk.

Bazel – це інструмент для автоматизації збірки проєктів, створений Google. Він підтримує великі кодові бази, багатомовні проєкти (наприклад, C++, Java, Python), та забезпечує високу продуктивність за рахунок інкрементальної збірки та кешування артефактів [29]. Кешування артефактів – це механізм збереження результатів попередніх збірок, щоб повторно їх не створювати під час наступних збірок.

Bazelisk – це «обгортка» (wrapper) над Bazel, яка автоматично завантажує потрібну версію Bazel, зазначену у проєкті. Це полегшує керування версіями Bazel без необхідності встановлювати їх вручну [30].

Під час встановлення та роботи з Bazel виникло кілька проблем з точки зору Python, а точніше з його версіями. Після додаткового вивчення документації Bazel було з’ясовано, що він використовує Python версій від 3.9 до 3.12.

Після встановлення необхідних інструментів переходимо до реалізації програмної частини системи.

3.2 Реалізація програмної частини

Нагадаємо, що на верхньому рівні абстракції наша система має наступний вигляд (рисунок 3.1).

Рівень абстракції – це ступінь узагальнення або спрощення, з яким розглядається система, об’єкт або процес, приховуючи складні деталі й фокусуючись лише на суттєвому.

Вхідні параметри передаються оператором при запуску програми як аргументи запуску.

Аргументи програми в C++ – це дані, які передаються програмі з командного рядка під час її запуску. Вони дозволяють впливати на

поведінку програми без перекомпіляції, наприклад: передати шлях до файлу, параметри конфігурації тощо [31].

Рисунок 3.1 – Узагальнена схема роботи програмної частини системи (Activity diagram)

Ініціалізація вхідних параметрів програмою має наступний вигляд (лістинг 3.2).

Лістинг 3.2 – Ініціалізація вхідних параметрів системи

```
int main(int argc, char *argv[])
{
    if (argc != 5)
    {
        std::cout << "\nError! Usage: <path to tflite
```

Продовження лістингу 3.2

```
    model> <path to classes names> <path to input video> <path
to output video>\n"
                        << std::endl;
            return 1;
        }
        const std::string model_path = argv[1];
        const std::string names_path = argv[2];
        const std::string video_path = argv[3];
        const std::string save_path = argv[4];
    }
```

Ініціалізація вхідних параметрів системи – це процес задання початкових значень змінним, параметрам або конфігураціям, які потрібні для запуску чи правильної роботи програмної системи. Як бачимо наших параметрів 4, а система перевіряє що параметрів 5. За умовою на вхід програма C++ завжди приймає першим один аргумент – шлях до головного файлу.

Після того, як програма успішно пройшла етап ініціалізації, переходимо до зчитування кадру. Читання кадру з камери здійснюється за допомогою засобів бібліотеки OpenCV. Для цього потрібно першочергово оголосити ресурс звідки будуть братись кадри та самі змінні, які будуть зберігати інформацію про кадри (лістинг 3.3).

Лістинг 3.3 – Відкриття ресурсу, який буде постачальником кадрів

```
    cv::VideoCapture capture;
        if    (all_of(video_path.begin(),    video_path.end(),
::isdigit) == false)
            capture.open(video_path);
        else
            capture.open(stoi(video_path));

        cv::Mat frame, stabilizedFrame;
```

Продовження лістингу 3.3

```
if (!capture.isOpened())
        throw "\nError when reading video steam\n";
```

Після успішного відкриття ресурсу переходимо до ініціалізації програмної стабілізації відео. Для цього необхідно виокремити деякі параметри ресурсу, який надає кадри, та ініціалізувати стабілізацію базуючись на першому кадрі (лістинг 3.4).

Лістинг 3.4 – Ініціалізація стабілізації

```
capture >> frame;
    cv::resize(frame, frame, cv::Size(1080, 720), 
cv::INTER_CUBIC);

    Stabilizer *stabilizer = new Stabilizer();
    stabilizer->init(frame, 
capture.get(cv::CAP_PROP_FRAME_COUNT));

      stabilizedFrame = stabilizer->doStabilize(frame);
```

Останніми етапами підготовки до основної частини програми є ініціалізація трекеру та моделі нейронної мережі.

Трекер має окремий файл реалізації функціоналу, тому в основній програмі необхідно і достатньо лише зробити ініціалізацію об’єкту трекеру без додаткових налаштувань (лістинг 3.5).

Лістинг 3.5 – Ініціалізація об’єкту трекеру

```
Tracking* tracker = new Tracking();
tracker->init();
```

Реалізація цих методів в файлі tracking.cpp має вигляд (лістинг 3.6).

Лістинг 3.6 – Параметри ініціалізації трекеру

```
void Tracking::init(){
    cv::TrackerKCF::Params param;
    param.detect_thresh = 0.15;
    param.sigma = 0.4;
    param.output_sigma_factor = 0.09;
    param.interp_factor = 0.1;
    param.max_patch_size = 5000;
    param.lambda = 0.0002;
    param.pca_learning_rate = 0.22;
    param.split_coeff = false;
    param.wrap_kernel = true;
    tracker = cv::TrackerKCF::create(param);
}
```

Поточні значення були знайдені експериментально як кращі з точки зору балансу швидкості та якості.

Наведемо короткий опис параметрів ініціалізації трекеру.

Detect_thresh – поріг детекції об'єкта. Визначає мінімальну впевненість, з якою об'єкт вважається знайденим. Менше значення → більш чутливий, але більше помилкових спрацювань.

Sigma – параметр Гаусового ядра. Впливає на згладжування та ширину функції подібності у фільтрі.

Output_sigma_factor – співвідношення для визначення ширини вихідного Гаусового ядра. Менше значення → менше розмиття, більш вузьке ядро.

Interp_factor – швидкість оновлення фільтра. Визначає, як швидко адаптується трекер до змін. Від 0 (не адаптується) до 1 (повністю новий шаблон).

Max_patch_size – максимальний розмір фрагменту зображення, який буде використовуватись. Обмежує розмір об'єкту для трекінгу.

Lambda – параметр регуляції. Запобігає перенавчанню фільтра, додає стабільність.

Pca_learning_rate – швидкість навчання PCA (principal component analysis), використовується для зменшення розмірності.

Split_coeff – показник розділення коефіцієнтів ядра. Якщо true – обробка роздільно реальної та уявної частин ядра.

Wrap_kernel – чи обгортати ядро (wrap-around). Дозволяє зменшити ефекти краю при застосуванні ядра, забезпечує періодичність.

Весь функціонал роботи з моделлю винесений в окремий файл yolov5_tflite.cpp, тому ініціалізація моделі нейронної мережі в основному файлі займає всього кілька строк (лістинг 3.7).

Лістинг 3.7 – Ініціалізація моделі нейронної мережі

```
YOLOV5 model;
model.confThreshold = 0.30;
model.nmsThreshold = 0.40;
model.nthreads = 4;
model.loadModel(model_path);
model.getLabelsName(names_path, labelNames);
```

Реалізація використаних двох методів loadModel та getLabelsName виглядає наступним чином (лістинги 3.8, 3.9).

Лістинг 3.8 – Реалізація методу loadModel

```
void YOLOV5::loadModel(const  std::string path)
{
    _model                                                    =
tflite::FlatBufferModel::BuildFromFile(path.c_str());
    if (!_model)
    {
        std::cout << "\nFailed to load the model.\n"
                  << std::endl;
```

Продовження лістингу 3.8

```
            exit(1);
        }
        else
        {
            std::cout << "\nModel loaded successfully.\n";
        }
        tflite::ops::builtin::BuiltinOpResolver resolver;
        tflite::InterpreterBuilder(*_model,
resolver)(&_interpreter);
        TfLiteStatus status = _interpreter->AllocateTensors();
        if (status != kTfLiteOk)
        {
            std::cout << "\nFailed to allocate the memory for
tensors.\n"
                      << std::endl;
            exit(1);
        }
        else
        {
            std::cout << "\nMemory allocated for tensors.\n";
        }
        // input information
        _input = _interpreter->inputs()[0];
        TfLiteIntArray  *dims  =  _interpreter->tensor(_input)-
>dims;
        _in_height = dims->data[1];
        _in_width = dims->data[2];
        _in_channels = dims->data[3];
        _in_type = _interpreter->tensor(_input)->type;
        _input_8                =                _interpreter-
>typed_tensor<uint8_t>(_input);
        _interpreter->SetNumThreads(nthreads);
    }
```

Лістинг 3.9 – Реалізація методу getLabelsName

```
    void        YOLOV5::getLabelsName(std::string        path,
std::vector<std::string> &labelNames)
    {
        // Open the File
        std::ifstream in(path.c_str());
        // Check if object is valid
        if (!in)
            throw std::runtime_error("Can't open ");
        std::string str;
        // Read the next line from File until it reaches the
end.
        while (std::getline(in, str))
        {
            // Line contains string of length > 0 then save it
in vector
            if (str.size() > 0)
                labelNames.push_back(str);
        }
        // Close The File
        in.close();
    }
```

Після того, як всі підготовчі процеси були зроблені (відкритий ресурс кадрів та ініціалізовані всі необхідні змінні) можна починати роботу з відеопотоком у нескінченному циклі.

Як було описано раніше в схемі, спочатку проходить процеси зчитування нового зображення з потоку та його стабілізація (лістинг 3.10).

Лістинг 3.10 – Зчитування та стабілізація кожного кадру

```
    capture >> frame;
            cv::resize(frame,   frame,   cv::Size(1080,   720),
cv::INTER_CUBIC);
```

Продовження лістингу 3.10

```
        if (frame.empty())
          break;

        stabilizedFrame = stabilizer->doStabilize(frame);
```

Після зчитування кадру іде перевірка чи пуста «зона стеження» (лістинг 3.11).

Лістинг 3.11 – Перевірка чи пуста «зона стеження»

```
if(containingRect.empty())
```

Якщо «зона стеження» пуста, то в роботу включається модель нейронної мережі для визначення об’єктів на кадрі та для зручності виводить всю необхідну інформацію (лістинг 3.12).

Лістинг 3.12 – Пошук об’єктів на кадрі

```
model.run(stabilizedFrame, out_pred);
std::cout << "\nModel run time 'milliseconds': " << duration.count() << "\n"
          << std::endl;
auto boxes = out_pred.boxes;
auto scores = out_pred.scores;
auto labels = out_pred.labels;
std::cout << "\nboxes count: " << boxes.size() << "\n"
                  << std::endl;

          for (int i = 0; i < boxes.size(); i++)
          {
              auto box = boxes[i];
              auto score = scores[i];
              auto label = labels[i];
              std::cout << "Detected: " <<
```

Продовження лістингу 3.12

```
    labelNames[label] << " at " << box << " with score: " << score << std::endl;
                    cv::rectangle(stabilizedFrame, box, cv::Scalar(255, 0, 0), 2);
                    cv::putText(stabilizedFrame, labelNames[label], cv::Point(box.x, box.y), cv::FONT_HERSHEY_COMPLEX, 1.0, cv::Scalar(255, 255, 255), 1, cv::LINE_AA);
                }
    cv::cvtColor(stabilizedFrame, stabilizedFrame, cv::COLOR_BGR2RGB);
                cv::imshow("output", stabilizedFrame);
```

Зауважимо, що вибір об’єкту можливий як в автоматичному режимі (буде обиратись об’єкт з найбільшим показником точності виявлення) так і в ручному, тобто «кліком» по обраному оператором об’єкту стеження (лістинг 3.13).

Лістинг 3.13 – Вибір об’єкту стеження

```
    if(containingRect.empty()){
                    containingRect = findContainingRectangle(cv::Point(mouseX, mouseY), boxes);
                    std::cout << "Mouse Click: (" << mouseX << ", " << mouseY << ")" << std::endl;
                    std::cout << "Selected Bounding Box: " << containingRect << std::endl;
                    if(!containingRect.empty()){
                     //capture >> frame;
                     tracker->init();
                     tracker->setTrackROI(stabilizedFrame, containingRect);
                    }
                }
```

Як можемо бачити, після «кліку» оператором іде перевірка. Якщо оператор обрав об’єкт, то він стає ціллю трекеру, якщо ні, то алгоритм не обере жодного об’єкту на даному кадрі.

Припустимо що оператор або система вже успішно обрали об’єкт. В такому випадку модель не буде надалі використовуватись, а буде працювати механізм трекінгу (лістинг 3.14).

Лістинг 3.14 – Обробка об’єкту трекером та оновлення його положення

```
status = tracker->track(stabilizedFrame);
            if(!status){
                containingRect = cv::Rect();
                mouseX = 0;
                mouseY = 0;
            }else{
                cv::rectangle(stabilizedFrame,      tracker-
>getBbox(), cv::Scalar(255, 0, 0), 2);
            }
            cv::imshow("output", stabilizedFrame);
            if(cv::waitKey(1) == 27){
                containingRect = cv::Rect();
                mouseX = 0;
                mouseY = 0;
            }
```

У разі втрати трекером об’єкту стеження він буде автоматично виключатись та передаватись управління нейронній мережі для визначення нового цільового об’єкту. Таким чином продовжується робота програмної частини до того часу доки її не зупинять вручну.

Після того, як положення об’єкту було оновлене трекером, то програма надсилає команду до плати для корегування положень сервоприводів (лістинг 3.15).

Лістинг 3.15 – Коригування положення сервоприводів

```
double errorX = (double)objectCenter.x - frame_center_x;
double errorY = (double)objectCenter.y - frame_center_y;
if (!arduinoController.updateServos(errorX, errorY)) {
     std::cerr << "Error updating servos. Check Arduino connection." << std::endl;
}
```

Як можна побачити, цей код вираховує різницю між центром кадру (поточним положенням сервоприводів) та цільовим положенням. Відповідно до цієї різниці відбувається корегування положення платформи.

3.3 Реалізація апаратної частини

Як було описано вище, апаратна частина доволі проста та складається з платформи, плати, яка нею керує, камери та двох сервоприводів. Комунікація з програмною складовою здійснюється за допомогою USB.

Найскладнішим у реалізації програмної частини було написання алгоритму, який буде добре співпрацювати з програмною частиною. Проблеми, які перед постали у його реалізації при прямій відправці кожної позиції об’єкту, визначеної трекером:

– переповнення буферу Serial на Arduino;

– ривки сервоприводів, які намагатимуться миттєво досягти цілі, що постійно змінюється;

– нестабільне та неефективне стеження.

Для вирішення цих проблем був прийнятий ряд змін відповідно до інженерських рішень: під час передачі даних позбавляємось від строк та переходимо та бінарний формат. Кожна команда відправляється у вигляді структурованого пакету (рисунок 3.2).

DA 6e 6e 86

Рисунок 3.2 – Приклад пакету, який приймає платформа від програми

Пакет складається з двох функціональних значень які розташовані на 2 та 3 позиціях. Вони представляють собою положення кожного з сервоприводів (значення на позиції 2 – координата X, значення на позиції 3 – координата Y). Четверте значення представляє контрольну суму crc8. Контрольна сума – деяке значення, розраховане на основі набору даних з використанням певного алгоритму, що використовується для перевірки цілісності даних при їх передачі або збереженні [33]. В даному випадку я використовую алгоритм розрахунку контрольної суми CRC-8 (лістинг 3.16).

Лістинг 3.16 – Функція розрахунку CRC-8 зі сторони програмної частини

```
uint8_t ArduinoController::crc8(const uint8_t *data, uint8_t len) {
     uint8_t crc = 0;
    for (uint8_t i = 0; i < len; i++) {
        crc ^= data[i];
        for (uint8_t j = 0; j < 8; j++) {
            if (crc & 0x80) crc = (crc << 1) ^ 0x07;
            else crc <<= 1;
        }
    }
    return crc;
}
```

Ця функція знаходиться як у програмній частині системи так і в апаратній для забезпечення «взаєморозуміння» апаратної та програмної частин.

Перетворення координат з кадру у відповідне положення сервоприводів досягається за допомогою функції перетворення координат, що надаються трекером, до вигляду, який використовуватиме Arduino (лістинг 3.17).

Лістинг 3.17 – Функція перетворення координат від трекеру до положення сервоприводів

```
    bool ArduinoController::updateServos(double errorX, double
errorY) {
        if (!isOpen()) return false;
        auto now = std::chrono::steady_clock::now();
        auto                    elapsed                    =
std::chrono::duration_cast<std::chrono::milliseconds>(now    -
last_send_time);
        if (elapsed.count() >= send_interval_ms) {
            double correctionX = pidX.calculate(errorX);
            double correctionY = pidY.calculate(errorY);
            double targetAngleX_double = 90.0 - correctionX;
            double targetAngleY_double = 90.0 - correctionY;
            targetAngleX_double                            =
std::clamp(targetAngleX_double, 0.0, 180.0);
            targetAngleY_double                            =
std::clamp(targetAngleY_double, 0.0, 180.0);
            uint8_t              finalAngleX               =
static_cast<uint8_t>(round(targetAngleX_double));
            uint8_t              finalAngleY               =
static_cast<uint8_t>(round(targetAngleY_double));
            std::cout << "Error(X,Y): (" << errorX << ", " <<
errorY << ") | "
                      << "Correction(X,Y): (" << correctionX <<
", " << correctionY << ") | "
                      <<     "TargetAngle(X,Y):     ("     <<
(int)finalAngleX << ", " << (int)finalAngleY << ")" << std::endl;
```

Продовження лістингу 3.17

```
        if (sendCommand(finalAngleX, finalAngleY)) {
            last_send_time = now;
            return true;
        } else {
            return false;
        }
    }
    return true;
}
```

На стороні Arduino читаємо Serial порт, шукаємо стартовий байт, читаємо пакет необхідної довжини та перевіряємо контрольну суму. Тільки після цього використовуємо дані.

– дані від програмної частини до апаратної передаються не кожен кадр. Було вирішено відправляти дані з частотою, близької до можливостям Arduino та сервоприводів. Це приблизно 10 Гц (раз в 100 мс). Більш того, дані відправлятимуться лише тоді, коли положення сервоприводів змінилась суттєво (0.5–1 градус) у порівнянні з останнім положенням. Це дозволить зменшити трафік передачі даних;

– була змінена логіка повороту сервоприводів. На стороні Arduino реалізована логіка згладжування. Мається на увазі те, що тепер сервоприводу не будуть намагатись одразу досягти цілі, а будуть іти до неї покроково в циклі до тих пір, доки не буде отримано нової команди або ціль не буде досягнута (лістинг 3.18).

Лістинг 3.18 – Функція забезпечення плавності руху сервоприводів

```
    void moveServosSmoothly() {
        bool moved = false;
        if (currentAngle1 != targetAngle1) {
            int diff1 = targetAngle1 - currentAngle1;
            int step1 = constrain(diff1, -MAX_SERVO_STEP,
MAX_SERVO_STEP);
```

Продовження лістингу 3.18

```
            currentAngle1 += step1;
            servo1.write(currentAngle1);
            moved = true;
        }
        if (currentAngle2 != targetAngle2) {
            int diff2 = targetAngle2 - currentAngle2;
            int step2 = constrain(diff2, -MAX_SERVO_STEP,
MAX_SERVO_STEP);
            currentAngle2 += step2;
            servo2.write(currentAngle2);
            moved = true;
        }
}
```

Це є головним механізмом захисту від артефактів та неправильних кадрів з боку стабілізації і, відповідно, це забезпечить правильну роботу сервоприводів. Артефакт – будь-яка небажана зміна даних, привнесена в цифровий процес залученою технікою чи технологією [34]. Суть проблеми, яку цей підхід дозволить вирішити: коли платформа різко повертається по команді від C++, стабілізатор у C++ може тимчасово «загубитися» або видати артефактний кадр, оскільки попередній кадр і поточний сильно відрізнятимуться через рух самої камери, а не тільки зовнішнього світу. Це може призвести до стрибка обчисленої позиції об'єкта і, як наслідок, до невірної команди для сервоприводів.

Припустимо, що через різкий поворот або артефакт стабілізатор в C++ на одному кадрі помилково обчислив цільову точку дуже далеко (targetAngleX = 150, хоча об'єкт майже по центру). Arduino у своєму loop() бачить велику різницю (150 – currentAngleX), але все одно повертає серво тільки на MAX_STEP (наприклад, 1 градус) у бік 150. До наступної ітерації loop() Arduino (через ~20 мс) C++ програма, швидше за все, вже

обробила наступний кадр, стабілізація «прийшла до тями», і тепер C++ надсилає коректну цільову точку (наприклад, targetAngleX = 95).

Після того, як були впровадженні описані вище зміни, алгоритм роботи на стороні Arduino став мати наступний функціонал:

– впроваджено стартове положення камери;

– покроковий рух сервоприводів під час зміни положення;

– додана затримка для зменшення кількості та ймовірності появи артефактів;

– алгоритмічна плавність рухів сервоприводів.

3.4 Результати та приклади роботи системи

Передбачається, що застосунок працюватиме без інтерфейсу та оператора, але для демонстраційного варіанту були внесені корективи для наочного відображення роботи системи.

Запуск системи виглядатиме однаково як за наявності інтерфейсу так і без нього (рисунок 3.3).

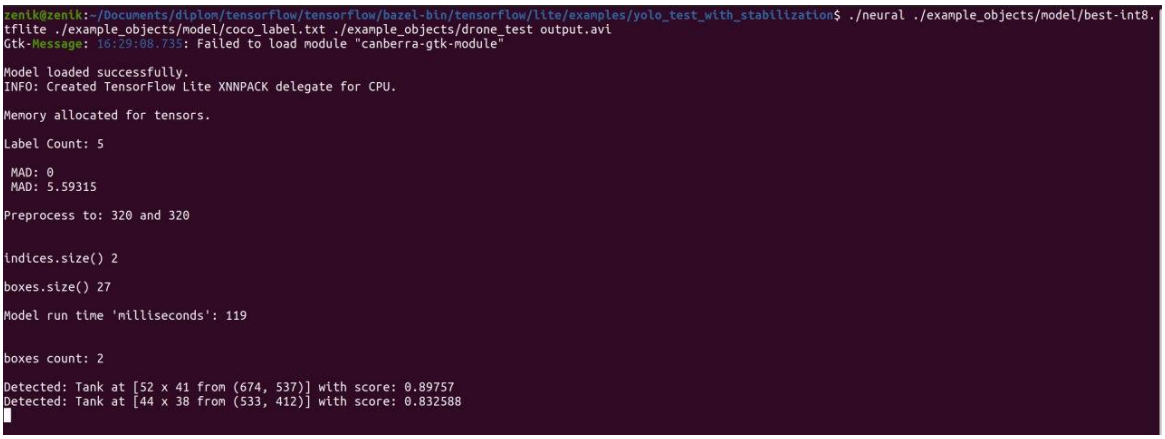

Рисунок 3.3 – Приклад запуску системи

В даному випадку для демонстрації було використано не відеопотік реального часу, а готове відео, для детальної демонстрації тільки

програмної частини. В даному випадку взаємодія з платформою вимкнена. Для того щоб використовувати реальний час необхідно и достатньо замість третього аргументу передати шлях до камери.

3.4.1 Результати та приклади роботи програмної частини системи

Після запуску програми почнеться пошук об’єктів з обраної предметної галузі. Для того щоб змінити предметну галузь необхідно і достатньо передати іншу модель та інший файл з «лейблами». Лейбли – умовні позначення, які використовувались в процесі навчання моделі та які використовуватимуться системою для визначення типу об’єкту. Якщо вивести зображення на екран то побачимо, що система знайшла два об’єкти на першому кадрі та чекає поки система або оператор обере за яким система буде стежити (рисунок 3.4).

Рисунок 3.4 – Приклад визначених об’єктів

Після того, як оператор або система оберуть об’єкт стеження, то буде запущено трекер (рисунок 3.5).

Рисунок 3.5 – Приклад обробки обраного об’єкту трекером

Також слід зазначити, що трекер та стабілізація можуть бути перезапущені не тільки у разі втрати об’єкту, а й у випадку коли програма ідентифікує різку зміну кадру (коефіцієнти MAD для стабілізації та MSE для трекінгу будуть більші за значення 15 та 150 відповідно. Ці значення отримані експериментально після обробки кількох відео різних форматів), що може свідчити про зависання потоку (рисунок 3.6).

```
Mouse Click: (564, 439)
Selected Bounding Box: [44 x 38 from (533, 412)]
 MAD: 5.59396
 MAD: 5.51757
mse: 36.9952
 MAD: 10.1973
mse: 37.6399
 MAD: 5.91052
mse: 67.0459
 MAD: 5.51326
mse: 38.5872
 MAD: 5.87703
mse: 36.9766
 MAD: 6.04127
mse: 37.8926
 MAD: 9.86927
mse: 44.0715
 MAD: 6.00105
mse: 61.7004
 MAD: 5.87906
```

Рисунок 3.6 – Вивід коефіцієнтів MAD та MSE

MAD – це середнє арифметичне абсолютних значень відхилень фактичних значень від прогнозованих. Вимірює середню величину помилки без урахування її напряму [35]. MSE – це середнє значення квадратів різниць між фактичними та прогнозованими значеннями. Цей показник більш чутливий до великих помилок, ніж MAD [36].

Після успішного тестування моделі та трекера на заздалегідь записаних відео було прийнято рішення перейти до випробувань на відеопотоці в реальному часі (рисунок 3.7).

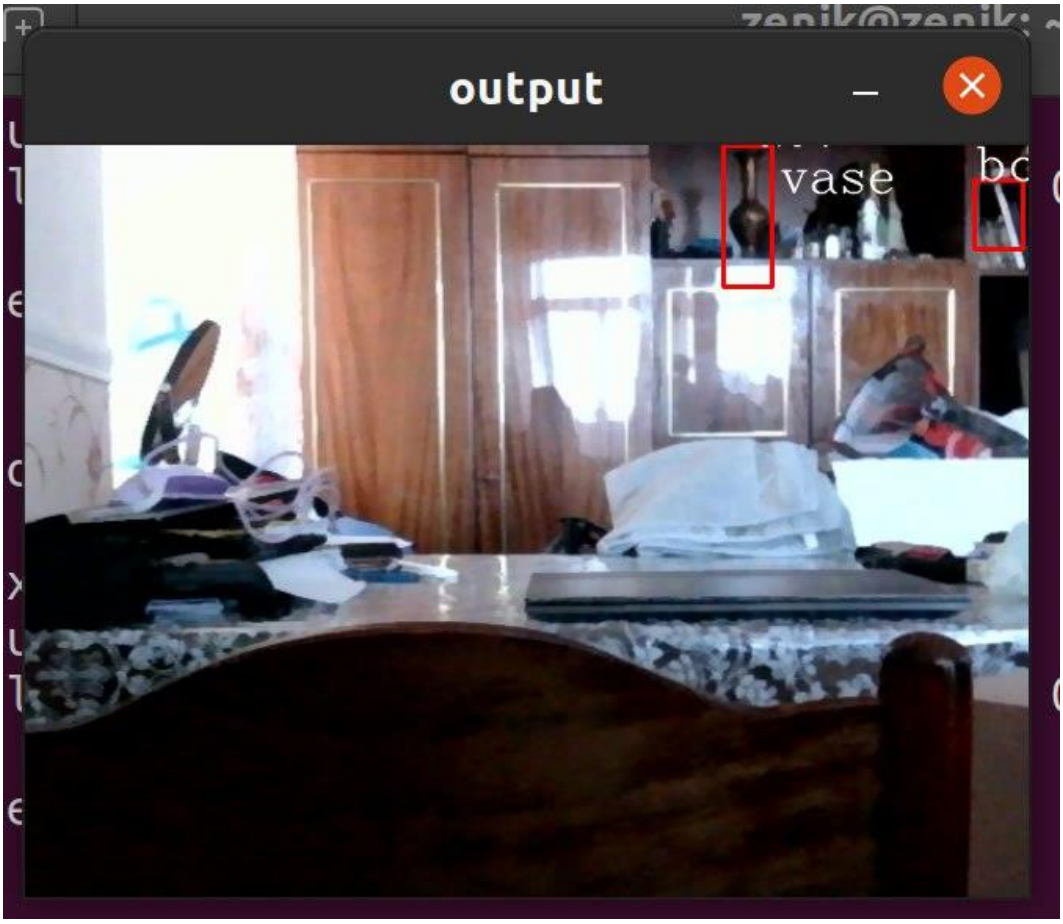

Рисунок 3.7 – Запуск усієї системи в реальному часі. Виявлення об’єктів на кадрі

Після визначення об’єкту на кадрі система перейшла до трекінгу (рисунок 3.8).

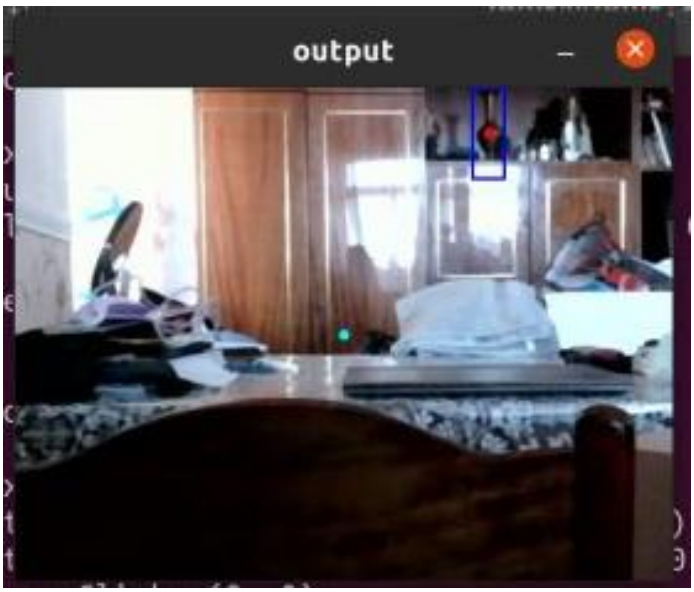

Рисунок 3.8 – Запуск усієї системи в реальному часі. Трекінг об’єкту

Після того, як трекер на об’єкті був ініціалізований, почався процес коригування сервоприводів відповідно до положення об’єкту. Після відповідних операцій з боку програмної складової системи та рухів сервоприводів був отриманий наступний результат (рисунок 3.9).

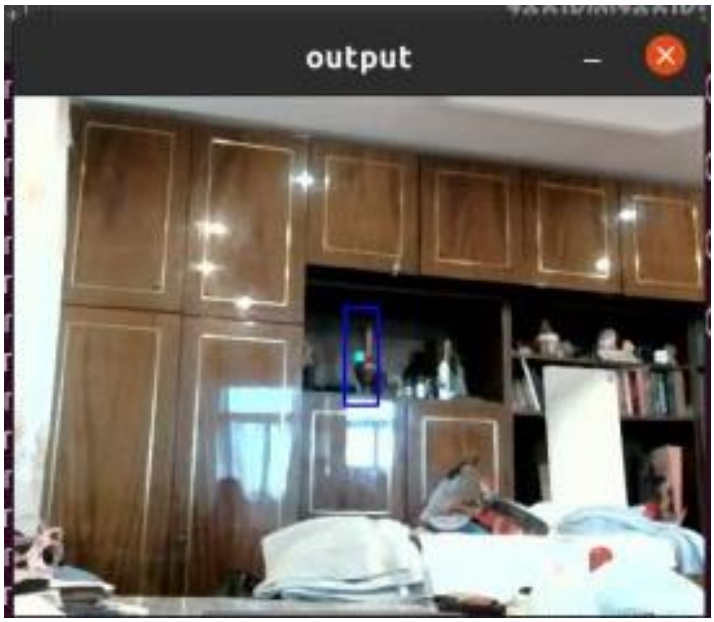

Рисунок 3.9 – Запуск усієї системи в реальному часі. Наведення камери на цільовий об’єкт

У тестах використовувалась модель на базі YOLOv5s та трекер KCF з зазначеними вище в роботі параметрами та завдяки цьому вдалось досягти швидкості роботи моделі 10–15 FPS та швидкості роботи трекеру 30 FPS, що є прийнятними для подібної системи.

3.4.2 Результати та приклади роботи апаратної частини системи

Побудована апаратна частина системи має наступний вигляд (рисунок 3.10).

Рисунок 3.10 – Апаратна частина системи

Після запуску системи відбувається, спершу, її налаштування та коригування стартового положення. Це робиться для того, щоб під час запуску система чітко розуміла початкове положення (рисунок 3.11).

Після того, як буде запущена програмна частина, відбудеться захоплення цільового об’єкту та до платформи почнуть надходити команди на зміну положення, то платформа буде на них реагувати відповідно (рисунок 3.12).

Рисунок 3.11 – Стартове положення платформи

Рисунок 3.12 – Змінене положення платформи відповідно до команд

Можна побачити, що платформа змінила своє положення від початкового. Це означає, що апаратна частина системи отримала команду від програмної частини, обробила її та коректно відреагувала на неї.

3.5 Висновки за розділом

На основі поставленої задачі та обраних технологій була створена та протестована система виявлення та стеження за об’єктами в реальному часі.

Програмна частина була розроблена на C++ в середовищі Linux Ubuntu. Ключовим завданням стала збірка та налаштування необхідних бібліотек, зокрема OpenCV для обробки зображень та розширених функцій трекінгу, та TensorFlow для роботи нейронної мережі, що вимагало використання спеціалізованих інструментів як CMake та Bazel. Логіка програми включає ініціалізацію системи через командний рядок, захоплення та стабілізацію відео, детекцію об'єктів за допомогою моделі YOLOv5, та їх подальше супроводження трекером KCF. Система гнучко перемикається між детекцією та трекінгом, дозволяючи як автоматичний, так і ручний вибір цілі, та надсилає команди керування на апаратну платформу.

Апаратна частина складається з рухомої платформи з камерою, керованої платою Arduino через сервоприводи. Значну увагу було приділено оптимізації взаємодії між програмною та апаратною частинами для усунення проблем зі стабільністю та швидкістю реакції. Це було досягнуто шляхом впровадження бінарного протоколу передачі даних з контролем цілісності (CRC-8), обмеження частоти відправки команд та реалізації алгоритму плавного руху сервоприводів на стороні Arduino, що також мінімізувало вплив артефактів від стабілізації зображення при різких рухах камери.

Тестування продемонструвало ефективну роботу системи. Програмне забезпечення коректно ідентифікує та супроводжує об'єкти на відео, досягаючи прийнятної швидкості обробки. Апаратна платформа точно реагує на команди, забезпечуючи наведення камери на об'єкт стеження. Впроваджені інженерні рішення забезпечили стабільну та злагоджену роботу всіх компонентів автономної системи стеження.

ВИСНОВКИ

На основі глибокого аналізу сучасних підходів та технологій було обґрунтовано та спроектовано відкриту модульну архітектуру системи виявлення та стеження на базі поворотної платформи. Вона базується на взаємодії ключових модулів: розпізнавання, стабілізації зображення, трекінгу та програмно-апаратного керування.

Вибір технологічного стеку був зумовлений прагненням до ефективності, швидкодії та можливості роботи на ресурсно обмежених платформах:

– детекція: згорткова нейронна мережа YOLOv5 у форматі TensorFlow Lite забезпечила оптимальний баланс між точністю та швидкістю розпізнавання;

– трекінг: алгоритм KCF був обраний за його високу швидкодію та незалежність від попереднього навчання;

– обробка: бібліотека OpenCV надала потужний інструментарій для роботи із зображеннями та стабілізації (метод оптичного потоку);

– керування: мова C++ стала основою програмної реалізації завдяки своїй продуктивності, а платформа Arduino – надійним та доступним рішенням для керування фізичною поворотною платформою.

Процес реалізації та тестування підтвердив ефективність обраного підходу. Було розроблено програмне забезпечення на C++ під Linux, що забезпечує повний цикл роботи: від захоплення та стабілізації відео до детекції, трекінгу та відправки команд керування. Особливу увагу було приділено вирішенню технологічних викликів:

Для забезпечення стабільної та швидкої взаємодії між програмною та апаратною частинами було впроваджено бінарний протокол передачі даних з контролем цілісності (CRC-8).

Для плавності руху та мінімізації впливу артефактів було реалізовано алгоритм плавного керування сервоприводами на стороні Arduino.

Система продемонструвала гнучкість у перемиканні між режимами детекції та трекінгу, а також здатність до самовідновлення після втрати об’єкта.

Тестування підтвердило, що розроблена система ефективно ідентифікує та супроводжує об'єкти, а апаратна платформа точно реагує на команди, забезпечуючи наведення камери.

У ході виконання даної роботи було успішно досягнуто ключової мети – розроблено та реалізовано функціональну автономну систему для виявлення та стеження за об'єктами в режимі реального часу. Проведене дослідження підтвердило високу актуальність створення доступних, гнучких та модульних рішень у цій галузі, здатних адаптуватися до широкого кола завдань – від цивільного моніторингу до специфічних військових потреб, мінімізуючи людське втручання та витрати.

Таким чином, створена система є універсальним, недорогим та відкритим до адаптації рішенням. Вона вигідно відрізняється своєю доступністю та гнучкістю порівняно з багатьма комерційними аналогами. Незважаючи на певні апаратні обмеження та потребу в подальшій оптимізації, ця робота закладає міцний фундамент для подальших розробок у сфері автоматизованого відеоспостереження, робототехніки та автономних систем. Вона є не лише функціональним прототипом, а й перспективною платформою для впровадження нових ідей та вирішення прикладних задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bengio Y., Courville A., Goodfellow I. Deep Learning. MIT Press, 2016. 800 с.

2. Учасники проектів Вікімедіа. Стабілізація зображення – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Стабілізація_зображення (дата звернення: 06.04.2025).

3. Цифрова обробка зображень. *Кафедра обчислювальної техніки*. URL: https://ot.vntu.edu.ua/images/Literature/Ochkur/met_coz.pdf (дата звернення: 06.04.2025).

4. Тищенко А. Афінне перетворення. *ВУЕ*. URL: https://vue.gov.ua/Афінне_перетворення (дата звернення: 06.04.2025).

5. Учасники проектів Вікімедіа. Оптичний потік – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Оптичний_потік#:~:text=Опти́чний%20пото́к%20(англ.,руху%20візерунку%20яскравості%20в%20зображенні. (дата звернення: 06.04.2025).

6. Репозитарій Вінницького Національного Технічного Університету. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/29265/9522.pdf (дата звернення: 06.04.2025).

7. Відстеження (комп'ютерна графіка) – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Відстеження_(комп'ютерна_графіка) (дата звернення: 06.04.2025).

8. Згорткові нейронні мережі (частина 1) – IT Master – електроніка та програмування. *Головна – IT Master – електроніка та програмування*. URL: https://itmaster.biz.ua/programming/vision/cnns1.html (дата звернення: 06.04.2025).

9. Hikvision. *Hikvision*. URL: https://hikvision.co.ua/?srsltid=AfmBOooegZUOHHCQ_OoyIKi9-RagD3fcDMeHp4yH8z4D9aD1yRiXkyTP (дата звернення: 06.04.2025).

10. Insta360 Link (CINSTBJ/A). *Insta360 Україна*. URL: https://insta360.com.ua/ru/cameras/link (дата звернення: 06.04.2025).

11. iFlight – Official Store. *iFlight – Official Store*. URL: https://shop.iflight.com/?srsltid=AfmBOoo6u3_7f8bYnNCHCk1XzsA_70aWTYvx7sM7e-6RiOdv3Xv_QWME (дата звернення: 06.04.2025).

12. Класифікація – тлумачення, орфографія, новий правопис онлайн. *Словник – тлумачний словник української мови, орфографічний словник онлайн*. URL: https://slovnyk.ua/index.php?swrd=класифікація (дата звернення: 06.04.2025).

13. Introduction to C++ Programming Language – GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/cpp-programming-intro/ (дата звернення: 06.04.2025).

14. GeeksforGeeks. What is OpenCV Library? – GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/opencv-overview/ (дата звернення: 06.04.2025).

15. Announcing TensorFlow Lite – googblogs.com. *googblogs.com*. URL: https://www.googblogs.com/announcing-tensorflow-lite/ (дата звернення: 06.04.2025).

16. R T. M. The main reason to use C++ over python for your Embedded Systems projects. *Medium*. URL: https://tejasmr.medium.com/the-main-reason-to-use-c-over-python-for-your-embedded-systems-projects-26171a0e56f3 (дата звернення: 06.04.2025).

17. Lynn T. Best Computer Vision Tools: Advice on Best Libraries & More. *Roboflow Blog*. URL: https://blog.roboflow.com/computer-vision-tools (дата звернення: 06.04.2025).

18. Reddit – The heart of the internet. *Reddit – The heart of the internet*. URL: https://www.reddit.com/r/computervision/comments/bibm0o/good_alternatives_to_opencv/?rdt=48820 (дата звернення: 20.04.2025).

19. LiteRT overview | Google AI Edge | Google AI for Developers. *Google AI for Developers*. URL: https://ai.google.dev/edge/litert (дата звернення: 06.04.2025).

20. LiteRT delegate for NPUs | Google AI Edge | Google AI for Developers. *Google AI for Developers*. URL: https://ai.google.dev/edge/litert/android/npu (дата звернення: 06.04.2025).

21. GitHub – ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite. *GitHub*. URL: https://github.com/ultralytics/yolov5 (дата звернення: 06.04.2025).

22. Quantization aware training | TensorFlow Model Optimization. *TensorFlow*. URL: https://www.tensorflow.org/model_optimization/guide/quantization/training (дата звернення: 06.04.2025).

23. High-Speed Tracking with Kernelized Correlation Filters. *arXiv.org*. URL: https://arxiv.org/abs/1404.7584 (дата звернення: 06.04.2025).

24. Arduino Documentation. URL: https://docs.arduino.cc/ (дата звернення: 06.04.2025).

25. PhD I. I. YOLOv4 vs YOLOv5. *Medium*. URL: https://medium.com/deelvin-machine-learning/yolov4-vs-yolov5-db1e0ac7962b (дата звернення: 06.04.2025).

26. GitHub – ultralytics/ultralytics: Ultralytics YOLO11. *GitHub*. URL: https://github.com/ultralytics/ultralytics (дата звернення: 06.04.2025).

27. Ultralytics. YOLOv5 vs YOLOv8: A Detailed Comparison. *Home – Ultralytics YOLO Docs*. URL: https://docs.ultralytics.com/compare/yolov5-vs-yolov8/#strengths_1 (дата звернення: 06.04.2025).

28. Half-precision Inference Doubles On-Device Inference Performance. *The TensorFlow Blog*. URL: https://blog.tensorflow.org/2023/11/half-precision-inference-doubles-on-device-inference-performance.html (дата звернення: 06.04.2025).

29. Bazel. *Bazel*. URL: https://bazel.build/ (дата звернення: 06.04.2025).

30. GitHub – bazelbuild/bazelisk: A user-friendly launcher for Bazel. *GitHub*. URL: https://github.com/bazelbuild/bazelisk (дата звернення: 06.04.2025).

31. Command Line Arguments In CPP – Geekster Article. Geekster Article. URL: https://www.geekster.in/articles/command-line-arguments-in-cpp/ (дата звернення: 06.04.2025).

32. OpenCV: cv: TrackerKCF Class Reference. *OpenCV documentation*. URL: https://docs.opencv.org/3.4/d2/dff/classcv_1_1TrackerKCF.html (дата звернення: 06.04.2025).

33. Контрольна сума – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Контрольна_сума (дата звернення: 06.04.2025).

34. Цифрові артефакти – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Цифрові_артефакти (дата звернення: 06.04.2025).

35. Contributors to Wikimedia projects. Mean absolute error – Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Mean_absolute_error (дата звернення: 20.04.2025).

36. Contributors to Wikimedia projects. Mean squared error – Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Mean_squared_error (дата звернення: 20.04.2025).