Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Автоматики та комп'ютеризованих технологій

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва

(тема)

Виконав:

студент II курсу, групи КТРСм-22-1

Мілько Д. В.

(прізвище, ініціали)

Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології

Тип програми освітньо-професійна

(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютеризовані та робототехнічні системи

(повна назва освітньої програми)

Керівник проф. Цимбал О. М.

(посада, прізвища, ініціали)

Допускається до захисту

Зав. кафедри _____ Невлюдов І. Ш.

(підпис) (прізвище, ініціали)

2023 р

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

11.01.2023  Мілько Д. В.

Харківський національний університет радіоелектроніки

Факультет *Автоматизації та комп'ютерно-інтегрованих технологій*

Кафедра *КІТАР*

Рівень освіти *другий (магістерський)*

Спеціальність *151 Автоматизація та комп'ютерно-інтегровані* технології
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Комп'ютеризовані та робототехнічні системи*
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри_____
(підпис)

«___»_____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Мілько Денис В'ячеславович*
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва*
затверджена наказом університету від *«03» Листопада 2023 р. № 1288 Ст*
2. Термін подання студентом роботи до екзаменаційної комісії *«03» гурдня 2023 р.*
3. Вихідні дані до роботи *Об'єкт дослідження – процес відстеження об'єктів у реальному часі. Предмет дослідження – система відеоспостереження. Функція системи – відстеження та збір інформації про відстежуваний об'єкт. Технічне забезпечення: IBM-сумісний персональний комп'ютер. Середа розробки програмного забезпечення для Python JetBrains PyCharm.*

4. Перелік питань, що потрібно опрацювати в роботі *Вступ. Огляд предметної області. Ознайомлення із поняттям систем відеоспостереження. Ознайомлення із поняттям комп'ютерного зору та нейронних мереж. Аналіз бібліотек OpenCV. Вибір засобів та інструментів для розробки. Навчання моделі нейронної мережі. Розробка програмного забезпечення. Експериментальне дослідження та аналіз результатів роботи. Висновки. Перелік джерел посилання.*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій *Графічний демонстраційний матеріал представлений у презентації: зображення результатів обробки відео; результати навчання моделі нейронної мережі; зображення тестування програмного забезпечення.*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання, аналіз завдання, уточнення*		
	плану роботи	*03.11.2023*	*Виконано*
2	*Огляд сучасного стану проблеми*		
	прийняття рішень при проектуванні		
	технологічних процесів	*06.11.2023*	*Виконано*
3	*Розробка математичного забезпечення*	*12.11.2023*	*Виконано*
4	*Розробка програмного забезпечення задачі*	*18.11.2023*	*Виконано*
5	*Проведення експериментальних*	*20.11.2023*	*Виконано*
6	*Оформлення пояснювальної записки*	*05.12.2023*	*Виконано*
7	*Підготовка презентації*	*06.12.2023*	*Виконано*
8	*Подання закінченої роботи керівникові*	*10.12.2023*	*Виконано*
9	*Подання роботи на рецензування*	*13.12.2023*	*Виконано*
10	*Попередній захист*	*14.01.2024*	*Виконано*
11	*Подання роботи до екзаменаційної комісії*	*17.01.2024*	*Виконано*

Дата видачі завдання *«03» 11 2023* р.

Студент _____ *Мілько Д. В.*
(підпис) (прізвище, ініціали)

Керівник роботи _____ *проф. Цимбал О. М.*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 76 с., 1 табл., 52 рис., 2 дод., 33 джерела.

СИСТЕМА ВІДЕОСПОСТЕРЕЖЕННЯ, СИСТЕМА КОМП’ЮТЕРНОГО ЗОРУ, БІБЛІОТЕКА КОМП'ЮТЕРНОГО ЗОРУ OPENCV, ВІДСТЕЖЕННЯ ОБ’ЄКТІВ У РЕАЛЬНОМУ ЧАСІ, ВИЯВЛЕННЯ ОБ’ЄКТІВ ЗА ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ, УПРАВЛІННЯ СИСТЕМАМИ БПЛА.

Об'єкт дослідження – процес відстеження об’єктів у реальному часі.

Мета роботи – проектування програмної системи відеоспостереження із використанням комп’ютерного зору.

Метод дослідження – проведення експериментів із моделями з використанням систем комп’ютерного зору та машинного навчання.

Предмет дослідження – використання моделей нейронних мереж для виявлення об’єктів.

Використані технічні та програмні засоби – середовище для розробки додатків на Python PyCharm 2021.2, бібліотека комп’ютерного зору OpenCV, бібліотека машинного навчання TensorFlow, бібліотека DroneKit, бібліотека NRF24, сервіс Google Colaboratory Pro.

У ході виконання роботи, було навчено модель нейронної мережі виявлення об’єктів. Розроблено програмне забезпечення системи відеоспостереження та управління польотним контролером серії ArduPilot.

Реалізовано програмний код, який надає користувачу можливість відслідковувати та виявляти об’єкті у зоні огляду камери, розраховувати відстань до об’єктів та координати об’єкту, контролювати польотний контролер БПЛА.

ABSTRACT

Explanatory note: 76 p., 1 table, 52 figures, 2 appendices, 33 sources.

VIDEO SURVEILLANCE SYSTEM, COMPUTER VISION SYSTEM, OPENCV COMPUTER VISION LIBRARY, REAL-TIME OBJECT TRAcing, NEURONIC NETWORK OBJECT DETECTION, UAV SYSTEM MANAGEMENT.

The object of research is the process of tracking objects in real time.

Purpose – to design a video surveillance software system using computer vision.

Research method – conducting experiments with models using computer vision and machine learning systems.

The subject of the study is the use of neural network models for object detection.

Hardware and software used – Python application development environment PyCharm 2021.2, OpenCV computer vision library, TensorFlow machine learning library, DroneKit library, NRF24 library, Google Colaboratory Pro service.

In the course of the work, a neural network model for object detection was trained. The software for the video surveillance system and the ArduPilot flight controller was developed.

The program code was implemented, which allows the user to track and detect objects in the camera's field of view, calculate the distance to objects and object coordinates, and control the UAV flight controller.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

CV – бібліотека комп’ютерного зору OpenCV;

CNN – згорткова нейронна мережа;

HUD – Head-Up Display;

IDE – середовище розробки;

TF – TensorFlow;

TFLite – TensorFlow Lite.

ВСТУП

У наш час, коли технологічний прогрес стрімко змінює наше повсякденне життя та спосіб ведення бізнесу, розробка розподілених систем відеоспостереження набуває особливої важливості. Актуальність розробки сучасної системи відеоспостереження зумовлена важливістю забезпечення безпеки, підвищення ефективності та впровадження інновацій в різних галузях.

Об'єктом дослідження є процес відстеження об'єктів у реальному часі, що є актуальним завданням в умовах сучасного світу, насиченого різноманітними інформаційними потоками та потребами в безпеці.

Мета роботи полягає в проектуванні програмної системи відеоспостереження, яка використовує комп'ютерний зір та інструменти машинного навчання. Використання бібліотеки комп'ютерного зору та нейромереж, зокрема згорткових нейронних мереж (CNN), є ключовим елементом вдосконалення цієї системи. Також необхідно розробити модуль програми для інтеграції системи спостереження із польотними контролерами БПЛА для забезпечення використання системи на великих просторах.

Для досягнення мети роботи, планується розглянути такі питання, як: аналіз предметної області та процесу навчання нейронної мережі на власних даних, вибір методів та засобів для розробки системи, навчання власної моделі нейронної мережі, розробка системи відеоспостереження із трекінгом об’єктів, інтеграція навченої моделі у систему розпізнавання, розробка HUD інтерфейсу, використати метод розрахунку відстані до об'єкту розглянутій у публікації на тему «Дослідження програмного методу визначення відстані до об’єкту за допомогою параметрів камери», розробити додаткові модулі програми для передачі даних через радіоканал та контролю польотного контролеру БПЛА, оформити роботу згідно ДСТУ 3008 – 2015 та методичних вказівок [1-4].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз систем відеонагляду та комп’ютерного зору

Системи відеонагляду є важливою складовою сучасного життя, як у громадських, так і у приватних сферах. Вони використовуються для забезпечення безпеки, відслідковування подій, контролю доступу, а також для управління та вирішення різних завдань в реальному часі. Технології відеонагляду постійно розвиваються, забезпечуючи більшу точність, швидкість і функціональність систем.

Основна мета систем відеонагляду полягає в наданні можливості спостереження за об'єктом або територією. Сучасні системи відеонагляду використовують високоякісні камери, які можуть бути розміщені в різних місцях і мати різний функціонал. Вони можуть фіксувати відео високої роздільної здатності, мають функцію нічного бачення, вбудовані датчики руху та аналізу об'єктів.

Системи відеонагляду можуть бути застосовані у різних сферах життя. Вони широко використовуються у великих містах для відслідковування руху транспорту, контролю над суспільною безпекою, в аеропортах, на вокзалах, на вулицях для вирішення проблем безпеки. Також вони використовуються в приватних будинках та офісах для захисту майна та надання спокою мешканцям [5].

Завдяки розвитку технологій штучного інтелекту, в системах відеонагляду використовуються алгоритми комп'ютерного зору для автоматичного виявлення об'єктів, розпізнавання осіб, аналізу руху та інших параметрів. Це дозволяє системам відеонагляду стати більш ефективними, точними і автономними.

Важливим етапом в розвитку систем відеонагляду є збільшення їхньої інтеграції з іншими системами безпеки, такими як системи виявлення

вторгнень, пожежні системи та системи контролю доступу. Це дозволяє створювати повні інтегровані рішення для забезпечення загальної безпеки об'єктів у реальному часі [6].

У майбутньому системи відеонагляду очікується ще більше інновацій, які підвищать їхню ефективність та широкий спектр застосування.

Системи стеження, зокрема за допомогою безпілотних літальних апаратів (БПЛА), відіграють важливу роль у сучасних технологіях нагляду і розвідки. Розвиваючи високоточні та ефективні методи стеження, ці системи забезпечують широкий спектр застосувань від військових операцій до цивільних завдань у галузі безпеки, агропромисловості, екології та інших сфер.

БПЛА використовуються для військового нагляду, розвідки та ведення спостереження в зоні конфліктів. Вони забезпечують військовим точні та оперативні дані щодо розташування противника, допомагаючи приймати ефективні рішення на полі бою. Також БПЛА використовуються в цивільних сферах, наприклад, для нагляду за природним середовищем, об'єктами критичної інфраструктури, аграрними землями та іншими об'єктами.

Сучасні БПЛА обладнані різноманітними сенсорами, включаючи теплові та інфрачервоні камери, що дозволяє здійснювати стеження навіть в умовах обмеженої видимості або вночі. Деякі можуть бути обладнані радарними системами, що розширює їх здатність до виявлення об'єктів під водою чи в зоні густої рослинності.

БПЛА забезпечують можливість здійснювати аналіз геопростору, визначати координати об'єктів, виконувати картографічні завдання та ведення точної геодезичної інформації. Використання алгоритмів штучного інтелекту дозволяє автоматизувати процес розпізнавання об'єктів на знімках і покращує точність збору інформації [7].

Сучасні БПЛА можуть мати системи самонавігації, що дозволяють їм автономно виконувати завдання стеження та розвідування. Їхні можливості постійно розширюються завдяки новим технологіям та інноваціям в галузі

аерокосмічної техніки та комп'ютерного бачення. Приклади сфер застосування БПЛА на рисунку 1.1.

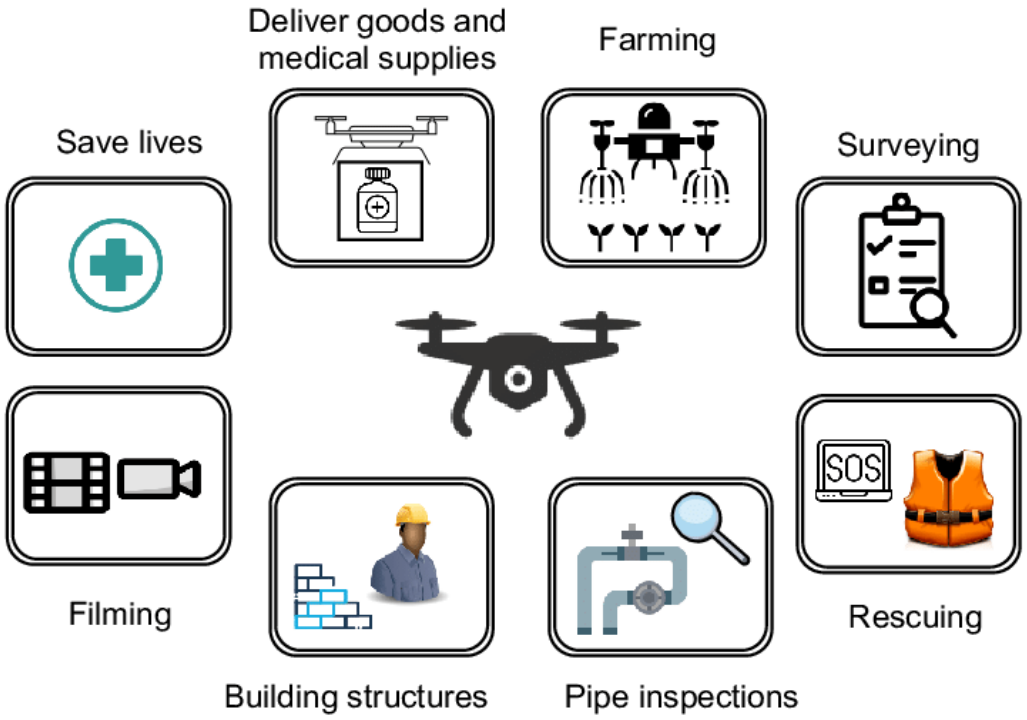

Рисунок 1.1 – Сфери використання БПЛА

Звичайні камери спостереження спроектовані як пасивні пристрої для запису відеоматеріалів для подальшого перегляду. Першим поколінням камер спостереження були системи замкненого телебачення (CCTV), що характеризуються низькою роздільною здатністю та обмеженою функціональністю. Однак зі зростанням впровадження технологій штучного інтелекту і машинного навчання камери відеоспостереження перетворилися на складні пристрої моніторингу для моніторингу в реальному часі, створення зображень високої чіткості, хмарного зберігання і багато чого іншого.

Сучасні камери відеоспостереження здатні не тільки записувати, а й аналізувати знятий матеріал у режимі реального часу. Завдяки вбудованим інтелектуальним алгоритмам вони можуть ідентифікувати, відстежувати й аналізувати об'єкти та поведінку в полі їхнього зору. Цей перехід від пасивного запису до превентивного моніторингу проклав шлях для таких

застосунків, як розпізнавання обличь, виявлення руху та виявлення аномалій, додаючи новий вимір безпеки та спостереження [8].

Використання комп'ютерного бачення в системах відеонагляду представляє собою важливий етап у розвитку технологій нагляду та розпізнавання об'єктів. Ці технології забезпечують автоматизовану обробку та аналіз величезного потоку відеоданих, що надходять від камер та сенсорів.

Однією з ключових переваг є можливість реального чи практично миттєвого виявлення подій чи об'єктів на відеозаписах. Алгоритми комп'ютерного бачення здатні розпізнавати обличчя, визначати рухливі об'єкти, класифікувати сцени та виконувати багато інших завдань.

У відеонагляді застосовуються методи машинного навчання для тренування моделей розпізнавання об'єктів, що дозволяє системі адаптуватися до різноманітних умов та типів об'єктів. Важливим елементом є аналіз поведінки об'єктів, що може слугувати для вчасного виявлення аномалій чи непередбачених ситуацій.

Крім того, системи комп'ютерного бачення можуть взаємодіяти з іншими підсистемами розподіленої системи відеонагляду. Наприклад, автоматичне виявлення певних об'єктів може тригерувати запуск безпілотних літальних апаратів (БПЛА) для подальшого вивчення області або надання додаткової інформації.

Застосування комп'ютерного бачення робить системи відеонагляду більш інтелектуальними та ефективними, сприяючи покращенню безпеки та автоматизації багатьох аспектів життєдіяльності. Такі технології є ключовим елементом розвитку розумних міст, промислових підприємств та сучасних систем безпеки.

Традиційні системи спостереження вимагають людського контролю, який є трудомістким і схильний до помилок. Алгоритми штучного інтелекту і машинного навчання можуть швидко і точно аналізувати і обробляти величезні обсяги даних, виявляючи потенційні загрози або аномалії, які може не помітити людина-спостерігач. Наприклад, камери відеоспостереження на

базі штучного інтелекту можна запрограмувати на розпізнавання таких закономірностей, як залишені без нагляду сумки в громадських місцях, або на ідентифікацію чи відстежування людей за кількома каналами камер. Ці можливості мають неоціненне значення для підвищення громадської безпеки та забезпечення своєчасного реагування на інциденти, пов'язані з безпекою. Ба більше, хоча спостереження за об'єктами та житловими приміщеннями є популярним варіантом використання, потенційні застосування практично безмежні. Наприклад, камери відеоспостереження на базі штучного інтелекту можуть виявляти і контролювати диких тварин, наприклад, види, що перебувають під загрозою зникнення, шляхом захоплення зображень і відеозаписів з високою роздільною здатністю, які можна використовувати для вивчення їхньої поведінки в дикій природі [9].

Підвищення точності комп'ютерного зору камер відеоспостереження зі штучним інтелектом має вирішальне значення з кількох причин. Помилкові спрацьовування, коли ситуація, що не становить загрози, помилково визначається як загроза, можуть призвести до непотрібної паніки і перерозподілу ресурсів. Тим часом хибно негативні результати, коли реальна загроза залишається непоміченою, можуть мати серйозні наслідки. Підвищеної точності можна досягти за рахунок поліпшення алгоритмів AI/ML для обробки зображень, виявлення об'єктів і розпізнавання образів. Це охоплює оптимізацію програмного забезпечення для кращої інтерпретації даних з датчиків камери і виконання точних прогнозів або ідентифікації.

Мікроконтролери є серцем багатьох вбудованих систем, включно з камерами відеоспостереження. Вони відповідають за обробку даних з датчиків камер, запуск алгоритмів виявлення та аналізу об'єктів і прийняття рішень на основі аналізу. У міру того, як алгоритми комп'ютерного зору стають складнішими, потреба в обчислювальній потужності збільшується, і традиційні мікроконтролери можуть ледве впоратися з ними, що призводить до зниження точності та уповільнення часу відгуку.

Але у використанні систем комп’ютерного зору та штучного інтелекту в системах відеонагляду, накладає певні обмеження на їх використання. Розглянемо мінуси подібних систем.

Затримки в обробці даних можуть призвести до неможливості захоплення важливих сцен або спрацьовування сигналів тривоги в режимі реального часу.

Хибно позитивні/хибно негативні результати, без можливості обробки алгоритмів камери можуть або помилково розпізнавати, або не помічати об'єкти, що призводить до необґрунтованих сигналів тривоги.

Продуктивність при слабкому освітленні. Камери відеоспостереження можуть зіткнутися з труднощами під час точного виявлення і відстеження об'єктів в умовах недостатнього освітлення [10].

Недостатні обчислювальні можливості. У камерах відеоспостереження можуть використовуватися мікроконтролери, яким не вистачає достатньої обчислювальної потужності для виконання алгоритмів комп'ютерного зору.

Інтегруючи передові мікроконтролери з можливостями штучного інтелекту і машинного навчання, виробники можуть значно підвищити точність зображення своїх камер відеоспостереження.

1.2 Комп’ютерний зір та нейронні мережі

Комп'ютерний зір – це різновид машинного навчання, який дає змогу комп'ютерам досягати високого рівня розуміння на основі відео та цифрових зображень.

Машинне навчання – це розробка алгоритмів і пов'язаних із ними систем, які можуть вивчати стратегії поведінки в конкретних середовищах за допомогою інструкцій і наборів навчальних даних.

Будучи різновидом штучного інтелекту, машинне навчання значною мірою уникає деяких з найбільших і більш філософських проблем ШІ, приділяючи особливу увагу підходам до навчання і навчання, які можуть

створювати ефективні машини для будь-якого конкретного середовища. Ця дисципліна фокусується на статистичних моделях, алгоритмах і методах навчання, які можуть формувати машини в таких різноманітних галузях, як виробництво, роздрібна торгівля, логістика ланцюжків поставок, виробництво харчових продуктів і будівництво. Приклади сфер використання комп'ютерного зору представлено на рисунку 1.2.

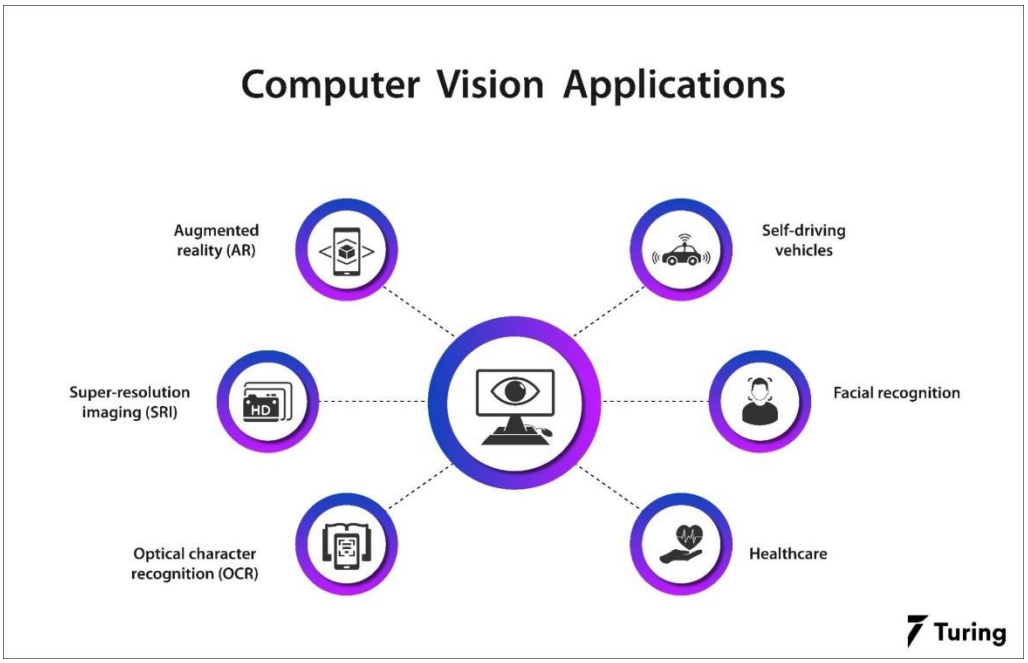

Рисунок 1.2 – Сфери використання комп'ютерного зору

У деяких підходах до машинного навчання особлива увага приділяється алгоритмам навчання розпізнаванню закономірностей у даних для обґрунтування стратегічних дій в аналогічних середовищах.

Контрольоване навчання: у моделях керованого навчання фахівці з даних надають навчальні набори даних для систем машинного навчання з каталогом вхідних даних і пов'язаних з ними бажаних результатів. За такого підходу система машинного навчання може зрозуміти очікувані результати певного набору дій і побудувати оптимальні стратегії для отримання цих результатів.

Навчання без нагляду: як легко зрозуміти з назви, підходи неконтрольованого навчання використовують неструктуровані набори даних

без пов'язаних з ними ідеальних результатів. Тоді система машинного навчання повинна зрозуміти закономірності в наборах даних, щоб розробити стратегії поведінки.

Навчання з підкріпленням: навчання з підкріпленням, яке зазвичай використовується для навчання незалежних машинних агентів у певній системі, використовує моделі кумулятивної винагороди для навчання агентів, як діяти в різних системах. Це застосування ML використовується в багатьох галузях, але значні дослідження в цій сфері були проведені в багатокористувацьких онлайн-іграх.

Глибоке навчання та нейронні мережі. Традиційно в системах машинного навчання і штучного інтелекту використовувалися лінійні або ітеративні підходи до машинного навчання. Починаючи з 1980-х років дослідники розробили "нейронні мережі" мозку, використовуючи структури вузлів і кластерів та зважені стратегії прийняття рішень. Таким чином, системи машинного навчання можуть розбивати складні проблеми на простіші, а результати простіших проблем можуть об'єднуватися в більш комплексне рішення для більших.

Глибоке навчання зробило ще один крок уперед, створивши багаторівневі нейронні мережі, в яких рівні мереж, що базуються на рішеннях, могли б, по суті, створити механізм вирішення проблем, що виникає. Наприклад, мозок із глибоким навчанням може мати шари, де прості підходи до розпізнавання образів можуть об'єднуватися для розв'язання складних завдань, таких як розпізнавання обличь на зображеннях.

У всіх цих та інших підходах до машинного навчання наголос завжди робиться на те, як навчати системи машинного навчання, моделювати середовище навчання для машинного навчання і використовувати машинне навчання для створення комплексних систем штучного інтелекту та автономних систем.

Під час розроблення штучного інтелекту або машинного навчання фахівцям із даних часто буває корисно обмежити застосовність цих систем.

Наприклад, замість того, щоб очікувати, що система навчатиметься і працюватиме в різних середовищах, науковці, які працюють із даними, можуть замість цього просто розробити машинне навчання для управління конкретним ланцюжком поставок або виробничим процесом [11].

Однак кінцева мета штучного інтелекту для багатьох дослідників – підвищити корисність цих систем у кількох сферах. Навіть на зорі штучного інтелекту вчені уявляли собі універсальний або загальний штучний інтелект, який міг би керувати всіма видами систем. Після цього в 1970-х і на початку 1980-х років стався поштовх до досліджень штучного інтелекту з метою знайти способи розвитку штучного інтелекту в таких галузях, як обробка зображень, розпізнавання мови та робототехніка.

Проблема штучного інтелекту в таких дисциплінах, як робототехніка, полягає в тому, що цим системам необхідно приймати, інтерпретувати і реагувати на візуальну інформацію. Таким чином, досягнення в таких галузях, як передові візуальні та акустичні датчики, системи навігації по навколишньому середовищу і можливості мобільності, прагнули йти в ногу з часом.

У міру відродження штучного інтелекту і машинного навчання останніми роками завдяки високопродуктивним обчисленням, великим даним і хмарним системам зростає і попит на обладнання, яке може підтримувати збір візуальних даних для додатків машинного навчання.

Комп'ютерний зір – це застосунок машинного навчання і штучного інтелекту, який витягує інформацію з цифрових зображень і відео та приймає осмислені рішення на основі цієї інформації.

Як і більшість систем машинного навчання, комп'ютерний зір вимагає значних обсягів даних для навчання алгоритмів інтерпретації цих даних.

Комп'ютерний зір зазвичай використовує дві різні технології.

Глибоке навчання. Як згадувалося раніше, глибоке навчання може сприяти вирішенню складних проблем. Що ще більш важливо, глибоке навчання з використанням нейронних мереж може, по суті, навчити "мізки"

машин сприймати візуальні дані та зберігати знання про закономірності, стратегії та зміни змінних навколишнього середовища з плином часу.

Згорткові нейронні мережі. CNN приймають візуальну інформацію, таку як зображення, і розбивають її на пікселі, використовуючи "згортки" (операцію створення математичної функції з двох інших функцій) для прогнозування цих даних.

Комп'ютерний зір – це різновид машинного навчання. Після того, як у середині 1980-х – середині 1990-х років інтерес до досліджень у галузі штучного інтелекту і машинного навчання, більша частина розробок у цій галузі була фрагментована на такі підобласті, як обробка природної мови, розпізнавання зображень і робототехніка.

Більше того, комп'ютерний зір можна визначити як різновид глибокого навчання. Однак замість опрацювання змодельованих даних або статистики комп'ютерний зір аналізує та інтерпретує візуальну інформацію.

Примітно, що комп'ютерний зір не потрібен у багатьох додатках машинного навчання. Система машинного навчання, що керує виробничою лінією або моделює цифрові двійники для морських танкерів, не має великого застосування можливостей комп'ютерного зору. Інформація, яку ці системи повинні вивчати і використовувати, доступна в числовому поданні.

З іншого боку, системам комп'ютерного зору для навчання і функціонування потрібна візуальна інформація. Системи комп'ютерного зору будуть поєднувати в собі підходи машинного навчання, що обговорювалися раніше, з обладнанням, таким як камери, оптичні датчики тощо. Цей підхід дійсно має деякі обмеження, зокрема проблеми з обладнанням і способи перетворення зображень у корисні структури даних для машинного навчання.

1.3 Інструменти роботи із комп’ютерним зором

У сучасному комп'ютерному зорі існують різноманітні інструменти та техніки, які дозволяють виконувати завдання обробки та розпізнавання

зображень. До таких інструментів відносяться алгоритми виокремлення особливостей, які визначають ключові точки та структури на зображенні, сприяючи подальшому аналізу. Нейронні мережі стали невід'ємною частиною сучасних систем комп'ютерного зору, здатних до глибокого навчання та розпізнавання складних патернів.

Методи фільтрації та відтворення зображень використовуються для покращення якості зображень та виділення конкретних деталей. Різні техніки фільтрації можуть застосовуватися для зменшення шуму, виокремлення кольорових або текстурних особливостей, що сприяє подальшій обробці та аналізу. Зокрема, статистичні та математичні моделі використовуються для аналізу відомостей та врахування взаємозв'язків між різними елементами зображення.

Однією з передових бібліотек у цьому контексті є OpenCV, яка поєднує у собі різноманітні методи та інструменти для ефективної обробки та аналізу зображень. Її відкритий характер дозволяє використовувати її в широкому спектрі проектів, включаючи системи відеонагляду та комп'ютерного зору [12].

OpenCV, або бібліотека комп'ютерного зору з відкритим вихідним кодом, є одним з найбільш визнаних і використовуваних інструментів у світі комп'ютерного зору. Її шлях, відзначений інноваціями та широким розповсюдженням, дає уявлення про розвиток самої галузі комп'ютерного зору.

Заснована в 1999 році компанією Intel, OpenCV замислювалася як спосіб підвищити доступність інструментів комп'ютерного зору. Intel визнала, що, зробивши такі інструменти доступними і з відкритим вихідним кодом, з'явиться більше додатків та інновацій. Гері Бредскі, одна з ключових фігур OpenCV, зіграв вирішальну роль у його ранньому розробленні [13].

Його перший загальнодоступний реліз відбувся 2000 року, і відтоді OpenCV регулярно отримував оновлення, кожне з яких покращувало його можливості та розширювало сферу його застосування. До 2012 року кількість

завантажень OpenCV перевищила 2 мільйони, що відображає його зростаюче значення. У 2012 році бібліотека перейшла під егіду OpenCV.org, забезпечивши її безперервність як проект із відкритим вихідним кодом. Бібліотека також отримала підтримку з боку таких компаній, як Google, Microsoft і Intel, що ще більше зміцнило його місце в галузі.

OpenCV надає широкі можливості для обробки зображень, включно з фільтрацією, перетворенням колірного простору та обчисленням гістограм. Крім того, він надає функції виявлення і зіставлення особливостей, які є основою для таких завдань, як зшивання панорам або розпізнавання об'єктів.

Детектори об'єктів в OpenCV включають каскади Хаара і моделі глибокого навчання, які широко використовуються для таких завдань, як розпізнавання обличь та інших об'єктів. Бібліотека також надає інструменти для роботи з геометрією зображень, включно з калібруванням камери і 3D-реконструкцією.

Що стосується машинного навчання, OpenCV вміє використовувати модулі для завдань, пов'язаних із комп'ютерним зором, таких як алгоритм k-найближчих сусідів (k-NN) і машини опорних векторів (SVM).

Бібліотека також добре підтримує обробку відео, включно з аналізом руху і відніманням фону, а також надає інструменти для створення графічних користувацьких інтерфейсів.

Однією з сильних сторін OpenCV є її інтеграція з іншими популярними бібліотеками та інструментами, такими як NumPy і TensorFlow.

Найбільш вражаючим аспектом OpenCV є його універсальність. Вона задовольняє потреби як новачків, так і професіоналів, а відкритий характер бібліотеки дає змогу їй постійно розширюватися й адаптуватися до потреб та інновацій глобальної технологічної спільноти.

Сучасний пейзаж комп'ютерного зору у XXI столітті переживає трансформаційні зміни, спричинені, насамперед, доступністю даних, обчислювальною потужністю та алгоритмічними інноваціями. Ключове місце

в цьому процесі посідає стрімкий розвиток глибокого навчання та роль, яку відіграють сучасні інструменти та структури.

Зростання глибокого навчання та його вплив на традиційний комп'ютерний зір: Історично завдання комп'ютерного зору базувалися на ручному проєктуванні функцій. Такі методи, як SIFT, HOG і виявлення меж, відігравали важливу роль у вилученні інформації із зображень. Однак ці методи мали свої обмеження, особливо в складних реальних сценаріях і великих наборах даних [14].

Введення глибокого навчання: Глибоке навчання радикально змінило парадигму комп'ютерного зору, оскільки воно може автоматично вивчати ієрархію функцій на основі необроблених даних. Зокрема, згорткові нейронні мережі (CNN), як підмножина архітектури глибокого навчання, стали ключовими для досягнення сучасних результатів у таких завданнях, як класифікація зображень, сегментація та виявлення об'єктів. У результаті традиційні методи комп'ютерного зору, хоча й не втратили своєї актуальності, почали використовуватися в поєднанні з методами глибокого навчання та іноді заміщатися ними.

1.4 Інструменти навчання нейронних мереж

TensorFlow – це наскрізна платформа з відкритим вихідним кодом для створення додатків машинного навчання. Це символьна математична бібліотека, яка використовує потік даних і диференційоване програмування для виконання різних завдань, орієнтованих на навчання і висновок глибоких нейронних мереж. Вона дозволяє розробникам створювати додатки для машинного навчання, використовуючи різні інструменти, бібліотеки та ресурси спільноти.

Наразі найвідомішою бібліотекою глибокого навчання у світі є TensorFlow від Google. Google використовує машинне навчання у всіх своїх

продуктах для покращення пошукової системи, перекладу, підписів до зображень або рекомендацій. Структура TensorFlow вказана на рисунку 1.3.

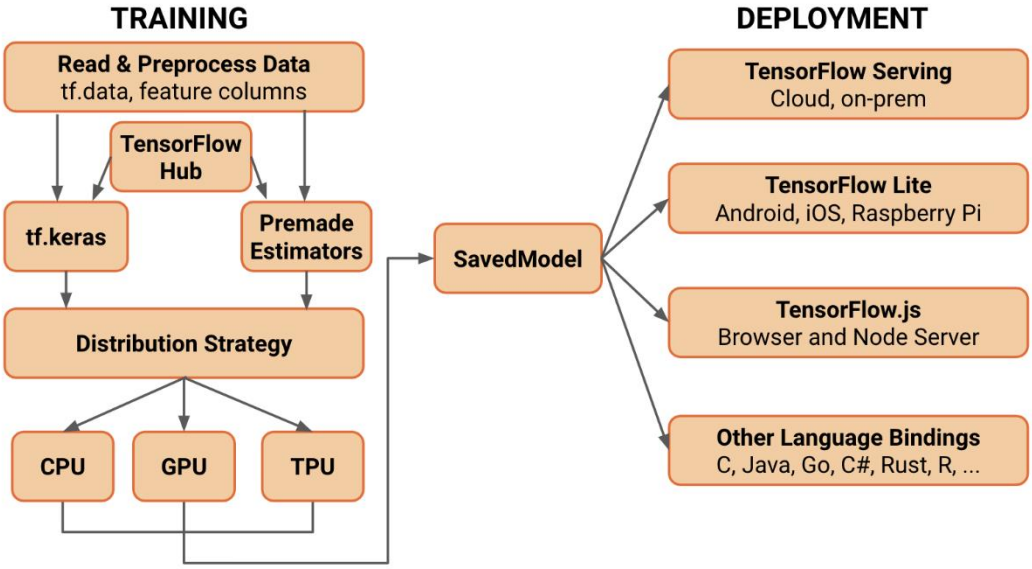

Рисунок 1.3 – Структура TensorFlow

Наведемо конкретний приклад: користувачі Google можуть отримати швидший і точніший пошук завдяки штучному інтелекту. Якщо користувач вводить ключове слово в пошуковий рядок, Google надає рекомендацію щодо того, яким може бути наступне слово.

Google хоче використовувати машинне навчання, щоб скористатися перевагами своїх величезних наборів даних і надати користувачам найкращий досвід. Три різні групи використовують машинне навчання: Дослідники, фахівці з даних, програмісти.

Всі вони можуть використовувати один і той самий набір інструментів для співпраці один з одним і підвищення своєї ефективності.

Google має не просто будь-які дані, а найпотужніший у світі комп'ютер, тому Tensor Flow було створено з урахуванням його масштабу. TensorFlow – це бібліотека, розроблена командою Google Brain Team для прискорення машинного навчання та досліджень глибоких нейронних мереж.

Її було створено для роботи на декількох процесорах або графічних процесорах і навіть мобільних операційних системах, і вона має кілька обгорток на декількох мовах, таких як Python, C++ або Java [15].

Кілька років тому глибинне навчання почало випереджати всі інші алгоритми машинного навчання на великих обсягах даних. Google побачив, що може використовувати ці глибокі нейронні мережі для покращення своїх сервісів: Gmail, Photo, пошукової системи Google.

Вони створили фреймворк під назвою Tensorflow, який дозволяє дослідникам і розробникам спільно працювати над моделлю ШІ. Після розробки та масштабування він дозволяє багатьом людям використовувати його.

Вперше він був оприлюднений наприкінці 2015 року, а перша стабільна версія з'явилася у 2017 році. Він має відкритий вихідний код за ліцензією Apache Open Source. Ви можете використовувати його, модифікувати і розповсюджувати модифіковану версію за певну плату, не сплачуючи нічого Google.

TensorFlow дозволяє створювати графіки потоків даних і структури для визначення того, як дані рухаються по графіку, приймаючи вхідні дані у вигляді багатовимірного масиву під назвою Tensor. Це дозволяє вам побудувати блок-схему операцій, які можна виконати над цими входами, які надходять на один кінець і надходять на інший кінець у вигляді вихідних даних.

Архітектура тензорного потоку складається з трьох частин: попередня обробка даних, побудова моделі, навчання та оцінка моделі

Він називається Tensorflow, тому що приймає вхідні дані у вигляді багатовимірного масиву, також відомого як тензори. Ви можете побудувати своєрідну блок-схему операцій (так званий граф), які ви хочете виконати над вхідними даними. Вхідні дані вводяться з одного кінця, потім вони проходять через цю систему множинних операцій і виходять з іншого кінця як вихідні дані.

Ось чому він називається TensorFlow, тому що тензор входить в нього, проходить через список операцій, а потім виходить з іншого боку.

Назва Tensorflow безпосередньо походить від його основного фреймворку: Тензор. У Tensorflow всі обчислення пов'язані з тензорами. Тензор – це вектор або матриця n-вимірності, яка представляє всі типи даних. Всі значення в тензорі мають ідентичний тип даних з відомою (або частково відомою) формою. Форма даних – це розмірність матриці або масиву.

Тензор може бути отриманий з вхідних даних або в результаті обчислень. У TensorFlow всі операції проводяться всередині графа. Граф – це набір обчислень, які виконуються послідовно. Кожна операція називається операційним вузлом, і вони з'єднані один з одним [16].

Граф окреслює операції та зв'язки між вузлами. Однак, він не відображає значення. Ребра вузлів є тензором, тобто способом заповнення операції даними.

TensorFlow використовує фреймворк графів. Граф збирає та описує всі обчислення серії, виконані під час навчання. Граф має багато переваг.

Він може працювати на декількох процесорах або графічних процесорах і навіть на мобільних операційних системах. Портативність графіка дозволяє зберігати обчислення для негайного або пізнішого використання. Граф може бути збережений для виконання в майбутньому.

Всі обчислення в графі виконуються шляхом з'єднання тензорів між собою. Тензор має вершину і ребро. Вузол виконує математичну операцію і видає результат у кінцевій точці. Ребра пояснюють вхідні/вихідні зв'язки між вузлами.

1.5 Обґрунтування мети

Для досягнення поставленої мети розробки системи відеонагляду, ключовим етапом є проектування алгоритму та розробка програмного забезпечення для відстеження об'єктів. Цей алгоритм буде базуватися на

використанні нейронної мережі, що дозволить системі ефективно впоратися із завданням відстеження об'єктів на відео.

Окрім функціоналу відстеження, розробляється програмний код для передачі даних через радіомодуль. Це необхідно для забезпечення зв'язку та обміну інформацією між системою відеонагляду та іншими пристроями чи системами у реальному часі.

Щоб реалізувати інтеграцію системи стеження на розвідувальні апарати, додатково потрібно впровадити методи роботи та контролю польотних контролерів. Це забезпечить ефективне управління рухом БПЛА та координування його дій з урахуванням інформації, отриманої від системи відеонагляду.

Враховуючи вимоги до системи, її оптимізація для роботи на мікрокомп'ютерах типу Raspberry PI стає важливою задачею. Такий підхід дозволяє використовувати систему в умовах обмежених обчислювальних ресурсів, що актуально для малих БПЛА.

З метою забезпечення ефективності передачі відеоінформації, важливо розділити модулі передачі даних, передбачення через нейронну мережу та управління польотним контролером на окремі потоки процесору. Це сприяє оптимальному використанню ресурсів та підвищує загальну продуктивність системи.

Розробка даного програмного забезпечення надасть можливість створювати розподілені блоки відеонагляду з використанням нейронних мереж. Можливість встановлювати систему на безпілотні літальні апарати додає гнучкості та розширює область використання даного програмного забезпечення.

Системи, які використовуватимуть це програмне забезпечення, стануть ідеальним засобом для проведення спостережень не лише за великими територіями підприємств, але й за аграрними угіддями та великими відкритими складами, надаючи комплексний та ефективний підхід до систем відеонагляду.

1.6 Постановка задачі

Задачею роботи є створення програмного забезпечення для системи відеонагляду, яке використовує нейронну мережу для виявлення об'єктів у відеопотоці. Програма призначена для обробки вхідних даних, що представляють собою відеопотік з камери стеження, підключеної до пристрою. Роздільна здатність вхідного зображення повинна відповідати якості 240p та мати формат MP4.

При отриманні вхідних даних програма виводить на зображення HUD, який знаходиться поверх зображення. Цей HUD містить інформацію про клавіши, які використовуються для керування програмним забезпеченням. Також, при русі курсору миші, програма відображає область зображення, яку можна відстежувати.

При натисканні лівої клавіші миші на обраній області, об'єкт починає відстежуватися системою. Використання нейронної мережі для виявлення об'єктів може бути увімкнене та вимкнене за допомогою відповідних клавіш.

Додатково в програмі реалізований цифровий зум, який діє відповідно до наявності відстежуваного об'єкту. При відсутності об’єкта відстеження, зум збільшує центральну частину зображення. Під час відстеження об'єкту, зум збільшує область, де розташований відстежуваний об'єкт.

Для забезпечення стабільної передачі відеоданих без втрати частоти кадрів та для використання нейронної мережі, необхідно реалізувати окремий асинхронний потік. Цей потік буде відповідальний за отримання, обробку та передачу результатів обробки зображення. Такий підхід забезпечить стабільність та ефективність функціонування системи.

Аналогічно, для методів контролю польотного контролера слід реалізувати асинхронний механізм. При відстеженні об'єкта за допомогою клавіш керування можна передавати інформацію про об'єкт до модулю розрахунку положення об'єкта у системі GPS. Після отримання цих даних,

окремий процес контролю дрону має коригувати політ БПЛА до зазначеного об'єкта. Також необхідно реалізувати можливість зупинення процесу польоту.

Для розрахунку положення об'єкта у системі GPS система комп'ютерного зору повинна визначати відстань до об'єкта на основі отриманих зображенням даних.

Для обміну даними між різними компонентами системи, включаючи контролери польоту, слід реалізувати окремий процес прийняття та відправки даних через радіомодуль. Це забезпечить ефективний обмін інформацією та взаємодію між всіма частинами системи.

1.7 Висновки до першого розділу

Проведено аналіз систем відеоспостереження, розглянуто основні аспекти побудування сучасної системи відеоспостереження. Також проведено аналіз понять комп’ютерного зору та машинного навчання. Розглянуто інструменти роботи із комп’ютерним зором, а саме бібліотеку OpenCV.

Проведено аналіз інструменту TensorFlow. Це впливова відкрита платформа машинного навчання, розроблена Google, яка використовує символьну математичну бібліотеку для глибокого навчання. Вона стала ключовим інструментом у всіх сферах Google.

Розробка системи відеонагляду передбачає проектування алгоритму відстеження об'єктів з використанням нейронної мережі та програмного забезпечення для передачі даних через радіомодуль. Інтеграція на розвідувальні апарати вимагає введення методів контролю польотного контролера. Розділення модулів на окремі потоки процесору покращує ефективність передачі відеоданих, а використання нейронних мереж розширює область застосування на безпілотні літальні апарати та інші об'єкти спостереження, забезпечуючи комплексний підхід до відеонагляду за різноманітними територіями та об'єктами.

2 ВИБІР МЕТОДІВ ТА ЗАСОБІВ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Вибір мови програмування та сторонніх бібліотек

Відповідно до поставлених вимог вибір мови програмування для реалізації потрібного функціоналу повинен бути зорієнтований на кросплатформеність. Серед варіантів, які варто розглянути, виділяються C++, Java та Python.

Спробуємо детальніше розглянути мову C++. Це загальнопризначена мова, яка була розроблена Б’ярном Страуструпом на початку 90х років як розширення мови С. Зараз вона широко використовується у різних системах завдяки таким перевагам, як:

– висока швидкодія роботи, практично на рівні мови С;

– ефективне використання процесорної потужності та пам'яті;

– можливість працювати на низькому рівні, напряму з пам'яттю, адресами та портами.

Серед недоліків можна відзначити:

– обмежена підтримка модульності, що призводить до уповільнення компіляції при підключенні сторонніх модулів;

– складність вивчення та виконання компіляції.

Java є більш сучасною мовою програмування порівняно із C++. Вона була розроблена компанією "Sun Microsystems" у 1995 році та з 2009 року перейшла під контроль "Oracle". Java є об'єктно-орієнтованою мовою програмування, і хоча вона має багато спільного з мовами С та C++, розробники успішно виправили ряд серйозних проблем. Наприклад, вони вирішили проблему помилок при написанні коду програмістом, переклавши основні операції на віртуальну машину.

Java має свої переваги, такі як:

– спрощений синтаксис мови C++, що полегшує читання і входження початківцям;

– автоматичне управління пам'яттю, що дозволяє ефективно використовувати процесорні ресурси;

– широкий вибір сторонніх бібліотек.

Проте є й недоліки:

– нижча швидкість в порівнянні з C та C++;

– багатослівність синтаксису, яка може спричинити накопичення коду та ускладнення його розуміння розробниками.

Python – це високорівнева, інтерпретована мова програмування, яка здобула широку популярність завдяки своїй простоті та високій продуктивності. Створена Гвідо ван Россумом у 1990 році, вона розроблялася як мова для написання чистого та легко зрозумілого коду.

Однією з ключових особливостей Python є його чистий синтаксис, який заснований на використанні відступів для визначення блоків коду. Це робить код на Python легким для читання та розуміння, сприяючи прискоренню процесу розробки.

Python також славиться своєю портативністю, що означає, що програми, написані на цій мові, можуть працювати на різних платформах без значних змін. Це робить його ідеальним вибором для розробників, які хочуть створювати кросплатформенні застосунки.

Багато років активного розвитку та підтримки спільноти призвели до створення величезної кількості сторонніх бібліотек та модулів для Python. Це дозволяє розробникам швидко та ефективно вирішувати різноманітні задачі без необхідності писати все з нуля [17].

Однак Python не є панацеєю, і він має свої обмеження. Зокрема, йому не властива висока швидкість порівняно із мовами, такими як C++, і відсутня статична типізація може призводити до деяких труднощів у великих проектах.

Вибір мови програмування – це завдання, яке вимагає врахування великої кількості факторів. У випадку Python його популярність, простота та

розширюваність роблять його привабливим вибором для багатьох розробників.

Python підтримує бібліотеку комп’ютерного зору OpenCV. Це спрощує реалізацію поставлених задач.

Для реалізації процесу контролю польотного контролеру, обрано бібліотеку DroneKit. Це потужна бібліотека та набір інструментів для розробки програмного забезпечення для безпілотних літальних апаратів (БПЛА) або дронів. Розроблена компанією 3D Robotics, ця бібліотека спрощує взаємодію з дронами і надає розробникам потужні інструменти для створення різноманітних додатків та функціоналу.

Однією з ключових особливостей DroneKit є його високий рівень абстракції, що дозволяє розробникам працювати на вищому рівні абстракції, не заглиблюючись у технічні деталі контролю апарату. Це робить процес розробки додатків для дронів більш доступним та зрозумілим для широкого кола розробників, навіть тих, хто не має глибоких знань у сфері авіації [18].

Бібліотека підтримує різні типи дронів, включаючи ті, що використовують платформи на базі PX4 та ArduPilot. Це робить DroneKit універсальним інструментом для взаємодії з різноманітними моделями дронів, дозволяючи розробникам створювати універсальні додатки.

Однією з сильних сторін DroneKit є його можливості взаємодії з GPS-даними, датчиками, обробкою відео та іншими аспектами, що визначають функціональність дрона. Це дозволяє створювати розширені програми для автопілотів дронів, такі як автоматичне планування маршрутів, слідування за об'єктами або виконання завдань зйомки.

Окрім того, DroneKit підтримує віддалене керування дронами через мережу Інтернет, що дозволяє віддаленим операторам або системам моніторингу взаємодіяти з дронами на великій відстані. Це робить бібліотеку ідеальним інструментом для використання в різних сценаріях, включаючи розвідку, агрокультуру, мапування та багато інших галузей.

В цілому DroneKit визначається своєю потужністю та простотою використання, роблячи його важливим ресурсом для розробників, які мають інтерес або завдання в сфері безпілотних літальних апаратів.

Для передачі даних одним із популярних модулів є NRF24. Цей модуль відносно компактний та енергоефективний, що робить його ідеальним вибором для вбудованих систем.

NRF24 працює на радіочастоті 2,4 ГГц та використовується для бездротового обміну даними між різними пристроями. Він може бути використаний для встановлення комунікаційного зв'язку між декількома системами або між БПЛА та віддаленим пультом керування.

Щоб забезпечити взаємодію з модулем NRF24 за допомогою Python, можна скористатися бібліотекою "RF24" або "nRF24L01". Ці бібліотеки надають інтерфейс для взаємодії з модулем NRF24 та спрощують процес передачі та отримання даних через бездротовий канал [19].

Для навчання нейронної мережі, слід використати бібліотеку TensorFlow. Це відкрита платформа для розробки та навчання глибоких нейронних мереж. Створена компанією Google Brain, TensorFlow надає розробникам широкий спектр інструментів для створення та навчання штучних нейронних мереж, використовуючи різноманітні архітектури.

Однією з ключових особливостей TensorFlow є його модульність та гнучкість. Він дозволяє створювати складні нейронні мережі з використанням різноманітних шарів, вузлів та оптимізаторів. TensorFlow підтримує широкий спектр задач машинного навчання, включаючи класифікацію, регресію, сегментацію, виявлення об'єктів, генерацію тексту та багато інших.

Крім того, TensorFlow надає зручне середовище для роботи з нейронними мережами. Вона підтримує як використання з популярними високорівневими інтерфейсами, такими як Keras, для швидкого створення моделей, так і можливість роботи на низькорівневому рівні для реалізації власних алгоритмів та оптимізації.

Одним з істотних переваг TensorFlow є його масштабованість та здатність працювати як на одному комп'ютері, так і на розподілених обчислювальних ресурсах, включаючи графічні процесори (GPU) та Tensor Processing Units (TPU).

Завдяки активній спільноті та постійному розвитку, TensorFlow залишається однією з провідних платформ для розробки нейронних мереж та застосування їх у різноманітних галузях, від медицини до фінансів та автомобільної промисловості.

2.2 Процес навчання нейронної мережі

Процес навчання нейронної мережі у TensorFlow – це складний, але фундаментальний етап в розробці моделей машинного навчання. Спочатку потрібно чітко визначити, яку задачу ви хочете вирішити. Це може бути класифікація, регресія, сегментація, генерація тексту чи інші завдання.

Після визначення задачі наступний крок – підготовка даних. Це включає в себе очищення, нормалізацію та розбиття даних на тренувальний та тестовий набори.

Далі йде створення моделі, визначення архітектури нейронної мережі. У TensorFlow це включає в себе створення відповідних шарів та налаштування параметрів моделі [20].

Після створення моделі йде етап компіляції, де визначають функцію втрат та оптимізатор для навчання, а також метрики, які слід відстежувати.

Далі настає етап навчання, де модель пристосовується до тренувальних даних. Цей етап може займати різну кількість часу в залежності від складності моделі та обсягу даних.

Після навчання моделі проводиться оцінка на тестовому наборі для визначення її ефективності та здатності до узагальнення на нові дані.

Якщо необхідно, може бути здійснене тонке налаштування моделі чи процесу навчання для досягнення кращих результатів.

Завершальним етапом є застосування навченої моделі для прогнозування на нових даних або в іншому контексті вирішення завдання. У TensorFlow для кожного з цих етапів існують відповідні інструменти та функції для ефективної розробки та навчання моделей глибокого навчання.

Для комфортного навчання моделей бібліотекою TensorFlow зазвичай використовують інтерактивні середовища виконання коду. Наприклад Jupyter Notebook. Інтерактивне середовище для виконання коду, візуалізації даних та написання тексту в одному документі. Він дозволяє об'єднати код, візуалізацію результатів та пояснення в єдиному файлі, що робить його потужним інструментом для наукових досліджень, навчання та спільної роботи. Ключові особливості Jupyter Notebook включають інтерактивність, підтримку різних мов програмування (Python, R, Julia), візуалізацію даних, підтримку текстового формату Markdown для вставки коментарів та легкість редагування та експорту документів у різні формати, такі як HTML, PDF та інші. Jupyter Notebook широко використовується у наукових дослідженнях, навчанні та аналізі даних [21].

Використання подібних інструментів забезпечує ізольовану середу для розробки, яка дозволяє контролювати версії бібліотек та пакетів.

Для навчання моделі для системи використовувався сервіс Google Colaboratory. Він базується на технології Jupyter Notebook і надає користувачам можливість виконувати код, візуалізувати дані та навіть навчати моделі машинного навчання безпосередньо у веб-браузері. Однією з ключових переваг Google Colab є можливість використовувати потужні графічні процесори (GPU) для прискорення обчислень, що особливо корисно при роботі з великими обсягами даних чи складними моделями машинного навчання. Також Colab інтегровано зі службами Google Drive, що дозволяє зберігати та спільно працювати над проектами в хмарі.

Для навчання моделі із використанням структурованих не навчаних моделей, створено стандартний алгоритм навчання, який складається із 6-ти основних кроків. Алгоритм для навчання моделі на рисунку 2.1.

Рисунок 2.1 – Алгоритм навчання моделі

Для пришвидшення навчання та використання відеоядер, було придбано додаткові ресурси.

Навчати мережу знаходити об’єкти на зображення, краще на основі оптимізованих не навчених моделях. Список таких моделей надає розробник бібліотеки TenserFlow. Вона називається TensorFlow 2 Detection Model Zoo. Список більше ніж з 30-ти готових до навчання моделей. На рисунку 2.2 зображено частинку списку моделей.

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
CenterNet MobileNetV2 FPN 512x512	6	23.4	Boxes
CenterNet MobileNetV2 FPN Keypoints 512x512	6	41.7	Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes

Рисунок 2.2 – Список моделей для визначення об’єктів

У списку представлені статичні дані ефективності моделей та тип результатів, які видає модель.

Так як модель буде використовуватися на міні-комп’ютері, то треба використовувати оптимізовану під мобільні платформи модель. Саме тому обрано моделі SSD MobileNet V2 FPNLite 320x320 та SSD MobileNet V2 FPNLite 640x640.

Перш за все, необхідно підготувати набори даних, на яких буде проводитися навчання. Для цього було використано інструмент LabelImg. Він забезпечує можливість проводити розмітку зображень та автоматичне створення xml файлів, які зберігають інформацію про розмічений об’єкт [22].

Для навчання моделі було розмічено 350 зображень із об’єктами.

Першим етапом навчання моделі є встановлення необхідних шляхів, де знаходиться файли для навчання, пайплайн моделі та сама модель. Код на рисунку 2.3.

```
import os
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8'
PRETRAINED_MODEL_URL =
'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobi
lenet_v2_fpnlite_640x640_coco17_tpu-8.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-
trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow',
'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models',
CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

Рисунок 2.3 – Визначення шляхів

Цей код призначений для налаштування середовища розробки та підготовки для використання нейронних мереж для завдань об'єктного розпізнавання з використанням бібліотеки TensorFlow. Основні дії включають визначення імені для власної та завантаженої моделей, задання URL-адреси для завантаження попередньо сформованої моделі, визначення шляхів до різних каталогів та файлів, які будуть використовуватися під час розробки та навчання моделі, а також перевірка існування кожного шляху та, у випадку його відсутності, створення відповідного каталогу. Код створює необхідну структуру каталогів і перевіряє наявність необхідних шляхів та файлів, створюючи їх за потреби.

Наступним етапом є налаштування середовища для використання TensorFlow API. Код перевіряє наявність необхідного каталогу, клонує репозиторій TensorFlow, а потім встановлює API, забезпечуючи його коректну роботу. Крім того, враховані різні варіанти для операційних систем, таких як Linux та Windows, і виконуються відповідні дії для кожної з них. Приклад коду зображений на рисунку 2.4.

```
if os.name=='nt':
    !pip install wget
    import wget

if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
# Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto
--python_out=. && cp object_detection/packages/tf2/setup.py . && python -m
pip install .
if os.name=='nt':
url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/pr
otoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep +
os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto
--python_out=. && copy object_detection\\packages\\tf2\\setup.py setup.py &&
python setup.py build && python setup.py install
    !cd Tensorflow/models/research/slim && pip install -e .
```

Рисунок 2.4 – Налаштування середовища

Далі необхідно завантажити та розпакувати модельний файл для використання при навчанні [23]. Дії варіюються залежно від операційної системи: для Linux (posix) використовується команда wget, а для Windows (nt) – wget.download. Потім файли переміщуються та розпаковуються за допомогою команд mv і tar для Linux або move і tar для Windows відповідно. Код представлено на рисунку 2.5.

```
if os.name =='posix':
    !wget {PRETRAINED_MODEL_URL}
    !mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf
{PRETRAINED_MODEL_NAME+'.tar.gz'}
if os.name == 'nt':
    wget.download(PRETRAINED_MODEL_URL)
    !move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf
{PRETRAINED_MODEL_NAME+'.tar.gz'}
```

Рисунок 2.5 – Завантаження моделі

Далі необхідно сформувати файл лейблів для об’єктів. Для цього виконаємо наступний код, який зображено на рисунку 2.6.

```
labels = [{'name':'tank', 'id':1}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

Рисунок 2.6 – Створення фалу лейблів об’єктів

Наступний код розпаковує архівний файл archive.tar.gz у шляху IMAGE_PATH, якщо він існує. Після цього клонує репозиторій, що містить скрипт для генерації файлів TFRecord, у шлях SCRIPTS_PATH. Встановлює конкретну версію бібліотеки protobuf. Нарешті, використовуючи скрипт generate_tfrecord.py, створює файли TFRecord для тренування та тестування,

використовуючи зображення з об'єктами та відповідними мітками. Код представлено на рисунку 2.7.

```
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord
{paths['SCRIPTS_PATH']}
!pip install protobuf==3.20.1
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'],
'train.record')}
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'],
'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'],
'test.record')}
```

Рисунок 2.7 – Створення TFRecords

Далі необхідно скопіювати пайплайн не навченої моделі та підготувати його для оновлення під навчання на конкретному середовищі (рисунок 2.8).

```
if os.name =='posix':
    !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}
if os.name == 'nt':
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'],
PRETRAINED_MODEL_NAME, 'pipeline.config')}
{os.path.join(paths['CHECKPOINT_PATH'])}

import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
config
```

Рисунок 2.8 – Копіювання та підготовка пайплайна

Після того, як файл конфігурації був імпортований, необхідно налаштувати конфігурацію для моделі. Цей код використовує протокол pipeline_pb2 для створення та налаштування конфігурації навчання та оцінки моделі. Він читає існуючий файл конфігурації з PIPELINE_CONFIG, змінює певні параметри, такі як кількість класів, розмір пакета, шлях до чекпоінту для доналаштування, тип чекпоінту та інші. Після внесення змін у конфігурацію,

він зберігає оновлену конфігурацію у той самий файл PIPELINE_CONFIG. Код представлено на рисунку 2.9.

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint =
os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] =
[os.path.join(paths['ANNOTATION_PATH'], 'test.record')]
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

Рисунок 2.9 – Зміна конфігурації моделі для навчання

Останній етап навчання це формування команди для запуска процесу навчання. Один із головних параметрів цього процесу є кількість кроків навчання. При використанні моделі із вхідною розмірністю зображення 640 на 640 пікселів, процес навчання із 2000 кроків займає близько 4-х годин. Код запуску процесу навчання зображено на рисунку 2.10.

```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'model_main_tf2.py')

command = "python {} --model_dir={} --pipeline_config_path={} --
num_train_steps=2000".format(TRAINING_SCRIPT,
paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
print(command)
```

Рисунок 2.10 – Команда початку навчання

Після завершення навчання, можна провести тестування моделі на одному із зображень.

Код використовує TensorFlow та інші бібліотеки для виявлення об’єктів на зображенні за допомогою попередньо навченої моделі. Він завантажує та відновлює модель, після чого використовує її для виявлення об’єктів на

тестовому зображенні. Функція detect_fn приймає зображення та повертає результати виявлення у вигляді словника.

Результати виявлення візуалізуються на тестовому зображенні за допомогою бібліотеки Matplotlib. Кожен виявлений об’єкт позначається на зображенні з вказівкою класу та ймовірності [24].

Для взаємодії з TensorFlow та обробки зображень використовуються бібліотеки OpenCV та NumPy. Код надає візуальне представлення результатів виявлення об’єктів на тестовому зображенні, що може бути корисним для аналізу та дослідження виявлених об’єктів. Приклад коду на рисунку 2.11.

```
config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'],
is_training=False)
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-
3')).expect_partial()
@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
category_index =
label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', '67.jpg')
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)
input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0),
dtype=tf.float32)
detections = detect_fn(input_tensor)
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections
detections['detection_classes'] =
detections['detection_classes'].astype(np.int64)
label_id_offset = 1
image_np_with_detections = image_np.copy()
viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)
plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```

Рисунок 2.11 – Процес тестування моделі на зображенні

Результат обробки зображення представлені на рисунку 2.12.

Рисунок 2.12 – Результат виявлення об’єкту

На цьому етапі є можливість оцінити результати навчання, таких як втрати класифікації, локалізації та загальні. В контексті навчання моделей, втрати (або функція втрат) визначають різницю між прогнозованими значеннями моделі та фактичними значеннями в наборі даних. Основна мета підбору параметрів моделі під час навчання – це мінімізація цих втрат, тобто зменшення різниці між прогнозованими та реальними значеннями [25].

Функція втрат визначає, як ефективно модель вирішує конкретне завдання. Наприклад, для задачі регресії втрати можуть вимірювати квадратичну або абсолютну різницю між прогнозованими та фактичними значеннями. У випадку задачі класифікації функції втрат можуть використовувати категоріальну крос-ентропію або інші метрики, які відображають точність класифікації.

Мінімізація втрат є ключовим етапом у навчанні моделей машинного навчання, і це виконується за допомогою алгоритмів оптимізації, таких як стохастичний градієнтний спуск. Підбір адекватної функції втрат і правильних параметрів для оптимізації впливає на якість та ефективність навчання моделі.

Для того, щоб провести аналіз результатів можна використовувати інструменти TensorBoard. Для цього необхідно виконати команду tensorboard –logdir ./. Після цього піднімається локальний сервер, який відображає залоговані дані навчання. Результати втрат на рисунку 2.13.

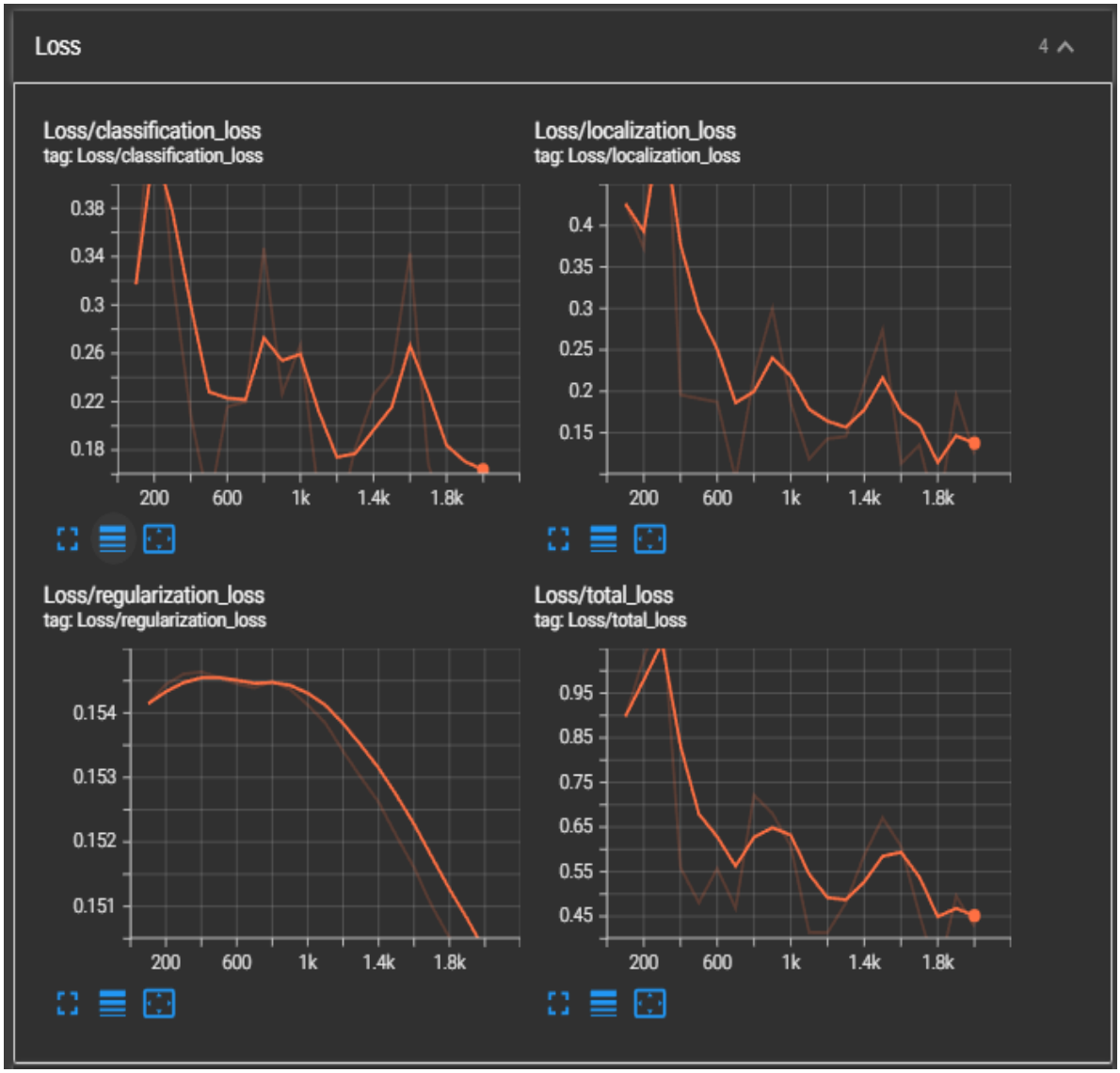

Рисунок 2.13 – Результат втрат при навчанні

Виходячи із графіків, можна прослідкувати, що відсоток загальних втрат досягла 45% за 2000 кроків. Цього зазвичай недостатньо для точного відстеження об’єктів. Це пов’язано із малою кількістю даних для навчання та кількості кроків.

Також можна оцінити точність виявлення на певних зображеннях. На рисунку 2.14 два зображення, зліва результат визначення за допомогою моделі, а на правому – істинне зображення, яке було розмічено вручну [26].

Рисунок 2.14 – Порівняння істинного зображення та передбачуваного

Заключним етапом роботи є компіляція моделі, конвертування у формат TFLite та створення архіву для вивантаження із серверу Google Colab. Для цього виконаємо команди, які зображені на рисунку 2.15.

```
FREEZE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'exporter_main_v2.py ')
command = "python {} --input_type=image_tensor --pipeline_config_path={} --
trained_checkpoint_dir={} --output_directory={}".format(FREEZE_SCRIPT
,files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'], paths['OUTPUT_PATH'])
print(command)
TFLITE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research',
'object_detection', 'export_tflite_graph_tf2.py ')
command = "python {} --pipeline_config_path={} --trained_checkpoint_dir={} --
output_directory={}".format(TFLITE_SCRIPT ,files['PIPELINE_CONFIG'],
paths['CHECKPOINT_PATH'], paths['TFLITE_PATH'])
print(command)
FROZEN_TFLITE_PATH = os.path.join(paths['TFLITE_PATH'], 'saved_model')
TFLITE_MODEL = os.path.join(paths['TFLITE_PATH'], 'saved_model',
'detect.tflite')
command = "tflite_convert \
--saved_model_dir={} \
--output_file={} \
--input_shapes=1,300,300,3 \
--input_arrays=normalized_input_image_tensor \
output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1'
,'TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' \
--inference_type=FLOAT \
--allow_custom_ops".format(FROZEN_TFLITE_PATH, TFLITE_MODEL, )
print(command)
```

Рисунок 2.15 – Компіляція та конвертація навченої моделі

Це заключний етап навчання моделі. Далі необхідно буде підготувати відповідний код, який буде використовувати цю модель у виявленні.

2.3 Висновки до другого розділу

Проведено аналіз та порівняння кількох мов програмування. Виділено сильні та слабкі сторони кожної з них. У результаті аналізу, зроблено висновок, що серед представлених мов, доцільніше використовувати мову Python. Так як вона надає великий об’єм додаткових бібліотек та інструментів.

Також зроблено огляд процесу навчання нейронної мережі. Виведено алгоритм навчання. Проведено детальний аналіз кожного з кроків процесу навчання мережі, представлені приклади коду та результати.

Із отриманих результатів можна зробити висновок, що модель потребує збільшення даних для навчання та кількості кроків при навчанні.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обробка зображення за допомогою навченої моделі.

Для того щоб використовувати навчену модель у коді, необхідно використовувати бібліотеку tensorflow. Важливо дотримуватися версії бібліотеки, на якій проходило навчання. Недотримання може привести до того, що створена модель не зможе бути прочитана бібліотекою іншої версії. Усі методи знаходяться у окремому файлі програми neural_detection.py.

Першим методом, який необхідно реалізувати, це метод читання файлу лейблів, тобто імен об’єктів, на яких була навчена нейронна мережа. Для цього треба створити текстовий файл, який буде містити список імен об’єктів.

Всередині методу, файл із іменами відкривається із кодування utf-8. Після цього проводиться зчитування строк відкритого файлу. Далі запускається цикл, який формує словник, який містить номер об’єкту та його назву. У кінці метод повертає створений словник. Код представлено на рисунку 3.1.

```
def load_labels(path='labels.txt'):
    with open(path, 'r', encoding='utf-8') as f:
        lines = f.readlines()
        labels = {}
        for row_number, content in enumerate(lines):
            pair = re.split(r'[:\s]+', content.strip(), maxsplit=1)
            if len(pair) == 2 and pair[0].strip().isdigit():
                labels[int(pair[0])] = pair[1].strip()
            else:
                labels[row_number] = pair[0].strip()
    return labels
```

Рисунок 3.1 – Метод завантаження списку імен об’єктів

Далі необхідно створити метод попередньої обробки зображення та завантаження моделі.

Метод завантаження моделі представляє собою виконання вбудованої функції бібліотеки TensorFlow. Вхідними даними для відкриття моделі є шлях до файлу самої моделі. Метод повертає змінну формату saved_model.

Метод попередньої обробки зображення використовується для конвертації вхідного кольорового зображення будь-якого розміру у розмір 640 на 640 пікселів. Далі кожен піксель зображення збільшується на 255 та конвертується у тип даних unit8, що відповідає 8-ми бітному цілочисельному типу. Далі значення пікселів нормалізується у діапазоні від 0 до 1. Далі додається додатковий розмір у розмірність осі 0, створюючи батч з одного зображення.

У кінці зображення повертається із методу. Код представлено на рисунку 3.2.

```
def load_model(model_path):
    model = tf.saved_model.load(model_path)
    return model


def preprocess_image(image):
    """Preprocess the input image."""
    image = cv2.resize(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), (320, 320))
    image = (image * 255).astype(np.uint8)
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    return image
```

Рисунок 3.2 – Метод завантаження моделі та обробки зображення

Останній необхідний метод у файлі, це метод виявлення об’єктів за допомогою моделі. Вхідними параметрами методу є сама модель, оброблене зображення та поріг прогнозування.

Першим етапом є виведення інформації про модель та параметри вхідних даних, які вона очікує. Це використовується для проведення аналізу та перевірки правильності зображення, яке подається до моделі. Далі вхідне зображення проходить нормалізацію. Після цього воно подається до моделі та повертається словник із результатами передбачення. Моделі зазвичай

повертають такі значення, як координати областей (bounding boxes), класи об'єктів, ймовірності (scores) та кількість об'єктів.

Далі отриманий словник розпаковується на чотири змінні. А саме: координати виявлених об’єктів на зображенні, класи об’єктів, відсоток співпадіння об’єкту на виділеній області зображення та кількість знайдених об’єктів.

Наступним кроком, метод пропускає отримані данні через цикл, який зчитує зібрану інформацію по кожному знайденому об’єкту, перевіряє відсоток співпадіння та формує вихідні результати, які містять положення об’єкта на зображенні, ім’я класу об’єкта та відсоток співпадіння. Код методу на рисунку 3.3.

```
def detect_objects(model, image, threshold):
    input_tensor_info = model.signatures["serving_default"].inputs
    img_uint8 = (image * 255).astype(np.uint8)
    output_dict = model.signatures["serving_default"](tf.constant(img_uint8))

    boxes = output_dict['detection_boxes'].numpy().tolist()[0]
    classes =
output_dict['detection_classes'].numpy().astype(int).tolist()[0]
    scores = output_dict['detection_scores'].numpy().tolist()[0]
    count = int(output_dict['num_detections'].numpy())

    results = []
    for i in range(count):
        if scores[i] >= threshold:
            result = {
                'bounding_box': boxes[i],
                'class_id': classes[i],
                'score': scores[i]
            }
            results.append(result)
    return results
```

Рисунок 3.3 – Метод отримання результатів виявлення

3.2 Дослідження трекерів об’єктів та розробка методів трекінгу

Трекінг об'єктів у відео або зображеннях є важливою задачею в області комп'ютерного зору та обробки зображень. OpenCV, відкрита бібліотека для обробки зображень і комп'ютерного зору, пропонує численні інструменти для трекінгу об'єктів.

Трекінг – це процес відстеження переміщення об'єкта на послідовних кадрах відео. Він використовується в багатьох областях, таких як слідкування об'єктів на відеозаписах, аналіз поведінки людей.

Усі трекери в OpenCV належать до окремого API довгострокового оптичного відстеження.

Довгострокове оптичне відстеження є важливою проблемою для багатьох програм комп'ютерного бачення. Розробка в цій галузі дуже фрагментована, і цей API є унікальним інтерфейсом, корисним для підключення кількох алгоритмів і їх порівняння.

Є три основні компоненти побудови трекеру на основі API: TrackerSampler, TrackerFeatureSet і TrackerModel. Перший компонент — це об'єкт, який обчислює патчі над кадром на основі останнього цільового розташування. TrackerFeatureSet — це клас, який керує функціями, можливе підключення багатьох із існуючих функцій (HAAR, HOG, LBP, Feature2D тощо). Останнім компонентом є внутрішнє представлення цілі, це модель зовнішності. Він зберігає всі варіанти станів і обчислює траєкторію (найбільш вірогідні цільові стани). Клас TrackerTargetState представляє можливий стан цілі. TrackerSampler і TrackerFeatureSet є візуальним представленням цілі, натомість TrackerModel є статистичною моделлю. Структура API трекерів зображена на рисунку 3.4.

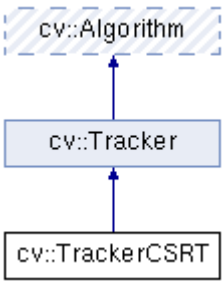

Рисунок 3.4 – Стандартна структура API трекерів у OpenCV

OpenCV надає низку вбудованих трекерів, які допомагають в реалізації трекінгу. Деякі з них включають:

– BOOSTING Tracker: базується на тому ж алгоритмі, що використовується для машинного навчання, що лежить в основі каскадів Хаара (AdaBoost), але, як і каскадам Хаара, йому більше десяти років;

– Трекер MIL: надає кращу точність, ніж трекер BOOSTING, але погано повідомляє про помилку відстеження;

– KCF Tracker: Kernelized Correlation Filters. Швидший ніж BOOSTING і MIL. Подібно до MIL, KCF погано справляється з повною оклюзією;

– CSRT Tracker: дискримінаційний кореляційний фільтр (з канальною та просторовою надійністю);

– GOTURN Tracker: єдиний детектор об’єктів на основі глибокого навчання, включений у OpenCV. Для запуску потрібні додаткові файли моделі.

Для роботи з трекерами у OpenCV, спочатку потрібно ініціалізувати об'єкт трекера з використанням відповідного конструктора. Після ініціалізації, трекер встановлюється на об'єкт, який необхідно відстежувати, і потім він оновлює позицію об'єкта на кожному новому кадрі відео [27].

Для проведення дослідження кожного із доступних трекерів, було створено тестову програму. За її допомогою можна відслідкувати стабільність відстеження та частоту кадрів, тобто оптимізацію відстеження. Усі трекери будуть тестуватися у однакових умовах на однаковому об’єкті. Під час тестування об’єкт відстеження буде змінювати своє положення у просторі та перекриватися іншими об’єктами.

Варто зазначити, що тестування будуть проходити CSRT, KCF та MIL трекери, так як вони не вимагають додаткових налаштувань та файлів для проведення відстеження. Це є одним із факторів оптимізації програмного коду для успішного використання у системах з обмеженою обчислювальною здатністю.

CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability) є одним з трекерів, доступних у бібліотеці OpenCV для відстеження об'єктів у відео та зображеннях. Він створений на основі методу кореляції, що

дозволяє визначити подібність між шаблоном об'єкта та кожним новим кадром відео [28].

Спочатку трекер CSRT створює модель об'єкта, що включає в себе інформацію з різних каналів (наприклад, колір, яскравість тощо). Ця модель допомагає зберегти інформацію про об'єкт і використовується для кореляції з новими кадрами.

CSRT використовує кореляцію між шаблоном об'єкта і кожним новим кадром відео для визначення, де знаходиться об'єкт на новому кадрі. Цей процес включає обчислення кореляції між шаблоном та новим кадром у різних масштабах та орієнтаціях, що дозволяє виявити об'єкт, навіть якщо він змінив свій розмір або орієнтацію.

Ще однією особливістю CSRT є спрощення зображення, яке використовується для внутрішнього обчислення. Зображення конвертується в чорно–біле та зменшується. Це допомагає зменшити обчислювальну складність і покращити продуктивність трекера.

CSRT є стійким до невеликих зсувів об'єкта на кадрі і може відстежувати об'єкт навіть у випадках, коли об'єкт тимчасово виходить за межі видимості на кадрі.

Проведемо тестування стабільності відстеження об’єкту та вплив відстеження на кількість кадрів за секунду. Результати тестування на рисунках 3.5 та 3.6.

Рисунок 3.5 – CSRT трекер, початок відстеження

Рисунок 3.6 – CSRT трекер, кінець відстеження

Виходячи з отриманих результатів, можна зробити висновок, що CSRT трекер стабільно відстежує об’єкти при зміні їх положення, перекриванням іншими об’єктами. Відстеження протрималося 22,3 секунди від початку. Але враховуючи початкову частоту відео у 60 кадрів на секунду, трекер навантажує систему та призводить до зменшення частоти кадрів до 24–29 на секунду.

KCF (Kernelized Correlation Filter) – це інший трекер для відстеження об'єктів у відео та зображеннях, який також використовує кореляцію для

визначення подібності між шаблоном об'єкта та кожним новим кадром відео. KCF має деякі особливості та переваги.

Використання фільтрів з ядрами. Основна ідея KCF – використовувати фільтри з ядрами для покращення точності трекера. Ці фільтри розглядають кореляцію між шаблоном об'єкта та кожним новим кадром відео як операцію згортки з використанням ядра.

Оновлення моделі згортки: KCF постійно оновлює свою модель об'єкта, використовуючи інформацію про кореляцію. Це допомагає забезпечити стійкість до змін у зовнішньому середовищі та положенні об'єкта.

Спрощення зображення: крім цього, KCF зазвичай використовує спрощені представлення зображень, такі як чорно–білі або зменшені за рахунок шкалювання, щоб зменшити обчислювальну складність та покращити продуктивність.

Стійкість до зсуву та відсутності: KCF також є стійким до невеликих зсувів об'єкта на кадрі і може відстежувати об'єкт, навіть якщо об'єкт тимчасово виходить за межі видимості на кадрі.

Швидкодія: KCF відомий своєю високою швидкістю роботи і може відстежувати об'єкти в реальному часі.

KCF є популярним і швидким методом для відстеження об'єктів у відео та зображеннях, і він використовується в різних додатках, де необхідно відстежувати об'єкти в реальному часі, таких як відеоспостереження, автоматичне водіння, розпізнавання обличчя та багато інших [29].

Проведемо тестування трекеру у тих самих умовах. Результат тестування на рисунках 3.7 та 3.8.

Рисунок 3.7 – KCF трекер, початок відстеження

Рисунок 3.8 – KCF трекер, кінець відстеження

Як видно із результатів тестування, KCF не в змозі відстежувати об’єкти при зміні наближення, положення та перекривання об’єкта відстеження. Трекінг протримався лише 2,4 секунди. Однак використання цього типу трекера збільшує частоту кадрів до 32–35 на секунду.

TrackerMIL – це трекер, який використовує навчання з прикладами для відстеження об'єктів у відео. Він працює, намагаючись знайти об'єкт на

кожному кадрі відео та слідкувати за ним на наступних кадрах. TrackerMIL заснований на ідей про те, як об'єкти можна відслідковувати на основі навчання з прикладами. Розглянемо основні принципи роботи TrackerMIL.

Навчання на прикладах: Спочатку він навчається розпізнавати об'єкт на основі прикладів. Для цього використовуються позитивні і негативні зразки. Позитивні зразки – це фрагменти, які містять об'єкт, який ви намагаєтеся відстежувати. Негативні зразки – це фрагменти, де об'єкт відсутній.

Відстеження об'єкта: після навчання він використовує отриманий класифікатор для визначення, чи належить об'єкт до цієї колекції позитивних зразків на нових кадрах. Якщо об'єкт відслідковується, то його положення оновлюється.

Оновлення класифікатора: TrackerMIL може час від часу оновлювати свій класифікатор, враховуючи нові позитивні зразки на кожному кадрі. Це допомагає пристосовуватися до змін у вигляді об'єкта та умовах освітлення.

Спрощення зображень: щоб полегшити обчислення, TrackerMIL може використовувати спрощені версії зображень, такі як чорно-білі або зменшені за рахунок шкалювання.

Спроможність працювати в реальному часі: TrackerMIL спроектований для використання в режимі реального часу та може відслідковувати об'єкти при руху і змінах у вигляді.

Загальна ідея полягає в тому, що TrackerMIL використовує навчання на прикладах для відстеження об'єкта на кожному кадрі відео. Це дозволяє використовувати його в різних застосуваннях, таких як відеоспостереження, автоматичне водіння та багато інших.

Проведемо тестування цього трекеру. Результати на рисунках 3.9 та 3.10.

Рисунок 3.9 – MIL трекер, початок тестування

Рисунок 3.10 – MIL трекер, кінець тестування

Виходячи с проведеного тестування MIL трекеру, можна дійти висновку, що стабільність відстеження на високому рівні. Зміна положення та перекривання об’єкта не заважає відстеженню. Але варто зазначити, що при використанні цього типу трекеру значно падає показник кадрів на секунду, до 11–15 раз на секунду.

На основі проведених тестувань створено таблицю 3.1, яка відображає переваги та недоліки трьох типів трекерів.

Таблиця 3.1 – Результати тестування трекерів

Тип трекеру	Падіння FPS	Відстеження при зміні положення об’єкту	Відстеження при зміні кута об’єкту	Відстеження при перекриванні об’єкту
CSRT	26	Так	Так	Так
KCF	32	Так	Ні	Ні
MIL	12	Так	Так	Так

За отриманими результатами можна зробити висновок, що для відстеження об’єкту в складних умовах та при обмеженій обчислювальної здатності, найкраще підходить саме CSRT трекер.

Перейдемо до побудови коду для використання трекеру у загальній системі відеонагляду. Для методів використання трекеру також створено окремий файл, який має назву tracker_functions.py.

Перш за все, організовано клас трекерів для швидкої зміни типу відстеження. Клас представляє собою 3 змінних, яким присвоюється по кожному із трьох трекерів. Код класу на рисунку 3.11.

```
class Trackers:
    tracker_1 = cv2.TrackerCSRT.create()
    tracker_2 = cv2.TrackerKCF.create()
    tracker_3 = cv2.TrackerMIL.create()
```

Рисунок 3.11 – Клас трекерів

Далі розроблено функцію використання трекеру. Вона приймає на вхід зображення (frame), інформацію про поточний стан відстеження (tracking, tracking_ok), область об'єкта, який відстежується (bbox), та інші параметри.

Функція спочатку отримує розмір зображення (height, width). Потім визначає центральну точку зображення (center_x, center_y).

Далі виконується логіка відстеження об'єкта. Якщо режим відстеження увімкнено (tracking), ініціалізується новий трекер з використанням області об'єкта (bbox). Якщо ініціалізація успішна, змінні tracking_ok та tracking встановлюються в True та False відповідно. Область об'єкта (bbox1) ініціалізується нульовими значеннями.

Далі визначається центр цільової області (target_center). Якщо відстеження увімкнено і виконано оновлення координат області (ret1), область оновлюється на кадрі. Центр цільової області обчислюється як середина області.

Якщо оновлення не вдалося, встановлюється прапор object_lost (втрата об'єкта), tracking_ok встановлюється в False, і відстеження об'єкта призупиняється.

На виході функція повертає змінене зображення (frame), нову область об'єкта (bbox1), стан відстеження (tracking, tracking_ok), прапор втрати об'єкта (object_lost) та центр цільової області (target_center). Приклад коду зображено на рисунку 3.12.

```
def tracking_process(frame, tracking, tracking_ok, bbox, object_lost):
    height, width, _ = frame.shape
    center_x = width // 2
    center_y = height // 2
    if tracking:
        Trackers.tracker_1.init(frame, bbox)
        print(Trackers.tracker_1.init(frame, bbox))
        print(bbox)
        tracking_ok = True
        tracking = False
    bbox1 = [0, 0, 0, 0]
    target_center = [center_x, center_y]
    if tracking_ok:
        ret1, bbox1 = Trackers.tracker_1.update(frame)
        if ret1:
            bbox1 = [int(e) for e in bbox1]
            cv2.rectangle(frame, (bbox1[0], bbox1[1]), (bbox1[0] + bbox1[2],
bbox1[1] + bbox1[3]), (0, 255, 0), 2)
            frame = draw_lines_to_edges(frame, bbox1)
            target_center = (bbox1[0] + bbox1[2]//2,bbox1[1] + bbox1[3]//2)
        else:
            object_lost = True
            tracking_ok = False
    return frame, bbox1, tracking, object_lost, tracking_ok, target_center
```

Рисунок 3.12 – Функція трекінгу об’єкта

Також, при відстеженні виконується інша функція, яка створює навколо цілі геометричні фігури, які дають змогу зрозуміти, де саме знаходиться ціль. Код функції зображено на рисунку 3.13.

```
def draw_lines_to_edges(frame, bbox):
    x, y, w, h = [int(e) for e in bbox]
    center = (x + w // 2, y + h // 2)

    height, width, _ = frame.shape

    cv2.line(frame, center, (0, center[1]), (0, 255, 0), 2)
    cv2.line(frame, center, (width, center[1]), (0, 255, 0), 2)
    cv2.line(frame, center, (center[0], 0), (0, 255, 0), 2)
    cv2.line(frame, center, (center[0], height), (0, 255, 0), 2)

    return frame
```

Рисунок 3.13 – Функція обробки зображення після трекеру

3.3 Визначення відстані до об’єктів

Визначення відстані до об'єктів – завдання, яке може бути вирішено різними методами в залежності від конкретного контексту та умов вимірювань. Один із широко використовуваних методів – тригонометрична тріангуляція. Цей підхід ґрунтується на вимірюванні кутів та використанні трикутникової геометрії для розрахунку відстані між спостерігачем і об'єктом. Інший ефективний спосіб – використання лазерних далекомірів. Ці пристрої використовують лазерне випромінювання для вимірювання часу, який лазерному променю потрібно, щоб відбитися від об'єкта та повернутися до приладу.

Не зважаючи на ефективність цих методів, їх інтеграція може бути складною та вимагати значних фінансових затрат. Зокрема, використання лазерних далекомірів може вимагати значних витрат на розробку та інтеграцію системи. У випадках, коли бюджет обмежений або розглядається використання в невеликих системах, доцільним є використання інших доступних методів вимірювання відстаней, які можуть бути більш

економічними та пристосованими до конкретних вимог. У ході дослідження, було проведено аналіз точності вимірювання відстані за допомогою параметрів матриці камери.

Для визначення відстані до об'єкта застосовується формула тонкої лінзи, що є загальноприйнятим методом у фотографії та оптиці. Ця формула, яка зазвичай використовується в об'єктивах звичайних камер, враховує лінійний розмір об'єкта, його розмір на матриці камери, і фокусну відстань лінзи. За допомогою цієї формули можна точно визначити відстань до об'єкта, що є важливим елементом у різних областях, таких як комп'ютерний зір та геодезія. Схема тонкої лінзи представлена на рисунку 3.14.

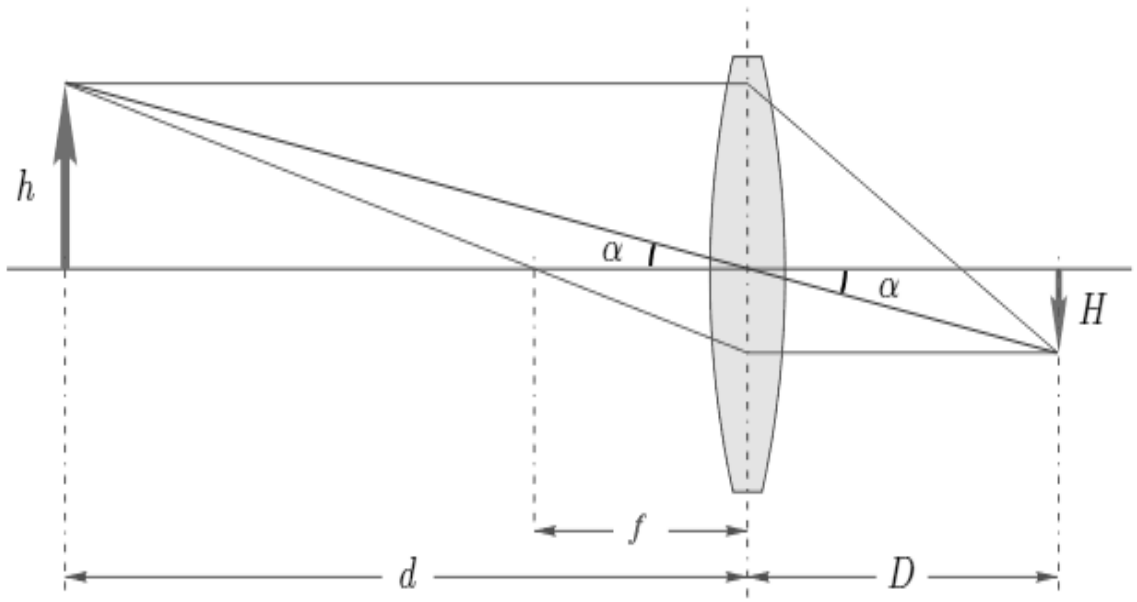

Рисунок 3.14 – Схема тонкої лінзи

При даному типі визначення відстані, необхідно мати сталі данні, які використовуються при розрахунку. А саме розмір об’єкта, фокусна відстань лінзи та розмір об’єкта на матриці об’єктива.

Розрахунок проводиться за формулою:

$$d = \frac{(f(H+h))}{H}, \tag{1}$$

де h – лінійний розмір об’єкта;

H – розмір об’єкта на матриці;

f – фокусна відстань лінзи.

Далі в процесі визначення відстані до об'єкта необхідно звернутися до параметрів обраної камери, якою буде здійснюватися фотозйомка сцени з досліджуваним об'єктом. Обраною опцією є камера із матрицею Samsung ISOCELL GW3, яка має фізичні розміри 6,5 мм х 4,86 мм. Фокусна відстань об'єктива цієї камери складає 25 міліметрів, а розмір вихідного зображення становить 1080 на 1920 пікселів.

На основі цих вхідних даних можна визначити розмір об'єкта на матриці камери, використовуючи зв'язок між фізичними розмірами об'єкта і його відображенням на матриці. Ці дані стають ключовими у подальших розрахунках відстані до об'єкта, дозволяючи враховувати специфікації камери для отримання більш точних результатів вимірювань [30].

Створимо окрему функцію у файлі tracker_functions.py, яка буде розраховувати відстань. Спочатку функція перевіряє, чи область об'єкта (bbox1) не є пустою. Якщо область існує, визначаються параметри камери, такі як розміри зображення та відстань до матриці камери. Для цього створено окремий файл, який зберігає параметри матриць для декількох популярних камер. Код файлу із константами зображено на рисунку 3.15.

```
class CameraConstants:
    def __init__(self, camera_choice='MAIN_CAMERA'):
        self.camera_choice = camera_choice

    @property
    def focal_length(self):
        if self.camera_choice == 'OV2640':
            return 0.00215
        elif self.camera_choice == 'OV2710':
            return 0.00369
        elif self.camera_choice == 'ISOCELL_GW3':
            return 0.025

    @property
    def matrix_y(self):
        if self.camera_choice == 'OV2640':
            return 3.59
        elif self.camera_choice == 'OV2710':
            return 5.85
        elif self.camera_choice == 'ISOCELL_GW3':
            return 6.5

    @property
    def matrix_x(self):
        if self.camera_choice == 'OV2640':
            return 2.68
        elif self.camera_choice == 'OV2710':
            return 3.27
        elif self.camera_choice == 'ISOCELL_GW3':
            return 4.86

    @property
    def image_size(self):
        if self.camera_choice == 'OV2640':
            return [800, 600]
        elif self.camera_choice == 'OV2710':
            return [1920, 1080]
        elif self.camera_choice == 'ISOCELL_GW3':
            return [1920, 1080]

    @property
    def fov(self):
        if self.camera_choice == 'OV2640':
            return 66
        elif self.camera_choice == 'OV2710':
            return 120
        elif self.camera_choice == 'ISOCELL_GW3':
            return 100
```

Рисунок 3.15 – Клас із константами параметрів для різних типів камер

Далі визначається кількість пікселів на один сантиметр (pixels_per_cm). З області об'єкта визначається його реальний розмір в сантиметрах (object_size), який включає півширину області.

Якщо розмір об'єкта не дорівнює 0, обчислюється відстань до об'єкта з використанням формули перспективи та відомих параметрів камери. Якщо розмір об'єкта дорівнює 0, відстань також встановлюється в 0.

На кадрі виводиться визначена відстань у вигляді тексту, та функція повертає змінене зображення (frame) та визначену відстань (distance). Код функції визначення відстані зображено на рисунку 3.16.

```
def distance_calculator(frame, bbox1, real_object_width):
    if not bbox1 == [0,0,0,0]:
        camera = CameraConstants(os.getenv("MAIN_CAMERA"))
        pixels_x, pixels_y = camera.image_size
        pixels_per_cm = pixels_x / camera.matrix_x
        x, y, w, h = [int(e) for e in bbox1]
        object_size = ((w / 2) / pixels_per_cm) / 100
        if object_size != 0:
            distance = (camera.focal_length * (real_object_width +
object_size)) / object_size
        else:
            distance = 0
    else:
        distance = 0
    cv2.putText(frame, f"{round(distance, 2)} M", (100, 300),
cv2.FONT_HERSHEY_DUPLEX, 0.6, (0, 255, 0), 2)
    return frame, distance
```

Рисунок 3.16 – Функція розрахунку відстані до об’єкту

Для тестування методу було створено відеозапис автомобіля, лінійний розмір якого дорівнює приблизно 2 м, так як стоїть під певним кутом до камери. Ці дані необхідно занести до програми для проведення тестування. Результати тестування представлені на рисунку 3.17.

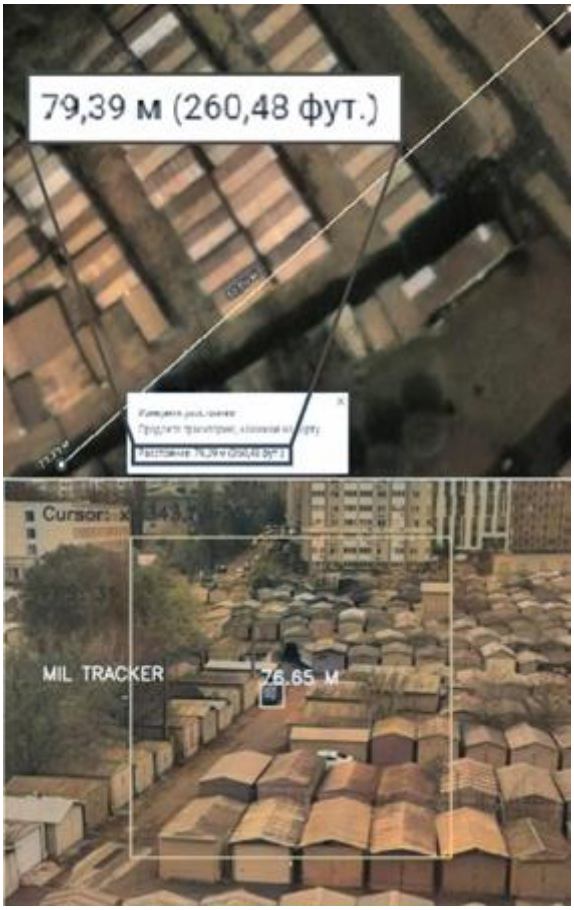

Рисунок 3.17 – Результати тестування методу визначення відстані

Виходячи із отриманих результатів, можна зробити висновок, що використання цього методу дає не достатньо точні дані о відстані, так як розмір одного і того ж об’єкту може змінюватися залежно від його кута розміщення. Також можуть впливати чинники викривлення зображення лінзами, неточний розрахунок розмірів об’єкту у пікселях та інші.

3.4 Управління польотним контролером

Для реалізації коду управління польотним контролером, використовується бібліотека DroneKit. Вона дозволяє встановлювати з’єднання із контролерами серії ArduPilot. Методи розташовані в окремому файлі SITL_drone_control.py.

Перша функція призначена для армування дрона, перевірки його готовності до польоту та виконання зліту на певну висоту.

Армування дрона – це процес підготовки та активації його систем для виконання польоту. Коли дрон армується, йому стає дозволено використовувати свої мотори та системи управління для виконання різних команд. Армування є важливою частиною передпольотної перевірки, і воно може включати в себе різні перевірки, такі як перевірка статусу GPS, стану акумуляторів, ініціалізації компаса та інших систем. Після армування пілот може відправляти команди дрону для виконання різних завдань, таких як взліт, навігація, фотографування чи зйомка відео [31].

Спочатку виводяться базові перевірки перед армуванням, і функція чекає, поки дрон буде готовий до армування (vehicle.is_armable).

Після цього дрон переходить у режим "GUIDED" і армується (vehicle.armed = True). Функція чекає, доки армування завершиться (while not vehicle.armed).

Після армування дрона виконується взліт на вказану висоту (aTargetAltitude) за допомогою команди vehicle.simple_takeoff(aTargetAltitude). Функція виводить інформацію про поточну висоту, чекає, поки дрон досягне 95% вказаної висоти, і тоді виводить повідомлення про досягнення цільової висоти, завершуючи виконання функції. Код функції зображено на рисунку 3.18.

```
def arm_and_takeoff(aTargetAltitude):
    print("Basic pre-arm checks")
    while not vehicle.is_armable:
        time.sleep(1)
    print("Arming motors")
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True
    while not vehicle.armed:
        time.sleep(1)
    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)
    while True:
        print("Altitude: ", vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
            print("Reached target altitude")
            break
        time.sleep(1)
```

Рисунок 3.18 – Функція ініціації дрона та підйому на певну висоту

Наступна функція додаткова, вона призначена для визначення відстані між двома точками на землі в метрах, використовуючи їхні географічні координати (широта і довгота). Для цього використовується бібліотека geopy.

Далі реалізовано функцію, яка визначає нові географічні координати (широта та довгота) для точки, яка знаходиться на певній відстані та заданому азимуту від поточного положення.

Код двох функцій приведено на рисунку 3.19.

```
def get_distance_metres(location1, location2):
    location1_coords = (location1.lat, location1.lon)
    location2_coords = (location2.lat, location2.lon)
    dist_meters = geodesic(location1_coords, location2_coords).meters
    return dist_meters
def calculate_target_gps(current_location, azimuth, target_distance):
    target_coords =
distance(meters=target_distance).destination((current_location.lat,
current_location.lon), azimuth)
    return LocationGlobalRelative(target_coords.latitude,
target_coords.longitude, 1)
```

Рисунок 3.19 – Функції обчислення відстані та положення цілі

Наступна частина коду відповідає за переміщення дрону до заданої географічної точки, яка передається у вигляді об'єкта LocationGlobalRelative. Функція також може зупинити рух дрона, якщо переданий параметр stop_flight має значення True.

Під час виклику функції виводиться повідомлення "Flying forward...", а потім викликається метод simple_goto дрона з переданою точкою як цільовою локацією. Це дозволяє дрону вибратися в задану точку у просторі [32].

Якщо параметр stop_flight має значення True, то після досягнення цільової точки дрон автоматично встановлює свої поточні координати як нову ціль для переміщення, зупиняючи тим самим свій рух. Це може бути використано, наприклад, для тимчасового припинення автопілотованого руху дрона. Приклад коду зображено на рисунку 3.20.

```
def fly_to_target(location: LocationGlobalRelative, stop_flight):
    print("Flying forward...")
    vehicle.simple_goto(location)
    if stop_flight:
        position_for_now = vehicle.location.global_frame
        vehicle.simple_goto(position_for_now)
```

Рисунок 3.20 – Функція переміщення дрону

Наступна функція, яка об’єднує попередні, представляє собою організований окремий потік для виконанні програми для автоматичного керування дроном. Вона працює в безкінечному циклі, очікуючи команди від черги drone_control_queue. Також, вона відправляє інформацію назад через чергу return_drone_info.

Спочатку функція викликає arm_and_takeoff(15), щоб забезпечити зліт дрона на висоту 15 метрів.

Далі вона перевіряє, чи черга drone_control_queue не є порожньою. Якщо отримано новий запит на керування, функція витягує необхідні параметри і розпочинає роботу. Код функції представлено на рисунку 3.21.

```
def drone_auto_control_thread(drone_control_queue: queue.Queue,
return_drone_info: queue.Queue):
    arm_and_takeoff(15)
    while True:
        if not drone_control_queue.empty():
            target_image_point, screen_size, start_flight, stop_flight,
target_distance = drone_control_queue.get()
            print("work")
            if start_flight:
                print("ok")
                screen_center_x = screen_size[1]/2
                now_drone_position = vehicle.location.global_frame
                azimuth_drone_from_north = vehicle.heading
                camera_fov = CameraConstants(os.getenv("MAIN_CAMERA")).fov
                target_x_on_image = target_image_point[0]
                angle_to_target = (camera_fov / screen_size[1]) *
(target_x_on_image - screen_center_x)
                processed_angle = angle_to_target
                print(processed_angle)
                print(azimuth_drone_from_north)

                azimuth_to_target_from_north = (azimuth_drone_from_north +
processed_angle)
                azimuth_to_target_from_north %= 360
                print(azimuth_to_target_from_north)
                target_location =
calculate_target_gps(vehicle.location.global_frame,
azimuth_to_target_from_north,
                                                        target_distance)
                fly_to_target(target_location, stop_flight)
                distance_from_gps = get_distance_metres(now_drone_position,
target_location)
                print("Distance to target: ", distance_from_gps)
                time.sleep(0.025)
                if stop_flight:
                    start_flight = False
                    return_drone_info.put(start_flight)
            else:
                time.sleep(0.025)
        else:
            time.sleep(0.025)
            continue
```

Рисунок 3.21 – Функція потоку контролю дрону

У разі, якщо параметр start_flight встановлений у True, функція розпочинає рух дрона. Вона обчислює кут нахилу цілі відносно центру екрану (processed_angle), обчислює азимут до цільової точки відносно півночі (azimuth_to_target_from_north) та обчислює нові GPS-координати цільової точки. Після цього викликає функцію fly_to_target для переміщення дрона до нової точки. Функція також обчислює відстань до цільової точки та виводить цю інформацію.

Якщо параметр stop_flight встановлений у True, це вказує на зупинку руху дрона. В такому випадку, функція відправляє в чергу return_drone_info сигнал про зупинку руху (start_flight = False).

В разі, якщо черга drone_control_queue порожня, функція продовжує чекати.

3.5 Передача даних через радіомодуль

Для передачі даних через радіомодуль, створено окремий файл із назвою nrf24_communication.py. Він включає в себе код який використовується для обміну даними між дроном та іншим пристроєм за допомогою модуля NRF24. Основна частина коду використовує бібліотеку nrf24 для налаштування та керування модулем NRF24.

Спершу встановлюються параметри, такі як номера пінів CE та CSN, канал, розмір пакету, адреси пристроїв (pipe) та інші, які зчитуються зі змінної оточення. Далі ініціалізується модуль NRF24 з вказаними параметрами.

Головний цикл main_comm_loop виконується постійно. Якщо черга input_queue не є порожньою, береться повідомлення і відправляється через модуль NRF24. Після цього знову встановлюється прослуховування для прийому даних.

Якщо модуль NRF24 отримує дані (функція radio.available() повертає True), то отримані дані зчитуються та виводяться на екран. Після цього дані додаються до черги output_queue для подальшого використання.

Важливо відзначити, що перед кожним читанням чи відправленням даних встановлюється адрес пристрою для прийому/відправлення та викликається time.sleep(1), що робить паузу на 1 секунду між операціями для забезпечення стабільності і уникнення колізій в мережі. Приклад коду на рисунку 3.22.

```
ce_pin = int(os.getenv("CE_PIN"))
csn_pin = int(os.getenv("CSN_PIN"))
radio = NRF24(ce_pin, csn_pin)
radio.begin(ce_pin, csn_pin)
channel = int(os.getenv("NRF_CHANNEL"))
radio.setChannel(channel)
pipe = [bytes(os.getenv("PIPE_RX")), bytes(os.getenv("PIPE_TX"))]
payload_size = int(os.getenv("PAYLOAD_SIZE"))
radio.setPayloadSize(payload_size)

radio.openReadingPipe(1, pipe[1])
radio.startListening()


def main_comm_loop(input_queue, output_queue):
    while True:
        if not input_queue.empty():
            message_data = input_queue.get()
            radio.openWritingPipe(pipe[0])
            radio.write(bytes(message_data, 'utf-8'))
            print(f"Data sent: {message_data}")
            radio.openReadingPipe(1, pipe[1])
            radio.startListening()
            time.sleep(1)

        if radio.available():
            received_data = 0
            radio.read(received_data)
            print(f"Received: {received_data}")
            output_queue.put(received_data)
            time.sleep(1)
```

Рисунок 3.22 – Код комунікації за допомогою модулю NRF24

3.6 Реалізація наближення та зчитування конфігурацій

Для комфорту використання програмного забезпечення, необхідно розробити функцію наближення зображення. Вона повинна відповідати за зумування (масштабування) зображення. Для цього використовуються такі параметри, як коефіцієнт масштабування factor та центр зумування zoom_center.

У разі, якщо factor не дорівнює 1 (тобто потрібно здійснити зум), обчислюються нові розміри зображення після зумування. Нові розміри визначаються як відношення оригінальних розмірів до factor [33].

Далі обчислюється область зображення (roi), яку слід взяти для подальшого зумування. Ця область визначається так, щоб центр зумування

zoom_center залишався посередині нового зображення, і при цьому область не виходила за межі оригінального зображення.

Зображення в цій області збільшується за допомогою функції cv2.resize. Якщо factor дорівнює 1 (тобто зумування не потрібно), то функція просто повертає оригінальне зображення без змін.

Загалом, ця функція дозволяє реалізувати зумування зображення відносно вказаного центру та коефіцієнта масштабування. Код функції наближення представлено на рисунку 3.23.

```
def zoom(image, factor, zoom_center):
    if factor != 1:
        height, width, _ = image.shape

        new_width = int(width / factor)
        new_height = int(height / factor)

        roi_x = max(0, int(zoom_center[0] - new_width / 2))
        roi_y = max(0, int(zoom_center[1] - new_height / 2))
        roi_width = min(new_width, width - roi_x)
        roi_height = min(new_height, height - roi_y)

        zoomed_roi = cv2.resize(image[roi_y:roi_y + roi_height, roi_x:roi_x +
roi_width], (width, height))
        return zoomed_roi

    return image
```

Рисунок 3.23 – Функція масштабування зображення

Також, важливою складовою програми є можливість настроювання конфігурації під різні системи, камери та типи підключення. Для цього необхідно створити функцію парсингу конфігураційного файлу. Вона використовує бібліотеку configparser для парсингу конфігураційного файлу, зазвичай із розширенням .ini. Файл конфігурації передається як аргумент file_path, і за замовчуванням вказано '.\\config.ini'.

Після парсингу конфігурації, отримані значення зберігаються у змінній config. Далі змінні з конфігурації, такі як MAIN_CAMERA, CE_PIN, CSN_PIN, NRF_CHANNEL, PIPE_RX, PIPE_TX, PAYLOAD_SIZE, і DRONE_IP, записуються в середовище виконання (os.environ.update). Це робиться з метою

зручності та доступності цих значень для інших частин програми, які можуть використовувати середовище виконання для звертання до цих змінних.

Отже, функція допомагає ініціалізувати ряд параметрів програми, витягуючи їх з конфігураційного файлу та розміщуючи їх у змінних середовища для подальшого використання. Приклад коду зображено на рисунку 3.24.

```
def parse_config(file_path='.\\config.ini'):
    config = configparser.ConfigParser()
    config.read(file_path)

    os.environ.update({
        'MAIN_CAMERA': config.get('CAMERA', 'MAIN_CAMERA'),
        'CE_PIN': config.get('NRF_COMM', 'CE_PIN'),
        'CSN_PIN': config.get('NRF_COMM', 'CSN_PIN'),
        'NRF_CHANNEL': config.get('NRF_COMM', 'NRF_CHANNEL'),
        'PIPE_RX': config.get('NRF_COMM', 'PIPE_RX'),
        'PIPE_TX': config.get('NRF_COMM', 'PIPE_TX'),
        'PAYLOAD_SIZE': config.get('NRF_COMM', 'PAYLOAD_SIZE'),
        'DRONE_IP': config.get('DRONE_CONTROL', 'DRONE_IP')
    })
```

Рисунок 3.24 – Функція парсингу конфігураційного файлу

3.7 Розробка основного циклу програми

Основний цикл програми повинен обробляти усю інформацію із методів, які були розроблені до цього. Головна його задача, це ініціація потоків, захват відеопотоку, його обробка та виведення результатів усіх використаних функцій та методів.

Код основного циклу починається із ініціалізації змінних, які будуть використовуватися у ході виконання програми. До цих змінних входять також шлях до моделі, лінійні розміри об’єкту відстеження, шлях до відеопотоку та черги, які використовуються в потоках. Код на рисунку 3.25.

```
parse_config()

# Initialize variables
object_lost = False
video_file = ".\\test.mp4"
cap = cv2.VideoCapture(video_file)
object_image_size = 40
cursor_x, cursor_y = -1, -1
tracking = False
bbox = None
ml_detection = False
start_flight = False
stop_flight = False
zoom_factor = 1.0
old_results = {}
real_object_width = 6
fps_limiter = 15
tracking_ok = False

# Initialize queues
img_buffer = queue.Queue()
ml_results = queue.Queue()
drone_control_queue = queue.Queue()
return_drone_info = queue.Queue()
input_comm_queue = queue.Queue()
output_comm_queue = queue.Queue()

# Load labels and model for neural detection
labels = load_labels()
model_path = 'Tensorflow/workspace/models/my_ssd_mobnet/export/saved_model/'
model = load_model(model_path)
```

Рисунок 3.25 – Ініціалізація змінних

Далі йдуть методи окремих потоків. А саме потік для використання навченої моделі, потік для комунікації за допомогою радіо модулю та потік для управління дроном.

Метод обробки зображення за допомогою нейронної мережі постійно обробляє зображення, використовуючи попередньо завантажену модель машинного навчання. Він включає невелику затримку у безкінечному циклі для регулювання швидкості виконання. Під час кожної ітерації перевіряється наявність зображення у черзі img_buffer. Якщо черга порожня, цикл продовжується. Якщо зображення присутнє, воно витягується з черги разом з розміром екрану.

Зображення піддається попередній обробці, а потім використовується для виявлення об'єктів за допомогою моделі. Об'єкти з ймовірністю менше 0.6

відсіюються. Результати обробки, такі як координати та класи об'єктів, зберігаються у словнику results_dict. Цей словник поміщається в чергу ml_results для подальшого використання в інших частинах програми [34].

Методи для комунікації та управління представляють собою виклики функцій, які були розглянуті раніше. Код потоків на рисунку 3.26.

```
def ml_detection_process():
    while True:
        time.sleep(0.025)
        if img_buffer.empty():
            continue
        else:
            ml_frame, screen_size = img_buffer.get()
            img = preprocess_image(ml_frame)
            res = detect_objects(model, img, 0.6)
            results_dict = {}

            for idx, result in enumerate(res):
                ymin, xmin, ymax, xmax = result['bounding_box']
                xmin = int(max(1, xmin * screen_size[1]))
                xmax = int(min(screen_size[1], xmax * screen_size[1]))
                ymin = int(max(1, ymin * screen_size[0]))
                ymax = int(min(screen_size[0], ymax * screen_size[0]))

                label = labels[result['class_id']]
                key = f'{label}_{idx}'
                results = [ymin, xmin, ymax, xmax]
                results_dict[key] = results
            ml_results.put(results_dict)


# Function for drone control thread
def drone_thread():
    drone_auto_control_thread(drone_control_queue, return_drone_info)


# Function for communication thread
def communication_thread():
    main_comm_loop(input_comm_queue, output_comm_queue)
```

Рисунок 3.26 – Методи потоків

Далі необхідно організувати функцію зворотнього захвату миші, яка буде використовуватися для тестування роботи програмного забезпечення. У функції є локальні та глобальні змінні, такі як cursor_x, cursor_y, tracking, та bbox. Під час спрацювання події миші, функція оновлює положення курсора, малює прямокутник навколо об'єкта на кадрі та встановлює область відстеження (bbox). Якщо натиснута ліва кнопка миші, встановлюється

початкове положення курсора, вмикається відстеження (tracking), і виводиться повідомлення "Start Tracking". Код функції зображено на рисунку 3.27.

```
def mouse_callback(event, x, y, flags, param):
    global cursor_x, cursor_y, tracking, bbox
    cv2.rectangle(frame, (
    x - int(object_image_size / 2), y - int(object_image_size / 2),
object_image_size, object_image_size),
                  (0, 255, 0), 2)
    if event == cv2.EVENT_LBUTTONDOWN:
        cursor_x, cursor_y = x, y
        bbox = (x - int(object_image_size / 2), y - int(object_image_size /
2), object_image_size, object_image_size)
        tracking = True
        print("Start Tracking")
```

Рисунок 3.27 – Функція захвату подій миші

Далі необхідно організувати із методів окремі потоки та запустити їх. Після цього починається основний цикл програми.

У циклі відбувається читання кадрів від камери (cap.read()), обробка відстеження та подій клавіш (tracking_process()), обробка подій клавіш для зміни параметрів відстеження та управління дроном.

Ключова частина коду включає обробку натискання клавіш для зміни параметрів відстеження та управління дроном. Наприклад, клавіші 'w' та 's' відповідають за збільшення та зменшення масштабу відео, 'q' та 'a' – за збільшення та зменшення розмірів об'єкта для відстеження, 'y' – для включення машинного навчання для виявлення об'єктів.

Також в коді є обробка результатів машинного навчання (ml_results) та їх відображення на відеокадрі. Якщо включено виявлення машинного навчання (ml_detection), то малюються прямокутники та назви виявлених об'єктів.

Наприкінці циклу виводиться оброблений відеокадр у вікні "Object Tracking", а також виконується перевірка на натискання клавіші ']'. Якщо вона натиснута, цикл завершується, тобто програма завершує свою роботу. Код основного циклу представлено на рисунку 3.28.

```
while True:
    timer = cv2.getTickCount()
    ret, frame = cap.read()
    screen_size = frame.shape
    original_frame = frame.copy()
    if not ret:
        break
    frame, bbox1, tracking, object_lost, tracking_ok, target_center =
tracking_process(frame, tracking, tracking_ok,
bbox, object_lost)
    if os.getenv("NRF_ON"):
        key_quit = cv2.waitKey(fps_limiter)
        input_comm_queue.put("Debug_Info")
        key = output_comm_queue.get()
    else:
        key = cv2.waitKey(fps_limiter)
        key_quit = cv2.waitKey(fps_limiter)
    zoom_point = target_center
    if key == ord('w'):
        zoom_factor = zoom_factor + 0.1 if zoom_factor < 5.0 else zoom_factor
    elif key == ord('s'):
        zoom_factor = zoom_factor - 0.1 if zoom_factor > 1.0 else zoom_factor
    elif key == ord('q'):
        object_image_size = object_image_size + 10 if object_image_size < 100 else
object_image_size
    elif key == ord('a'):
        object_image_size = object_image_size - 10 if object_image_size > 10 else
object_image_size
    elif key == ord('y'):
        ml_detection = True
    elif key == ord('c'):
        tracking = False
        tracking_ok = False
    elif key == ord('t'):
        ml_detection = False
    elif key == ord('g'):
        start_flight = True
        stop_flight = False
    elif key == ord('h'):
        stop_flight = True
    if ml_results.empty():
        img_buffer.put([original_frame, screen_size])
    if ml_detection:
        if not ml_results.empty():
            results_dict = ml_results.get()
            old_results = results_dict.copy()
        for key, results in old_results.items():
            ymin, xmin, ymax, xmax = results
            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 3)
            cv2.putText(frame, key, (xmin, min(ymax, screen_size[0] - 20)),
                        cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 2,
cv2.LINE_AA)
        old_results = old_results.copy()
    frame = DrawFunctions(frame).draw_aim()
    frame = zoom(frame, zoom_factor, zoom_point)
    frame, target_distance = distance_calculator(frame, bbox1, real_object_width)
    if start_flight and tracking_ok:
        if not return_drone_info.empty():
            stop_flight = return_drone_info.get()
        else:
            drone_control_queue.put([target_center, screen_size, start_flight,
stop_flight, target_distance])
    frame = DrawFunctions(frame).draw_control_keys()
    frame = DrawFunctions(frame).debug_info(cursor_x, cursor_y)
    cv2.imshow('Object Tracking', frame)
```

Рисунок 3.28 – Основний цикл програми

3.8 Висновки до третього розділу

У третьому розділі було розглянуто процес розробки програмного забезпечення системи відеоспостереження. Детально розглянуто кожен із методів, які використовуються у програмі.

Особлива увага приділяється процесу інтеграції навченої моделі у програмний код. Розглядаються обов’язкові дії, які необхідні для стабільної роботи системи виявлення.

Проведено аналіз трекерів бібліотеки OpenCV. Зроблено порівняння стабільності трьох різних трекерів. Проведено дослідження у однакових умовах, визначено максимальний час, який трекери стабільно відстежували об’єкт. В результаті по отриманим даним зроблено висновок, що використання трекеру CSRT є найдоцільнішим з точки зору навантаження на систему та стабільності відстеження.

Розглянуто основний цикл програми, який включає в себе розділення на потоки різних функцій для підвищення оптимізації програмного забезпечення. Такі функції, як керування польотним контролером БПЛА, передача даних через радіомодуль та застосування моделі для виявлення об’єктів винесені в окремі потоки та не впливають на якість та кількість кадрів при передачі зображення, що позитивно відгукається на зручності використання розробленої системи відеоспостереження.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Проведення експериментального дослідження розробленого програмного забезпечення

Для дослідження програмного забезпечення, необхідно організувати підключення до польотного контролеру. Для симуляції польотного контролеру, можна використовувати симулятор SITL. Для його запуску, необхідно мати персональний комп’ютер із системою Linux.

Для того, щоб запустити SITL на операційній системі Windows, можна використовувати емулятор терміналу систем Unix Cygwin64. Завдяки цьому, є можливість запуску симулятору контролера та дрону в цілому. Крім зазначених інструментів необхідно також встановити проксі сервер MAVProxy. Він забезпечить зв’язок між емулятором Лінуксу із запущеним симулятором та системою Windows [35].

Після цього можна починати дослідження розробленого програмного забезпечення.

Щоб перевірити ефективність виявлення об’єктів, та загалом протестувати систему, було обрано кілька відеозаписів на основі яких буде проводитися тестування.

Насамперед слід дослідити функції трекінгу об’єктів. Для цього необхідно обрати мишею необхідний об’єкт на зображенні та натиснути ліву клавішу миші. Для зміни об’єкту, достатньо вибрати тим же чином інший об’єкт. А для припинення відстеження натиснути клавішу F. Результати тестування на рисунку 4.1 та 4.2.

Рисунок 4.1 – Дослідження роботи трекеру

Рисунок 4.2 – Дослідження роботи трекеру

Наступним етапом, слід протестувати ввімкнення та роботу функції виявлення об’єктів за допомогою нейронної мережі. Для цього слід використовувати клавіши Y та T для ввімкнення та вимкнення режиму відповідно.

Результати дослідження вказано на рисунку 4.3.

Рисунок 4.3 – Дослідження процесу виявлення об’єктів

Як видно із результатів експериментів, не всі об’єкти на кадрі були знайдені нейромережею. Це свідчить про недостатню навчаність моделі. Для виправлення цієї ситуації слід збільшити кількість даних для навчання та кількість кроків самого навчання.

Далі слід протестувати зум до центру екрану та до об’єктів під час відслідковування. Для цього використовуються клавіши W та S для збільшення та зменшення відповідно. Результати тестування на рисунках 4.4 та 4.5.

Рисунок 4.4 – Зум без відстеження

Рисунок 4.5 – Зум із відстеженням

Як видно із результатів дослідження, при збільшенні зображення без трекінгу об’єкта, наближається центр зображення. А при відстеженні, центр збільшення зосереджено на самому об’єкті.

Далі протестуємо навігацію дрону при відстеженні об’єкта. Для цього використаємо симулятор SITL. Для початку руху дрону у напрямку об’єкта використовується клавіша G. Результати тестування на рисунках 4.6 та 4.7.

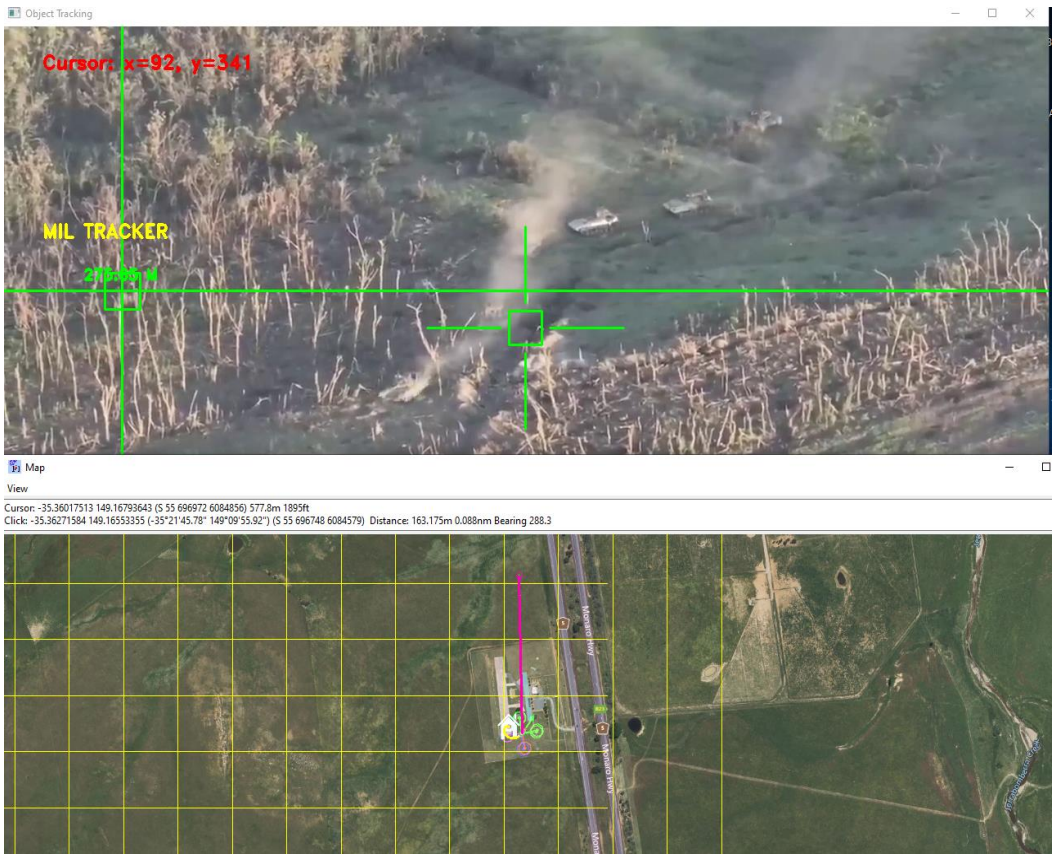

Рисунок 4.6 – Дослідження процесу контролю дрона

Рисунок 4.7 – Дослідження процесу контролю дрона

Як видно із результатів дослідження, враховуючи положення об’єкту на зображенні, розраховується нова точка, у якій знаходиться ціль. Для цього використовується кут огляду камери, який відомий із документації та розрахована відстань. Також, при потребі, дрон можна зупинити у будь-який момент.

Для виводу інформації при використанні дрона, достатньо під’єднати аналоговий вихід дисплею RaspberryPi до передавача відеосигналу, який встановлено на БПЛА.

За результатами проведених експериментів можна зробити висновок, що розроблене програмне забезпечення має ряд переваг, що важливі у системах відеоспостереження. Також дослідження показали, що існують деякі недоліки програмного забезпечення, особливо при використанні навченої моделі нейронної мережі.

4.2 Висновки до четвертого розділу

Проведений аналіз розробленого програмного забезпечення показав, що використання системи відеоспостереження надає змогу вести стабільне відстеження об’єктів навіть на великих відстанях. Стабільність роботи трекерів дозволяє передавати дані на польотний контролер та змінювати координати цілі польоту у реальному часі.

Дослідження поведінки інтегрованої навчена модель виявлення об’єктів показало, що модель потребує покращення. А саме збільшення кількості навчальних даних та кількості кроків навчання.

Системи управління БПЛА та функція розрахунку координат цілі забезпечують точний контроль польотного контролеру у реальному часі та можуть використовуватися для слідування за об’єктом.

ВИСНОВКИ

Системи відеонагляду є важливою складовою сучасного життя, як у громадських, так і у приватних сферах. Вони використовуються для забезпечення безпеки, відслідковування подій, контролю доступу, а також для управління та вирішення різних завдань в реальному часі. Технології відеонагляду постійно розвиваються, забезпечуючи більшу точність, швидкість і функціональність систем.

Згідно поставленої задачі, було спроектовано та реалізовано програмне забезпечення, яке надає можливість проводити відеоспостереження використовуючи інструменти комп'ютерного зору. Завдяки зручним бібліотекам TensorFlow та OpenCV, вдалося реалізувати просту у використанні програму, яка може бути інтегрована в різні системи нагляду.

Для збільшення ефективності відеоспостереження, також навчено нейронну мережу на власних даних, які були зібрані у відкритому доступі. Навчена модель показала непогані результати виявлення об'єктів. Хоча для повноцінної інтеграції системи, слід збільшити кількість даних для навчання і також збільшити кількість кроків при процесі навчання.

Додатково, реалізовано корисні функції, такі як трекінг об'єктів у реальному часі із зручним керуванням зміни об'єкту відстеження та припинення відстеження. Додано зум, який адаптується під ситуацію та процес спостереження.

Для інтеграції програмного забезпечення у системи із використанням дронів та БПЛА, створено додатковий функціонал, який допомагає контролювати політ дрону та визначати координати цілі.

Для передачі даних організовано також окремий блок для використання радіомодуля NRF24. Завдяки чому можна обмінюватися інформацією між різними блоками системи та головним центром.

Функції, які навантажують систему, були окремо розділені на незалежні потоки, які весь час обмінюються інформацією із основним циклом програми. Це дозволяє забезпечити стабільну частоту кадрів при великому навантажені системи процесом виявлення об’єктів нейромережею.

Недоліками розробленого програмного забезпечення є не достатньо точна модель виявлення об’єктів та розрахунок відстані, так як використання однієї камери без стереокамер або далекомірів не дозволяє точно та зручно розрахувати відстань до об’єкту. Також, необхідно покращити дизайн інтерфейсу для більш комфортного користування.

Розроблене програмне забезпечення може використовуватися у системах відеоспостереження великих за площею об’єктів, таких як склади, аграрні ділянки та інші.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 3008 – 2015. Державний стандарт України. Документація, звіти у сфері науки. Структура і правила оформлення – Чинний від 1 липня 2017 р.

2. Методичні вказівки до підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньо-професійних програм: «Автоматизоване управління технологічними процесами», «Комп'ютерно-інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та роботехнічні системи» / упоряд. : І.Ш. Невлюдов, Р.В. Артюх, В.В. Безкоровайний, Н.П. Демська, В.В. Євсеєв, О.І. Филипенко, О. М. Цимбал. Харків : ХНУРЕ, 2021. 55 с. (дата звернення: 18.10.2023).

3. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології»: довід. / І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, Г.В. Пономарьова. К.: Київ-56, пр. Космонавта Комарова, 1, 2016. 320 с. (дата звернення: 18.10.2023).

4. Положення про протидію академічному плагіату в ХНУРЕ / nure.ua. URL: https://nure.ua/wpcontent/uploads/Main_Docs_NURE/polozhennjaproakademichnu-dobrochesnist.pdf (дата звернення: 18.10.2023).

5. Anthony C. Caputo. Digital Video Surveillance and Security // Butterworth-Heinemann. – 2014 – P. 210 – 224. ISBN 978-0124200425.

6. Ross Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems // Wiley. – 2020 – P. 550 – 584. ISBN 978-1119642787.

7. Yunqian Ma. Intelligent Video Surveillance Systems and Technology // CRC Press. – 2009 – P. 234 – 256. ISBN 978-1439813287.

8. P. K. Garg. Unmanned Aerial Vehicles: An Introduction // Wiley. – 2012 – P. 144 – 181. ISBN 978-1683927099.

9. Paul G. Fahlstrom. Introduction to UAV Systems // Butterworth-Heinemann. – 2014 – P. 214 – 252. ISBN 978-1119978664.

10. Amita Kapoor. Deep Learning with TensorFlow and Keras: Build and deploy supervised, unsupervised, deep, and reinforcement learning models // Packt Publishing. – 2022 – P. 24 – 74. ISBN 978-1803232911.

11. Вивчаємо Python: [Електронне видання] / Автор.: Марк Лутц – Діалектика, 2020. – 103 с. (дата звернення: 13.10.2023).

12. Richard Szeliski. Computer Vision: Algorithms // Springer. – 2022 – P. 110 – 124. ISBN 978-0-07-057913-2.

13. Thushan Ganegedara. TensorFlow in Action // Manning. – 2022 – P. 323 – 354. ISBN 978-1617298349.

14. WEKA. // Сайт WEKA. URL: https://www.weka.io/. (дата звернення 01.09.2023).

15. TensorFlow API 2.13.0. // Сайт TensorFlow. URL: https://www.tensorflow.org/api_docs/python/tf. (дата звернення 10.09.2023).

16. DroneKit Docs // Сайт DroneKit. URL: https://dronekit.io/. (дата звернення 15.09.2023).

17. TensorFlow 2 Detection Model Zoo // Сайт GitHub. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. (дата звернення 16.09.2023).

18. DroneKit GitHub Repo. // Сайт GitHub. URL: https://github.com/dronekit/dronekit-python. (дата звернення 15.09.2023).

19. Intel Articles // Сайт Intel. URL: www.intel.com/content/www/us/en/developer/articles/ (дата звернення 03.10.2023)

20. KC Tung. TensorFlow 2 Pocket Reference: Building and Deploying Machine Learning // O'Reilly Media. – 2021 – P. 155 – 175. ISBN 978-1492089186.

21. Antonio Gulli. Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API // Packt Publishing. – 2019 – P. 531 – 565. ISBN 978-1838823412.

22. Run AI. // Сайт Run AI. URL: https://www.run.ai/. (дата звернення 22.10.2023).

23. TechTarget. // Сайт TechTarget. URL: https://www.techtarget.com/. (дата звернення 23.10.2023).

24. Google Colaboratory // Сайт Google URL: https://colab.google/. (дата звернення 24.10.2023)

25. Jupyter // Сайт Jupiter URL: https://jupyter.org/. (дата звернення 26.10.2023)

26. Machine Learning Made Easy // Сайт MathWorks. URL: https://www.mathworks.com/matlabcentral/fileexchange/50232-machinelearning-made-easy. (дата звернення 01.11.2023)

27. Jupyter // Сайт Jupiter URL: https://jupyter.org/. (дата звернення 26.10.2023)

28. Цимбал О.М., Бронніков А.І. Системи адаптації роботів і технологія OpenCV: Навчальний посібник – Харків: ХНУРЕ, 2019. – 148 с.

29. Nevliudov, I., Tsymbal, O., Bronnikov, A. (2022). Fuzzy Decision-Making for Intelligent Robotic System. In: Sergiyenko, O. (eds) Optoelectronic Devices in Robotic Systems. Springer, Cham. https://doi.org/10.1007/978-3-031-09791-1_9.

30. Мілько Д. В., Цимбал О. М. Дослідження програмного методу визначення відстані до об’єкту за допомогою параметрів камери. ADED-2023 випуск 2. М-во освіти і науки України; Харків. нац. ун-т радіоелектроніки.– Харків: [Електронне видання], 2023.

31. NRF24 2.0.0 // Сайт PYPI URL: https://pypi.org/project/nrf24/. (дата звернення 30.10.2023)

32. Reinhard Klette. Concise Computer Vision: An Introduction into Theory and Algorithms // Springer. – 2014 – P. 325 – 344. ISBN 978-1447163190.

33. ArduPilot SITL Simulator // Сайт ArduPilot. URL: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. (дата звернення 11.11.2023)

34. SITL Simulator GitHub // Сайт GitHub. URL: https://github.com/ArduPilot/ardupilot_wiki/blob/master/dev/source/docs/sitl-simulator-software-in-the-loop.rst. (дата звернення 12.11.2023).

35. Cygwin // Сайт Cygwin. URL: https://www.cygwin.com/. (дата звернення 13.11.2023)