НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Метод відстеження руху об'єкта з керуванням положення камери
Виконав: студент 4 курсу, групи ІО-02
(шифр групи)

Бабич Валерій Юрійович
(прізвище, ім’я, по батькові) (підпис)

Керівник проф., д.ф-м.н., с.н.с. Гордієнко Ю. Г.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) с.в. Виноградов Ю. М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент ас. каф. СПіСКС д.ф. Сергієнко П. А.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Бабича Валерія Юрійовича

1. Тема проєкту Метод відстеження руху об'єкта з керуванням положення камери керівник проєкту проф., д.ф-м.н., с.н.с. Гордієнко Юрій Григорович,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27.05 2024 року № 2112-с
2. Термін здачі студентом закінченого проєкту *20 травня 2021 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні та статистичні дані, патенти на винахід*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Опис предметної області, дослідження методики відстеження руху об’єкту з керуванням положення камери безпілотного літального апарату*
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
схема роботи методу для відстеження руху (схема структурна), діаграма класів (схема функціональна), алгоритм використання (схема принципова).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	с.в. Виноградов Ю. М.		

7. Дата видачі завдання 01.04.2024

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*06.04.2024-07.04.2024*	
2.	*Вивчення та аналіз завдання*	*08.12.2021-14.04.2024*	
3.	*Вибір технології для розробки*	*15.04.2024-21.04.2024*	
4.	*Розробка алгоритмів для задачі згідно теми проєкту*	*22.04.2024-28.04.2024*	
5.	*Програмна реалізація системи*	*29.04.2024-05.05.2024*	
6.	*Оформлення пояснювальної записки*	*06.05.2024-12.05.2024*	
7.	*Захист програмного продукту*	*30.05.2024*	
8.	*Передзахист*	*12.06.2024*	
9.	*Захист*	*21.06.2024*	

Студент _____ БАБИЧ Валерій
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ ГОРДІЄНКО Юрій
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

В бакалаврському дипломному проєкті реалізовано алгоритм відстеження руху об’єкту з керуванням положення камери, призначений для фокусування безпілотного літального апарата на цілі,який базується на використанні глибинного машинного навчання і нейронних мереж.

Програма дозволяє визначити об’єкт, а також відцентрувати його так щоб від постійно був посередині нашої камери на такій відстані, щоб можна було розгледіти ціль в повній мірі. Програмний продукт був створений на мові програмування Python, з використанням UnrealEngine, як симулятор польотів дрону.

Для створення симуляції безпілотного літального апарату використовується бібліотка для мови Python – AirSim, а для глибинного навчання Yolo8 та OpenCV.

ABSTRACT

In the bachelor's thesis project, we implemented an object tracking algorithm with camera position control designed to focus an unmanned aerial vehicle on a target, based on deep machine learning and neural networks.

The program allows us to detect an object and center it so that it is constantly in the middle of our camera at a distance that allows us to see the target in full. The software product was created in the Python programming language using UnrealEngine as a drone flight simulator.

To create a simulation of an unmanned aerial vehicle, the AirSim library for Python was used, and Yolo8 and OpenCV were used for deep learning.

Довідки	Формат	Значення	Найменування	Кіл. листів	№ екземпляра	Додаток
			Документація загальна			
			Заново розроблена			
	А4	ІАЛЦ.467200.002 ТЗ	Метод відстеження руху	4		
			об'єкта з керуванням			
			положення камери			
			Технічне завдання			
	А4	ІАЛЦ.467200.003 ПЗ	Метод відстеження руху	55		
			об'єкта з керуванням			
			положення камери			
			Пояснювальна записка			
	А4	ІАЛЦ.467200.004 Д1	Схема роботи методу для відтсеження руху об’єкту	1		
			(Схема структурна)			
	А4	ІАЛЦ.467200.005 Д2	Діаграма класів	1		
			(Схема функціональна)			
	А4	ІАЛЦ.467200.006 Д3	Алгоритм використання	1		
			(Схема принципова)			
	А4	ІАЛЦ.467200.007 Д4	Текст програмного коду	17		

					ІАЛЦ.467200.001 ОА			
Зм.	Лист.	№ докум.	Підпис	Дата				
Розробила		Бабич В.Ю.			**Метод відстеження руху об'єкта з керуванням положення камери** Опис альбому	Літ.	Аркуш	Аркушів
Перевірив		Гордієнко Ю.Г.					1	1
						КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-02		
Н. Контр.		Виноградов Ю.М.						
Затвердив								

ТЕХНІЧНЕ ЗАВДАННЯ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Метод відстеження руху об'єкта з керуванням положення камери»

Київ – 2024 р.

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Бабич В.Ю.*			**Метод відстеження руху об'єкта з керуванням положення камери**	*Літ.*	*Аркуш*	*Аркушів*
Перевірив		*Гордієнко Ю.Г..*					*1*	*4*
Реценз.					***Технічне завдання***	КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-02		
Н. Контр.		*Виноградов Ю.М.*						
Затв.								

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Проєкт охоплює дослідження методу відстеження руху об’єкта з керуванням положення камери. Розглянемо і перспективи даної роботи. Можливостей розвитку цієї галузі розробки безліч, але в нинішніх реаліяї я виділяю основну галузь і це: військова, розвідувальна.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Основою для розробки стало завдання бакалаврського проєкту за освітньо-професійною програмою «Комп’ютерні системи та мережі» спеціальності 123 «Комп’ютерна інженерія», затверджене кафедрою Обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут імені Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою дипломного проєкту є розробка нових методів відстеження руху об’єкта з керуванням положенням камери, а також проектування відповідного пристрою. Це включає створення ефективних алгоритмів для аналізу зображень і відео, отриманих з камери дрону, з метою точного визначення положення та траєкторії об’єкта. Інтеграція цих алгоритмів з системою керування дозволить автоматично налаштовувати положення камери для оптимального стеження за об’єктом, забезпечуючи високу якість відеозйомки та точність відстеження. Проєкт передбачає розробку як апаратного, так і програмного забезпечення для безпілотного літального апарата (БПЛА). Апаратна частина включає вибір і налаштування необхідних

сенсорів, контролерів та механізмів керування камерою. Програмна частина зосереджена на розробці нейронних мереж для обробки зображень та алгоритмів для оцінки траєкторії руху об’єкта. Окрім цього, проєкт спрямований на забезпечення стабільної роботи системи в реальних умовах, включаючи різні погодні умови та рівні освітленості. Результати цієї розробки можуть бути застосовані в різних галузях, таких як відеоспостереження, кіноіндустрія, спортивні трансляції та рятувальні операції.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет, науково-технічна літератури.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

– Система повинна забезпечувати відстеження руху об’єкту, отриманого з камери дрону.

– Система повинна визначати положення об’єкту у просторі

– Система повинна тримати об’єкт на однаковій відстані по центру камери.

5.2 Вимоги до програмного забезпечення

– Версія Windows 11 або Windows 10.

– Unreal Engine 4.27

- AirSim

- VS Code

- Python 3.9

5.3 Вимоги до апаратної частини

– Відеокарта: 512MB ATI AMD Radeon Graphics.

– Процесор: AMD Ryzen 7.

– Оперативна пам’ять: 16 Гб.

– Материньська плата: ASUSTeK COMPUTER INC. X521IA (FP6)

– Пам’ять на жорсткому диску: 20 Гб.

6 ЕТАПИ РОЗРОБКИ

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*06.04.2024-07.04.2024*	
2.	*Вивчення та аналіз завдання*	*08.12.2021-14.04.2024*	
3.	*Вибір технології для розробки*	*15.04.2024-21.04.2024*	
4.	*Розробка алгоритмів для задачі згідно теми проєкту*	*22.04.2024-28.04.2024*	
5.	*Програмна реалізація системи*	*29.04.2024-05.05.2024*	
6.	*Оформлення пояснювальної записки*	*06.05.2024-12.05.2024*	
7.	*Захист програмного продукту*	*30.05.2024*	
8.	*Передзахист*	*12.06.2024*	
9.	*Захист*	*21.06.2024*	

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Метод монокулярної оцінки глибини»

Київ – 2024 р.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

БПЛА	Безпілотний літальний апарат
НМ	Нейронна мережа
ДС	Датасет
КВ	Канал відеопередачі
КЗ	Канал зв’язку
КУ	Канал управління
ПК	Персональний комп’ютер
API	Програмний інтерфейс додатку
ТР	Трекер
ДТ	Детектор

ВСТУП

Літальні безпілотні апарати – одна з найбільш динамічно розвиваються галузей сучасного суспільства. В комбінації з глибинним навчання та нейнонними мережами їх актуальність тільки збільшується.

Відстеження об’єкту за допомогою дрону(або іншого безпілотного літального апарату) може використовуватись у багатьох галузях. Цю технологію можуть використовувати як і фотографи для мирних цілей, так і військові для виконання бойових завдань. Використовувати краще за все такі апарати як FPV або DJTrello. Також відстежженя об’єкту спрощує використання дронів, але ускладнює їх розробку. Не забуваємо про те, що засоби РЕБ можуть з легкістю відключити КУ, КЗ чи КВ, або зовсім деактивувати наш безпілотний літальний апарат. Ми вже бачили використання дронів, в останній час більше у військовому сенсі, але це також цікавий об’єкт дослідження, який може зберегти багато життів союзників і погубити ворогів.

Перевага безпілотних літальних апаратів полягає у простоті управління, у можливості інтеграції з нейронними мережами, у відносно дешевому виробництві. PD-2, Raybird-3, RAM II UAV, SHARK, SkyKnight 2, БпАК-МП-1 «Spectator, QBOND888, UJ-22 Airborne, UJ-23 TOPAZ, АСУ-1 «Валькірія», «Бобер», БпАК «Гор», БпАК «Сич», Лелека-100, «Рубака», СКІФ, БпАК А1-СМ «Фурія», E-300 Enterprise це все українські бпла, якими ми повинні пишатися.

РОЗДІЛ 1

ОГЛЯД ТЕОРЕТИЧНИХ ЗАСАД ВІДСТЕЖЕННЯ ОБ’ЄКТІВ

1.1 Застосування відстеження об’єктів

1) Відеоспостереження за допомогою камер відеоспостереження

Відстеження об'єктів за допомогою камер відеоспостереження широко використовується для забезпечення безпеки в громадських місцях, таких як аеропорти, торгові центри та стадіони. Ця технологія допомагає виявляти підозрілі дії, відстежувати переміщення осіб і запобігати злочинам.У дослідженні, представленому в статті "Real-Time Object Tracking System Using OpenCV"[12], описується розробка системи відеоспостереження з використанням OpenCV для реального часу відстеження об'єктів. Лондонська поліція використовує систему відеоспостереження, яка дозволяє автоматично відстежувати підозрюваних у реальному часі, згідно статті авторів[12].

2.) Автономні транспортні засоби

У галузі автономного водіння відстеження об'єктів є критичним елементом для безпечної навігації автомобілів. Системи відстеження визначають місцезнаходження і рухи інших транспортних засобів, пішоходів та перешкод на дорозі, щоб автономні автомобілі могли адекватно реагувати на дорожні умови і уникати аварій.. Згідно статті авторі**в**[14], Tesla використовує комп'ютерний зір та трекери для відстеження об'єктів на дорозі, що дозволяє їхнім автомобілям автономно керувати.

3) Спортивні аналітики

У спортивних змаганнях системи відстеження об'єктів використовуються для аналізу руху гравців, м'ячів та інших об'єктів. Це дозволяє тренерам і аналітикам отримувати важливі дані щодо стратегій, продуктивності гравців та удосконалення тренувальних програм.. Існує система відстеження гравців в NBA, яка використовує камери для відстеження позицій гравців і м'яча в реальному часі.

4) Робототехніка

У промисловому виробництві та автоматизованих процесах відстеження об'єктів є важливою частиною оптимізації та автоматизації виробничих процесів. Роботи здатні взаємодіяти з навколишнім середовищем і виконувати завдання, що вимагають точності і координації, завдяки системам відстеження об'єктів, які надають їм інформацію про положення і стан об'єктів на виробничих лініях.

5) Системи доповненої реальності

У сфері доповненої реальності відстеження об'єктів дозволяє реалізувати інтерактивні ігри, освітні програми та візуалізацію даних. Технології AR використовуються для накладання віртуальних об'єктів на реальний світ, забезпечуючи нові можливості для інтерактивного навчання, розваг і роботи з даними.

6) Промислове виробництво

У виробничих процесах відстеження об'єктів використовується для моніторингу якості продукції, управління запасами і оптимізації виробничих процесів. Це дозволяє підприємствам знижувати витрати і

підвищувати ефективність виробництва, завдяки точному контролю і управлінню ресурсами.

Ці приклади демонструють, як технології відстеження об'єктів, засновані на OpenCV, застосовуються в реальному житті для різних цілей, від безпеки до робототехніки та доповненої реальності.

1.2 Проблематика відстеження об’єктів

Існують фактори, які заважають легко і коректно налаштувати нейронну мережу на відстеження того чи іншого об’єкту. Зокрема:

1) Шуми на зображенні

Шуми на зображенні, такі як мерехтіння, забруднення об'єктами або нечіткість, можуть значно ускладнювати процес відстеження об'єктів. Алгоритми відстеження повинні бути стійкими до таких шумів і здатними правильно ідентифікувати об'єкти навіть при незначних артефактах на зображенні.

2) Складна форма об'єкта

Об'єкти зі складною або непередбачуваною формою можуть викликати труднощі у точному відстеженні через велику варіативність їх вигляду на зображенні. Наприклад, складні та неоднорідні текстури чи нестандартні форми можуть призводити до помилок у визначенні положення об'єкта.

3) Освітлення

Зміни в умовах освітлення, такі як тіні, пряме сонячне світло або штучне освітлення, можуть робити об'єкти на зображенні менш видимими або змінювати їх вигляд. Алгоритми відстеження повинні адаптуватися до

різних умов освітлення для забезпечення стабільного і точного відстеження об'єктів.

4) Складна траєкторія руху об'єкта

Об'єкти, що рухаються по непрямоліній траєкторії або зі змінною швидкістю, представляють виклик для алгоритмів відстеження. Потрібно розробляти алгоритми, які можуть ефективно передбачати майбутнє положення об'єкта і забезпечувати плавне відстеження навіть при складних рухах.

5) Проблематика перетворення проєкції тривимірного зображення у двовимірне зображення

Перетворення тривимірного навколишнього середовища у двовимірне зображення для подальшого використання у відстеженні може призводити до втрати інформації або спотворення просторових відносин. Ефективні алгоритми відстеження повинні враховувати цю проблему для забезпечення точності і достовірності результатів.

6) Обробка зображення у реальному часі

Вимоги до обробки зображення у реальному часі ставлять великі виклики перед алгоритмами відстеження. Потрібно забезпечити достатню швидкість обробки кадрів відеопотоку для забезпечення миттєвої реакції системи на зміни в оточенні об'єкта.

7) Точність відстеження

Однією з головних проблем є досягнення високої точності відстеження об'єктів в реальному часі. В залежності від умов освітлення, швидкості руху об'єктів, шуму і інших факторів точність може значно

коливатися. Наприклад, об'єкти зі схожими характеристиками або подібні об'єкти можуть призводити до помилок у відстеженні.

8) Обробка великої кількості даних

За високої швидкості кадрів відеопотоку і великій кількості камер у системі відстеження, виникає необхідність в обробці величезного обсягу даних. Це може призводити до затримок у реальному часі, які не прийнятні для деяких застосувань, таких як автономні автомобілі або системи безпеки.

9) Взаємодія зі змінним оточенням

Оточення, в якому відбувається відстеження, може змінюватися через освітлення, погодні умови, рух інших об'єктів і так далі. Це створює виклики для алгоритмів відстеження, оскільки вони повинні адаптуватися до змінюючихся умов для збереження ефективності і точності.

10) Витрати на обчислення

Розробка і використання ефективних алгоритмів відстеження може вимагати значних обчислювальних ресурсів. Великі системи відстеження об'єктів, такі як великі мережі відеоспостереження або розгорнуті системи автономних транспортних засобів, потребують потужних обчислювальних платформ для обробки великої кількості даних в реальному часі.

11) Проблема обробки частково видимих об'єктів

У складних умовах, таких як затемнені або частково приховані об'єкти, можуть виникати труднощі у відстеженні. Алгоритми відстеження повинні вміти розпізнавати об'єкти навіть при обмеженому видимому полі, що ускладнює завдання.

Проблематика відстеження об'єктів включає в себе ряд технічних викликів, які вимагають розробки нових алгоритмів, використання потужних обчислювальних ресурсів і урахування специфічних умов відстеження для досягнення надійного і ефективного відстеження в реальному часі.

1.3 Визначення і пояснення основних понять дипломної роботи

1.3.1 Трекер

Трекер – це пристрій або програмне забезпечення, який визначає положення об’єкта. Відомий всім приклад – це GPS-трекер, який використовується для навігації. В даній роботі трекер використовується з поєднанні з камерою дрону і є основним для відстеження об’єкта. Також ТР не може коректно працювати без детектора.Принцип роботи трекерів об'єктів можна розділити на кілька ключових етапів:

1) Ініціалізація

Ініціалізація трекера включає вибір області на першому кадрі відеопотоку, яка містить об'єкт для відстеження. Це може бути зроблено вручну оператором або автоматично за допомогою детектора об'єктів.

Ручна ініціалізація: Оператор вибирає об'єкт на першому кадрі.

Автоматична ініціалізація: Алгоритм детекції визначає об'єкт, який потім передається трекеру для подальшого відстеження.

2) Екстракція ознак

Після ініціалізації трекер витягує ознаки, що характеризують об'єкт, для того щоб відстежувати його в наступних кадрах. Це можуть бути текстурні ознаки, колірні ознаки, градієнти або інші характерні риси об'єкта.

3) Прогнозування положення

Трекер прогнозує нове положення об'єкта в поточному кадрі на основі його положення в попередньому кадрі. Це робиться за допомогою різних моделей руху, які враховують швидкість і напрямок руху об'єкта.

4) Пошук об'єкта

Після прогнозування трекер виконує пошук об'єкта у вказаній області поточного кадру. Це може включати порівняння екстрагованих ознак з прогнозованими або використання алгоритмів кореляції.

5) Оновлення моделі

Після визначення нового положення об'єкта трекер оновлює свою модель, що включає оновлення ознак об'єкта та моделі руху для забезпечення точності в наступних кадрах.

1.3.2 Детектор

Детектор – це пристрій або програмне забезпечення, який виявляє наявність об’єкту, його рух та геомеричні характеристики. ДТ також має розповсюджене використання у повсякденному житті:

- детектор металу
- детектор руху(інфрачервоний або ультразвуковий)
- детектор диму і/чи газу(протипожарні системи)
- детектор радіохвиль

Принцип роботи детекторів базується на кількох основних етапах: передобробка зображення, екстракція ознак, класифікація та пост-обробка.

1) Передобробка зображення

На цьому етапі вхідне зображення обробляється для поліпшення якості та підготовки до подальших етапів аналізу. Це може включати масштабування, нормалізацію, фільтрацію шуму та інші операції для забезпечення чіткості і зменшення впливу сторонніх факторів.

2) Екстракція ознак

Після передобробки зображення, алгоритм екстрагує ключові ознаки, які характеризують об'єкти на зображенні. Ці ознаки можуть включати крапки, краї, текстури та інші важливі характеристики.

3) Класифікація

На цьому етапі використовуються алгоритми машинного навчання або глибокого навчання для класифікації об'єктів на основі екстрагованих ознак. Класифікатори можуть бути простими або складними, як глибокі нейронні мережі.

4) Пост-обробка

Після класифікації застосовується пост-обробка для покращення результатів детекції. Це може включати фільтрацію неправдивих спрацьовувань, згладжування результатів та інші методи для підвищення точності.

1.3.3 БПЛА(або дрон)

БПЛА[17] – це літальний апарат, який керується дистанційно за допомгою оператора або за допомогою вбудованих комп’ютерних ситем. Основна перевага дронів – це відстуність людини на борту, що в багатьох

випадках допомагає зберегти її життя. В сучасних реаліяї використання БПЛА набуває все більшої популярності, особливо у військовій сфері. Безпілотні літальні апарати (дрони) мають широкий спектр застосувань завдяки своїй мобільності, гнучкості та можливості оснащення різноманітними сенсорами та інструментами. Основні функціїї дронів у різних сферах:

1) Аерофотозйомка та відеозйомка

Дрони обладнані високоякісними камерами використовуються для зйомки фотографій та відео з висоти. Це дозволяє отримувати зображення з унікальних ракурсів, що є корисним для кіновиробництва, рекламної індустрії, архітектури та туризму. Дрони дозволяють знімати кінематографічні кадри з повітря, які раніше були доступні лише за допомогою вертольотів чи кранів.Аерозйомка використовується для створення вражаючих рекламних роликів і презентацій нерухомості.

2) Картографування та геодезія

Дрони оснащені GPS та іншими сенсорами можуть виконувати аерофотозйомку для створення карт та 3D-моделей місцевості. Вони використовуються для геодезичних робіт, моніторингу будівельних майданчиків, планування територій та оцінки земельних ділянок. Дрони забезпечують точні дані для геодезичних зйомок, знижуючи час і витрати порівняно з традиційними методами. Регулярне використання дронів дозволяє відстежувати прогрес будівництва та виявляти потенційні проблеми.

3) Сільське господарство

Дрони використовуються для моніторингу стану сільськогосподарських угідь, оцінки здоров'я рослин, внесення добрив та

засобів захисту рослин. Вони можуть оснащуватися різними сенсорами, включаючи мультиспектральні камери для точного землеробства. Дрони дозволяють отримувати детальну інформацію про стан посівів, виявляти хвороби та шкідників, аналізувати вологість ґрунту. Використання дронів для точкового розподілу добрив та засобів захисту рослин підвищує ефективність і знижує витрати.

4) Інспекція та моніторинг інфраструктури

Дрони застосовуються для інспекції важкодоступних об'єктів, таких як лінії електропередач, нафто- і газопроводи, вітряні турбіни, мости та будівлі. Вони можуть швидко і безпечно проводити огляди, виявляти пошкодження і забезпечувати моніторинг стану об'єктів. Інспекція ліній електропередач і вітряних турбін за допомогою дронів дозволяє виявляти проблеми та проводити планове обслуговування. Дрони використовуються для моніторингу стану будівель і мостів, забезпечуючи безпеку і тривалість експлуатації.

5) Пошук і рятувальні операції

Дрони використовуються в пошукових і рятувальних операціях для швидкого обстеження великих територій, виявлення постраждалих і доставки екстрених вантажів. Вони можуть оснащуватися тепловізорами та іншими сенсорами для роботи в складних умовах. Дрони з тепловізорами можуть ефективно знаходити зниклих людей в лісах, горах або під завалами. Використання дронів дозволяє оперативно доставляти медикаменти, їжу та інші необхідні ресурси до важкодоступних місць.

6) Охорона і безпека

Дрони використовуються для забезпечення безпеки і охорони об'єктів, патрулювання територій, моніторингу масових заходів і виявлення

підозрілої активності. Вони можуть оснащуватися камерами високої роздільної здатності, тепловізорами та іншими сенсорами. Дрони забезпечують патрулювання територій підприємств, виявлення несанкціонованих проникнень та підвищення рівня безпеки. Моніторинг подій за допомогою дронів допомагає забезпечити безпеку учасників і швидке реагування на інциденти.

7) Екологічний моніторинг

Дрони використовуються для моніторингу стану навколишнього середовища, оцінки впливу діяльності людини на природу, вивчення екосистем та контролю забруднення. Вони можуть збирати дані про стан водних ресурсів, якості повітря та змін клімату. Дрони збирають дані про якість води, стан берегової лінії і виявлення забруднень. Використання дронів для збору зразків повітря та вимірювання рівня забруднення дозволяє оперативно виявляти проблеми.

1.3.4 Нейронна мережа

Нейронна мережа[16] – це вид ком’ютерної системи, яка може вчитися та виконувати поставленне завдання. Базується на знаннях про людський мозок і нейронних звязках людини. В даному дипломному проєкті нейронні мережі використовуються для обробки зображень та відео в реальному часі, що і допоможе постійно відстежувати об’єкт за допомгою камери дрону.

Зокрема, навожу приклад роботи нейронної мережі, яка розпізнає на картинці різні об’єкти:

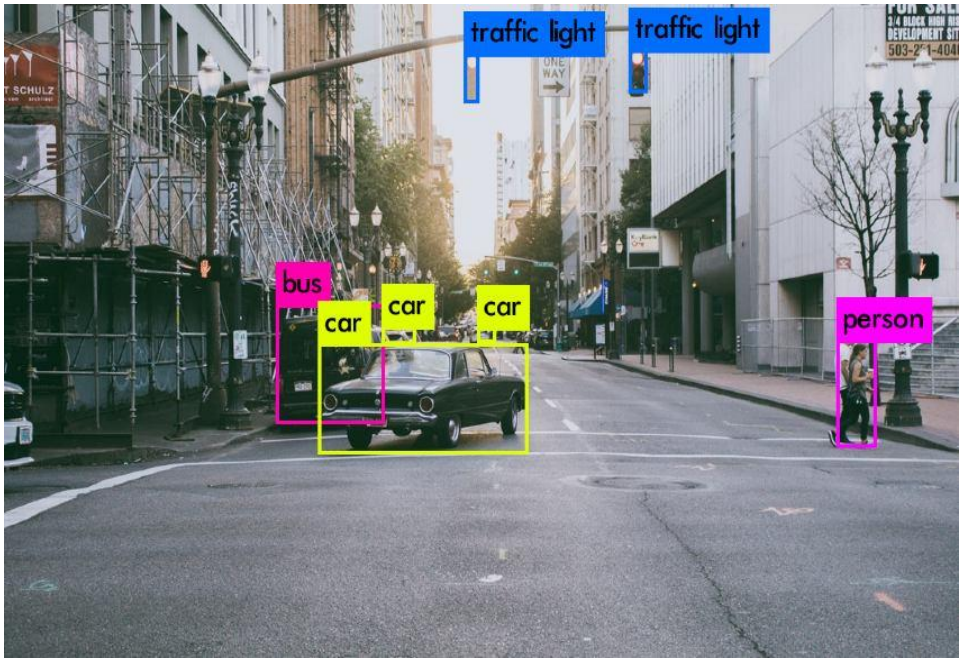


Рис.1.1 - Приклад роботи нейронних мереж

Нейронна мережа складається з вузлів (нейронів), які організовані в кілька шарів: вхідний шар приймає дані для обробки, сховані шари здійснюють обробку даних і виділяють особливості, вихідні шари повертають результат обробки, наприклад, класифікацію або регресійне значення.

Дані проходять через мережу шляхом передачі сигналів від одного нейрона до іншого. Кожен нейрон отримує вхідні дані, які обробляються за допомогою функції активації, а потім передаються наступному шару.

Навчання нейронної мережі відбувається через оптимізацію вагів, які визначають силу зв'язків між нейронами. Основні етапи навчання нейронних мереж включають: передача сигналу, обчислення помилки, зворотне поширення. Після обчислення градієнтів, ваги мережі оновлюються за допомогою оптимізаційних алгоритмів, таких як градієнтний спуск або його варіації.

За архітектурою нейронних мереж вони поділяються на:

- Повнозв'язна мережа (Fully Connected Network, FCN): Нейрони в кожному шарі підключені до всіх нейронів наступного шару. Повнозв'язна мережа широко використовується для класифікації зображень, де завдання полягає у визначенні до якого класу належить дане зображення. Наприклад, визначення, чи зображення містить кота або собаку. Також повнозв'язна мережа застосовуються для задач регресії, де необхідно прогнозувати певне числове значення на основі вхідних даних. Наприклад, прогнозування цін на нерухомість або передбачення погодних умов.

- Конволюційна нейронна мережа (Convolutional Neural Network, CNN): Використовується для обробки зображень і відео, застосовуючи операції зводок (convolutions), що дозволяють зменшити обчислювальні витрати. Конволюційні нейронні мережі використовуються для класифікації зображень. Вони автоматично виділяють важливі особливості з зображення на різних рівнях абстракції. Наприклад, система розпізнавання автомобілів на дорогах.

- Рекурентна нейронна мережа (Recurrent Neural Network, RNN): Використовується для обробки послідовних даних, таких як текст або часові ряди. Має внутрішні зв'язки, що дозволяють зберігати інформацію про попередні стани. Рекурентні нейронні мережі та їх варіанти (Long Short-Term Memory, LSTM; Gated Recurrent Units, GRU) використовуються для аналізу тексту, машинного перекладу, розпізнавання мови. Наприклад, переклад текстів з однієї мови на іншу, системи голосового помічника, такі як Siri або Google Assistant.

- Глибока нейронна мережа (Deep Neural Network, DNN): Мережа з великою кількістю схованих шарів, що дозволяє моделювати складні функції. Глибока нейронна мережа використовується для створення нових даних, таких як зображення, текст або аудіо, на основі навчальних даних.Наприклад, створення нових фотографій або картин, що імітують стиль художника, створення текстів, що імітують стиль певного автора.

Нейронні мережі є потужним інструментом для вирішення різноманітних задач в області штучного інтелекту. Вони здатні автоматично навчатися на великих обсягах даних, виділяти важливі ознаки і приймати складні рішення. Їх застосування варіюється від простих задач класифікації до складних генеративних моделей, що створюють нові дані. Успіх нейронних мереж в значній мірі зумовлений їх здатністю до самонавчання і масштабованості на великі обсяги даних.

1.3.5 Датасет

Датасет[15] - це набір даних, зібраних і організованих для аналізу або навчання моделей машинного навчання. Датасети можуть містити числові, текстові, зображення або інші типи даних і є основою для побудови моделей штучного інтелекту. Датасети можуть містити різні типи даних, такі як числові значення, текст, зображення, відео, аудіо та категоріальні дані. Дані можуть бути зібрані з різних джерел, включаючи бази даних, веб-скрапінг, сенсори, анкети, соціальні мережі та багато іншого. Багато датасетів, особливо ті, що використовуються для супервізованого навчання, включають розмітку або мітки, які вказують на правильні результати для кожного зразка. Зазвичай датасет розділяється на тренувальну, валідаційну

та тестову вибірки для оцінки ефективності моделі. Тренувальна вибірка використовується для навчання моделі, валідаційна - для налаштування гіперпараметрів, а тестова - для остаточної оцінки продуктивності. Якість даних у датасеті є критично важливою для успіху моделі. Вона включає точність, повноту, консистентність та актуальність даних. Існує багато загальновідомих датасетів, які широко використовуються для навчання та тестування моделей, наприклад, MNIST для розпізнавання рукописних цифр, CIFAR-10 для класифікації зображень, та IMDB для аналізу тональності текстів.

Підготовка датасету до використання включає етапи препроцесингу, такі як очищення даних, нормалізація, обробка пропущених значень та кодування категоріальних змінних.

ВИСНОВКИ ДО РОЗДІЛУ 1

Розглянуто застосування відстеження об’єктів. Виділена проблематика навчання та правильності роботи нейронних мереж для виконання поставленого завдання. Також було виділено та описано основі поняття, які знадобляться під час розробки та розуміння принципу дії відстеження об’єктів. Цієї мінімальної теоретичної бази буде досить, щоб перейти до наступного розділу з розумінням усього необхідного.

РОЗДІЛ 2

ОГЛЯД ІСНУЮЧИХ МЕТОДІВ

2.1 Узагальнення схема роботи ситеми

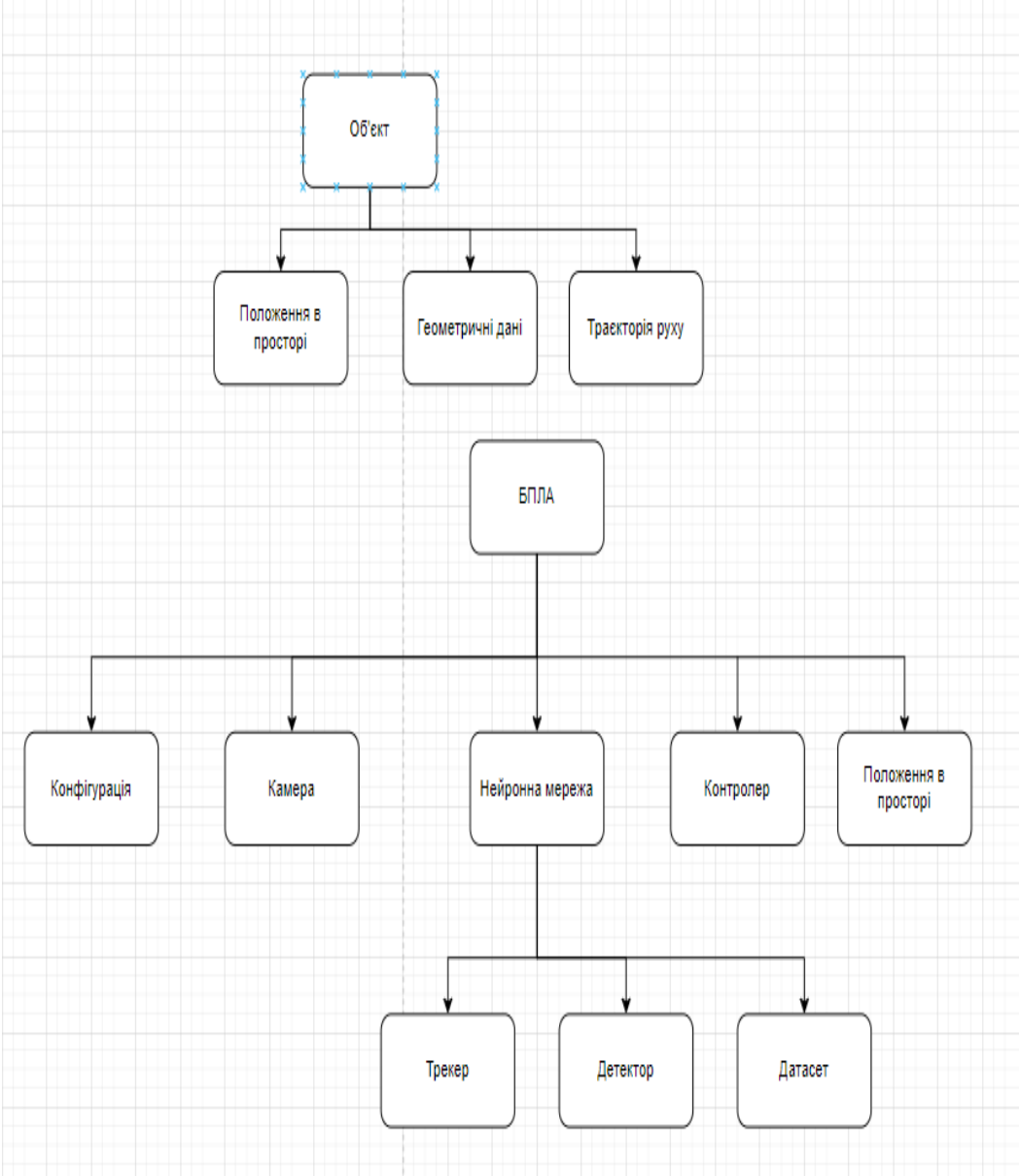


Рис.2.1 - Узагальнена схема роботи системи

2.2 Аналіз популярних трекерів OpenCV та їх порівняння

2.2.1 Аналіз BOOSTING трекеру

BOOSTING Tracker - це відстеження об'єктів у реальному часі, засноване на новій онлайн-версії алгоритму AdaBoost. Класифікатор використовує навколишній фон як негативні приклади на кроці оновлення, щоб уникнути проблеми дрейфу. Цей класифікатор потрібно тренувати під час виконання на позитивних і негативних прикладах об'єктів. Початкова обмежувальна рамка, надана користувачем (або іншим алгоритмом виявлення об'єктів), береться як позитивний приклад для об'єкта, а багато ділянок зображення за межами цієї рамки розглядаються як фон. На новому кадрі класифікатор запускається на кожному пікселі в околиці попереднього розташування і записується оцінка класифікатора. Нове розташування об'єкта - це те, де оцінка максимальна. Отже, тепер ми маємо ще один позитивний приклад для класифікатора. Коли надходить більше кадрів, класифікатор оновлюється цими додатковими даними. Алгоритм вже застарілий, і він працює нормально, але вагомих причин використовувати його немає, особливо коли є інші просунуті трекери (MIL, KCF), засновані на схожих принципах. Серед мінусів – це низька ефективність відстеження. Недостатньо достовірно визначає, коли відстеження завершилося невдачею.

Першоджерело: [9]

2.2.2 Аналіз MIL трекеру

Цей трекер схожий за ідеєю на описаний вище трекер BOOSTING. Велика різниця полягає в тому, що замість того, щоб розглядати тільки поточне розташування об'єкта як позитивний приклад, він шукає в невеликій околиці навколо поточного розташування, щоб згенерувати кілька потенційних позитивних прикладів. У MIL вказуються не позитивні і негативні приклади, а позитивні і негативні набори. Колекція зображень у

позитивному наборі - це не всі позитивні приклади. Натомість, лише одне зображення в позитивному наборі має бути позитивним прикладом!

Алгоритм має високу продуктивність. Він не дрейфує так сильно, як трекер BOOSTING, і робить розумну роботу при частковій оклюзії. MIL Tracker один із найкращих трекерів. Значним мінусом цього ТР є неможливість відновлення після повної оклюзії.

Першоджерело: [9]

2.2.3 Аналіз CSRT трекеру

У дискримінаційному кореляційному фільтрі з канальною та просторовою надійністю використовується карта просторової надійності для налаштування підтримки фільтра на частину вибраної області з кадру для відстеження. Це забезпечує збільшення і локалізацію вибраної області та покращує відстеження непрямокутних областей або об'єктів. Він використовує лише 2 стандартні функції. Він також працює з порівняно меншою частотою кадрів (25 кадрів в секунду), але забезпечує вищу точність відстеження об'єктів.

Першоджерело: [9]

2.2.4 Аналіз MedianFlow трекеру

Внутрішньо цей трекер відстежує об'єкт у прямому та зворотному напрямках у часі та вимірює розбіжності між цими двома траєкторіями. Надійно виявляє збої у відстеженні та вибирає надійні траєкторії у відеопослідовності. Трекер найкраще працює, коли траєкторія руху передбачувана і невелика за відстанню. На відміну від інших трекерів, які продовжують працювати навіть тоді, коли відстеження явно не вдалося, цей трекер знає, коли відстеження не вдалося.

Першоджерело: [9]

2.2.5 Аналіз TLD трекеру

TLD розшифровується як відстеження, навчання та виявлення. Як випливає з назви, цей трекер розбиває довгострокове завдання відстеження на три компоненти - (короткострокове) відстеження, навчання і виявлення. Детектор локалізує всі появи, які були помічені до цього часу, і коригує трекер, якщо це необхідно.Навчання оцінює помилки детектора і оновлює його, щоб уникнути цих помилок у майбутньому. Вихідні дані цього трекера мають тенденцію поводити себе нестабільно. Наприклад, якщо ви відстежуєте пішохода, а на сцені є інші пішоходи, цей трекер іноді може тимчасово відстежувати іншого пішохода, а не того, якого ви планували відстежувати. Позитивним моментом є те, що цей трекер відстежує об'єкт у більшому масштабі, русі та оклюзії. Цей трекер може бути хорошим вибором в тому випадку, коли об’єкт прихований за іншим об’єктом.

Найкраще він працює під час оклюзії на декількох кадрах. Також найкраще відстежує зміни масштабу. Значним недоліком є велика кількість хибних спрацьовувань, що робить його майже непридатним для використання.

Першоджерело: [9]

2.2.6 Аналіз MOSSE трекеру

MOSSE розшифровується як мінімальна вихідна сума квадратів помилок використовує адаптивну кореляцію для відстеження об'єктів, яка створює стабільні кореляційні фільтри при ініціалізації за допомогою одного кадру. Трекер MOSSE стійкий до змін освітлення, масштабу, пози та нежорстких деформацій. Він також виявляє оклюзію на основі співвідношення піку до бічної сторони, що дозволяє трекеру призупинити роботу і продовжити з того місця, де він зупинився, коли об'єкт знову з'являється. Трекер MOSSE також працює з більш високою частотою кадрів (450 кадрів в секунду і навіть більше). До позитивних моментів слід додати,

що його також дуже легко реалізувати, він настільки ж точний, як і інші складні трекери, і набагато швидший. Але за шкалою продуктивності він відстає від трекерів на основі глибокого навчання.

Першоджерело: [9]

2.2.7 Аналіз GOTURN трекеру

З усіх алгоритмів відстеження в класі трекерів це єдиний алгоритм, заснований на згортковій нейронній мережі. Він стійкий до змін точки зору, освітлення та деформацій. Але він не дуже добре обробляє оклюзію.

З GOTURN, будучи трекером на основі згорткованої нейронної мережі, використовує модель Caffe для відстеження. Модель Caffe і прототекстовий файл повинні бути присутніми в каталозі, в якому знаходиться код.

Першоджерело: [9]

2.2.8 Порівняння трекерів між собою

Проаналізувавши всі трекери і зібравши інформацію з офіційного сайту з документацією[9], отримуємо їх порівняльну таблицю:

Таблиця 2.1 – Порівняння трекерів

Трекер	Точність	Швидкість	Стійкість до змін освітлення	Здатність до масштабування	Складність реалізації	Найкраще використання
BOOSTING	низька	низька	низька	низька	низька	Просте відстеження, мінімальні зміни траєкторії руху цілі

Продовження таблиці 2.1

MIL	середня	низька	середня	середня	середня	Складні умови, кардинальні зміни траєкторії руху
KCF	висока	висока	середня	низька	низька	Відстеження цілей з плавними рухами
CSRT	висока	середня	висока	висока	середня	Складні траєкторії руху
MedianFlow	висока	висока	низька	низька	низька	Відстеження з постійним рухом
TLD	середня	низька	середня	висока	висока	Виявлення і довгострокове відстеження
MOSSE	середня	дуже висока	низька	низька	низька	Для платформ з обмеженими ресурсами
GOTURN	висока	низька	висока	висока	висока	Відстеження на основі глибокого навчання

Серед усіх трекерів, найкращим для виконання функціїї відстеження і керування камерою дрону є KCF трекер. Аргументую свій вибір низькою складністю до реалізації, що допоможе легко і швидко провести необхідне дослідження. Високі показники точності і швидкості виконання завдання дають можливість провести адекватну оцінку розробленого методу відстеження. Також цей трекер має середні показники стійкості до змін освітлення, що не є великою перевагою, але і не є великим недоліком в

порівнянні з іншими. Здатність до масштабування в даному проекті взагалі не важлива, тому низькі показники KCF трекеру в даному аспекті також не важливі. Отже, трекер KCF буде основни для виконання дипломної роботи.

2.3 Вибір технологій за допомогою яких буде проведена розробка та тестування системи.

2.3.1 Вибір мови програмування та її бібліотек

Основна реалізація буде виконана за допомогою мови програмування Python та її біблотек. Порівнювати Python будемо з мовою програмування C++, яка теж широко і успішно використовується у програмуванні дронів та створенні нейронних мереж, та включає в себе всі принципи ООП(об'єктно орієнтованого програмування). Python це простий і гнучкий інструмент можливості якого можна використовувати у будь-якому аспекті програмування. Також мова програмування Python має зрозумілий для прочитання та легкий написання код в порівнянні з іншою мовою яка б теж могла зайняти його місце – C++ . Ще одною перевагаю Python'у є велика кількість бібліотек, які дуже легко імпортуються до коду, в порівнянні з його конкурентами. З іншого боку C++ хоч і збільшує час розробки та її складність, але продуктивність роботи коду швидша, а затрати пам'яті меньші. Все ж таки Python виглядає краще на фоні своїх мов конкурентів, через велику кількість своїх плюсів, звісно кожна мова програмування має свої особливості, яких немає у Python'і, але це працює і навпаки. Отже, Python обрано для виконання проєкту. Основними бібліотеками будуть AirSim. OpenCV та Yolov8.

2.3.2 Бібліотека AirSim та аргументація її вибору

AirSim - потужна бібліотека, яка розроблятється компанією Microsoft. На жаль популярності ця утиліта не має і її спільнота фактично нульова. AirSim має прекрасну офіційну документацію, яка повністю висвітлює все

що наявно цій бібліотеці і надає практичні приклади використання тих чи іншнх речей. Суть AirSim полягає у створенні та налаштуванні БПЛА у симуляторі за допомогою скриптів та конфігураційних файлів. Також є можливіть інтегрувати до програмного забезпечення дрону нейронні мережі, що відкриває неймовірні можливісті. Контролювати БПЛА можна за допомогою будь якого геймпаду, або відповідних контролерів. Конкурентів у бібліотеки AirSim немає, тому і обирати по факту нічого, це найкраща утиліта для програмування та симуляцій роботи дронів.

2.3.3 Бібліотека OpenCV та аргументація її вибору

OpenCV – це також доволі потужна утиліта для використання технологій Computer vision, розробляється компанією Intel, у вигляді open source(відкритий для всіх) проєкту. Ця бібліотека дозволить нам за допомогою детектора, який буде натренованих по ходу проєкту, та трекерів злегкістю фіксувати камеру на об’єктах та змінювати відповідно положення камери, як нам потрібно. OpenCV є основним в даному проекті, оскільки на базі його трекерів і буде формуватися більша частина роботи. Адекватних аналогів OpenCV не існує. Бібліотека має велику підтримку як зі сторони розробників, так зі сторони спільноти, більшість проблем та важливих питань вже обговорено на форумах, сформульована детальна документація. Причин для вибору іншої бібліотеки не має. Отже, OpenCV також приєднується до набору технологій даного проекту.

2.3.4 Бібліотека Yolov8 та аргументація її вибору

Yolov8 – це також open source продукт, який використвоується для тренування нейроннж мереж. Основними перевагами Yolov8 є: висока швидкість розпізнавання об’єктів, висока точність розпізнавання цілей, гнучкість використання, простота і найважливіше сумісність з OpenCV. В тексті дипломного проекту було вказано, що трекер і детектор це речі які повинні існувати разом. OpenCV використовується в даній роботі для

створення трекерів, а Yolov8 для тренування детектора, тобто вони також в умовах даного проекту не можуть існувати один без одного. Також великим плюсом Yolov8 є його актуальність, оскільки це один із найсучасніші інструментів у галузі комп'ютреного зору, та величезна спільнота та підтримка з сторони спільноти.

2.3.5 Вибір середовища для створення симулятора

Unreal Engine[6] – ігровий двигун, який розробляється компанією Epic Games. На вибір ми мажмо всього два ігровових рушія це Unreal Engine та Unity, для того щоб аргументувати вибір саме першого з них, спочатку порівняємо важливі для виконання роботи якості цих інстрементів та їх можливості. Почнемо з Unity, його перевагами є: простота використання, гнучкість, велика спільнота, інтеграція з AirSim та підтримка мови програмування C#. Стосовно Unreal Engine, він хоч і має складнішу структуру, але його перевагами над аналогом змушують обрати саме його, а саме такі переваги як: вища якість графіки, складний, потужніший редактор, висока продуктивність, підтримка технології Blueprint, що дозволяє швидко створювати та тестувати нові функції без необхідності писати код. Отже, для більш реалістичних та деталізованих симуляцій дронів, особливо якщо важлива висока якість графіки, краще обрати Unreal Engine.

2.3.6 Вибір редактору коду

Visual Studio Code - це безкоштовний редактор коду, розроблений компанією Microsoft. Вибір редактору це індивідуально, для виконання проекту можна скористатися і будь яким іншим редактором. Я буду користуватисть сами ним, тому що він не навантажує систему, має вбудований термінал, інтагрецію з Git, також має можливості створення точок зупинення під час виконання, відстеження значень змінних і покрокове виконання коду. Найголовнішим плюсом є велика кількість

розширень редактору, що дозволить спростити написаня, читання та редагування коду.

ВИСНОВКИ ДО РОЗДІЛУ 2

Розглянуто найпопулярніші трекери для OpenCV. На базі офіційної документації створено таблицю основних характеристик. Після аналізу наявних рішень головним для виконання завдання я обрав трекер - KCF..

Також було проведено вибір технологій для розробки з докладною аргументацією того чи іншого вибору. А саме були обрані, мова програмування, бібліотеки та інші утиліти, середовище розробки симулятора та редактор коду. Збільшення теоретичної бази та вибір технологій дозволяє перейти до проектування та розробки системи підготовленим.

РОЗДІЛ 3.

ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

3.1 Встановлення віртуального оточення

Перед тим як перейти до оточення, встановимо потрібні нам утиліти. Необхідно встановити на Windows 10/11:

1. Unreal Engine 4.27
2. Скомпілювати AirSim (https://microsoft.github.io/AirSim/build_windows/)

Встановимо нашу середу розробки, це буде VS Code. Скачуємо інсталятор за посиланням[4] і встановлюємо.

Тепер можна перейти до встановлення мови програмування, а саме Python версії 3.9. Для цього потрібно скачати інсталятор та запустити його за посиланням[10].

Наступним кроком буде встановлення віртуального середовища за допомогою дистрибутива miniconda. Виконаємо це за допомогою командної строки та наступних команд:

1. curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe -o miniconda.exe
2. start /wait "" miniconda.exe /S
3. del miniconda.exe

Створюємо наше віртуальне середовище за допомогою Python 3.9.

Виконаємо наступні пункти:

1. Запуск miniconda: Натисніть кнопку "Win" -> "Anaconda Prompt (Miniconda3)".

2. Встановіть Python 3.9: conda create -n ball-tracking python=3.9

Остання команда створює віртуальне середовище з назвою “ball-tracking” на базі мови програмування Python версії 3.9. Навіть якщо в операційній системі буде встановлена інша версія мови Python, при активації віртуального середовища ми будемо працювати з зазначеною вище версію, тобто 3.9.

Наступним кроком буде активація віртуального середовища за допомогою команди, яку ми вводимо до командної строки:

- conda activate ball-tracking

Та вводимо в командну строку наступні команди для встановлення потрібних нам бібілотек:

1. pip install msgpack-rpc-python
2. pip install "numpy>=1.25.0,<1.26"
3. pip install "opencv-python>=4.5.1,<4.7"
4. pip install "airsim==1.8.1"

Для перевірки правильності встановлення виконуємо наступні дії.

Завантажимо тестовий проєкт за посиланням:

https://e.pcloud.link/publink/show?code=kZE4DnZQ0IPTLhzdjuaEDoOzv1vUuG0s2G7

Тестовий проєкт містить такі файли:

1. BallTracking.zip -- файл проєкту Unreal Engine.
2. settings.json -- конфіг файл для AirSim.
3. 2024-02-19-test-01.py -- код на пайтон.
4. ball-tracking-demo.mp4 -- демо відео, що має вийти в результаті цього проєкту.

Після встановлення усих потрібних нам технологій на даному етапі, перевіримо правильність роботи симулятора.

1. Скопіюйте "settings.json" до "C:/Users/<ім'я користувача>/Documents/AirSim/settings.json"
2. Розпакуйте BallTracking.zip
3. Запустіть BallTracking.uproject проєкт, натисніть Play (виберіть режим коптера).
4. Запустіть 2024-02-19-test-01.py.

Якщо все правильно ми можемо керувати нашим дроном за допомогою контролера і отримуємо ось такий результат на екрані:

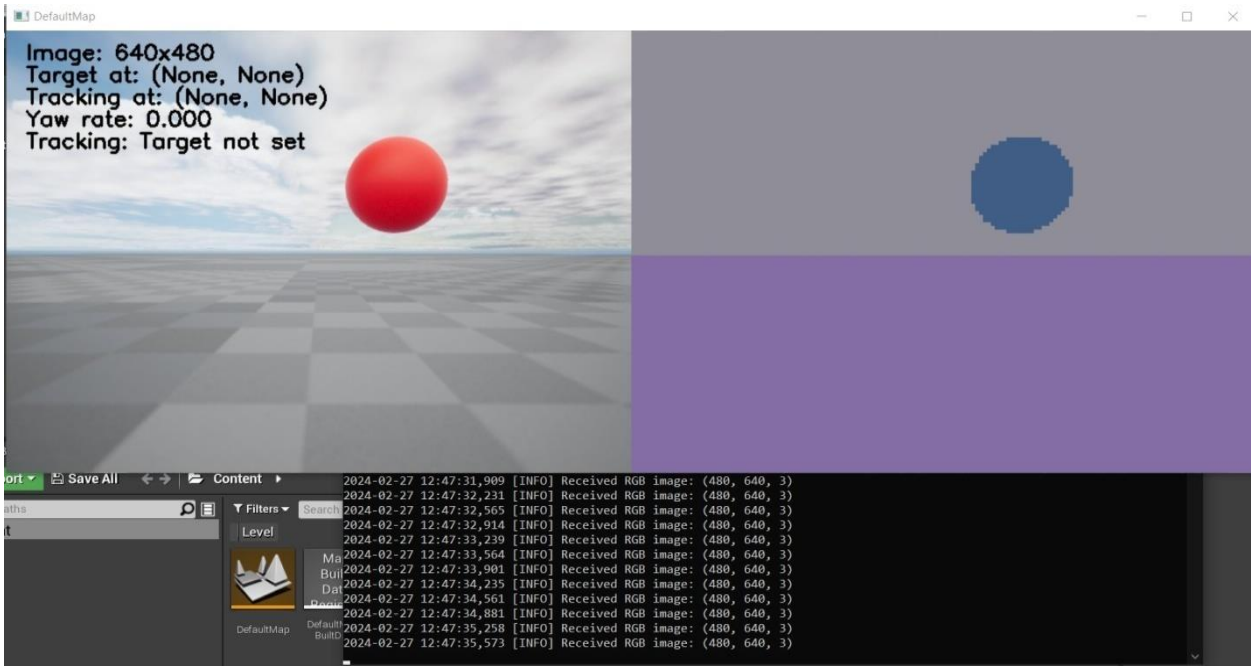


Рис.3.1 - Результат тестування

Також можуть виникнути проблеми з запуском початкової програми. Якщо таке трапилося знову запускаємо наше віртуальне середовище за допомогою команди:

- conda activate ball-tracking

Перевіряємо правильність встановлення версії мови Python за допомогою команди:

- python -v

Якщо версія збігається з потрібною нам, а саме 3.9 переходимо до наступного пункту. При невідповідності повертаємося на початок і знову завантажуємо інсталятор мови Python потрібної нам версії.

Тепер можемо перевірити правильність встановлення бібліотек за допомогою команди:

- pip list

Шукаємо у списку потрібні нам утиліти, а саме

1. msgpack-rpc-python будь якої версії
2. numpy версій від 1.25.0 до 1.26
3. opencv-python версій від 4.5.1 до 4.7
4. airsim версії 1.8.1

Якщо хоча б одна з версій не підходить для нашого проекту, перевстановлюємо відповідні бібілотеки за допомогою команд описаних вище.

Перевіривши утиліти на правильність роботи, перевіримо також версію ігрового двигуна, що виступає в ролі симулятора, а саме версію Unreal Engine яка повинна бути 4.27. Це можна перевірити декількома методами, наприклад:

1. Запустити програму Epic Games і у вкладці Unreal Engine на кнопці запуску вказується версія.
2. Запустити Unreal Engine за допомогою ярлику, при завантажені буде відразу написана версія вашого ігрового двигуна.

Знову ж таки при невідповідності версії, повертаємося назад і перевстановлюємо заново потрібну нам версію.

Якщо проблем з версіями не виникло, страшною проблемою може бути конфігураційний файл setting.json. Проблемою шляху в якому він зберігається C:/Users/<ім'я користувача>/Documents/AirSim/settings.json" є застосування операційною системою Windows хмарного середовища

OneDrive. Проблема виникає при кожному повторному запуску програми. Конфігураційний файл налаштувать дрону приймає стандартні значення і витирає потрібний нам код. Я знайшов спосіб вирішення цієї проблеми, але для їх вирішення рекомендую запускати проект за допомогою віртуальних операційних систем, оскільки для вирішення потрібно не зовсім традиційними методами витягнути папку Documents з хмари і зробити її основною для системи. Отже головна ідея вирішення, це вилучити папку Documents з хмари OneDrive і зробити її стандартною за замовчуванням, реалізацій такої ідеї безліч, але всі вони можуть дуже сильно пошкодити систему, тому і рекомендую запускати проект на віртуальній операційній системі, якщо виникла така проблема.

Також можуть виникнути проблеми в керуванні дроном, тут все набагато простіше, перевіряємо правильність підключення контролера і за допомогою безлічі онлайн ресурсів перевіряємо правильність його роботи.

3.2 Створення датасету

Для початку у нашому вже запущеному проекті виконаємо такі зміни:

Змінюємо траєкторію польоту кулі так, щоб зробити її подібною на прямокутний трикутник. Це для того, щоб візуально спостерігати за об'єктом під час його руху в бік дрону, від дрону, та у бік від дрона.

Для цього змінюємо в нашому редакторі Unreal Engine розташування контрольних точок Movement Blueprint, що вже розміщений на карті та до якого прив'язана куля. Детальніше про технологію Blueprint можна прочитати за посиланням[11]

Тепер можемо створити наш набір даних або ж датасет(в подальшому будемо використовувати цей термін). Для цього нам потрібно отримати зображення з камери дрону. При запуску симулятора

натискаємо клавішу F1 і отримуємо перелік базових функцій і відразу натискаємо F3 для того щоб прибрати вдображення каркасу навколишнього середовища.

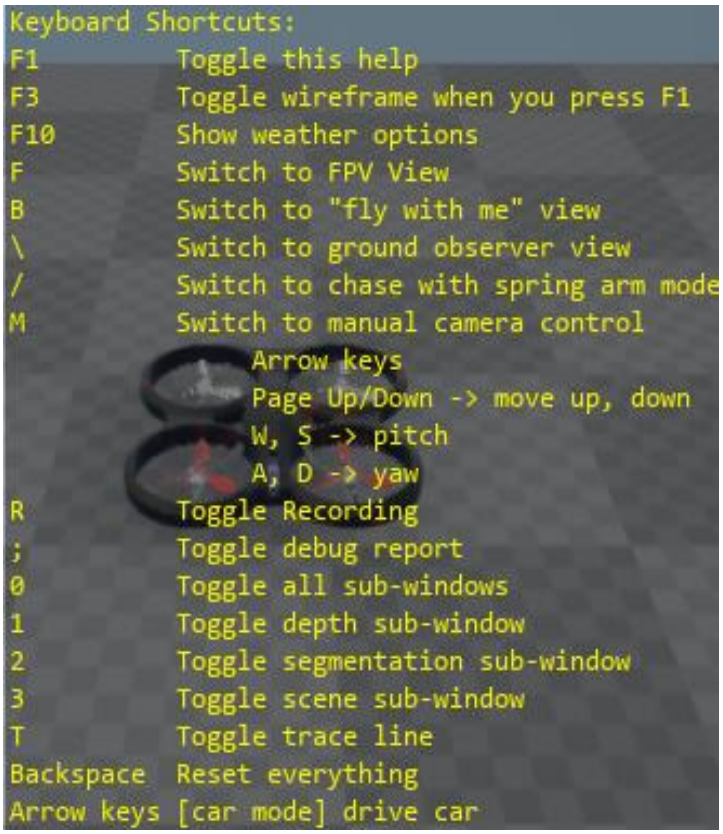


Рис.3.2 – Перелік базових функцій дрону

F1 – відкрити довідник

F3 – структурне відображення, автоматично вмикається при натисканні клавіши F1

F10 – Показник погодних умов

F – перемикання на вид від першого лиця камери

B – вид від третього обличчя навколо дрону

\ - вид знизу дрону

/ - режим переслідування

M – контроль положення камери за допомогою стрілочок на клавіатурі і клавіш W, S для зміни вертикального положення камери і клавіш A, D для зміни горизонтального положення камери

R – запис з камери дрону

; - звіт помилок

0 – відкрити всі допоміжні вікна

1 – відкрити вікно камери з глибинним баченням

2 – відкрити вікно камери з виділеними сегментами

3 – відкрити допоміжне вікно з відображенням сцени симулятора

T – відображення лінії траєкторії руху дрону

Backspace – перезавантаження налаштувань

Використаємо функцію, що активується за допомогою клавіши R, що дозволить нам отримати зображення з камери дрону і відразу збереже їх до відповідної директорііїї:

“C:/Users/<ім'я користувача>/Documents/AirSim/”

Використовуючи контролер, облітаємо кулю з різних ракурсів. Також потрібно створити мітки для тренування нашого датасету, для цього обводимо на кожному зображені нашу кулю прямокутною рамкою одного кольору на всіх зображеннях і підписуєма написом “ball”. Повторюємо політ з відеозаписом 10 разів і виставляємо мітки на зображеннях.

Копіюємо наш датасет у папку проекту під назвою “Dataset”.

3.3 Тренування НМ та інтеграція в алгоритм трекера

Детектор ми будемо тренувати за допомогою нейронних мереж. Потім інтегруємо детектор до трекеру, тому що вони не можуть існувати один без одного.

Для тренування НМ потрібно створити два файли в директорії проекту це nn_model.py і dataset.yaml.

nn_model.py – містить код, який забезпечить тренування НМ

dataset.yaml – конфігураційний файл, який забезпечить взаємодію коду з датасетом

Після того, як ми заповнили відповідні файли потрібним нам кодом, запускаємо файл nn_model.py у віртуальному середовищі за допомогою команди: python nn_model.py

Очікуємо результат виконання програми.

Отримуємо файл який містить в собі натреновану нейронну мережу: yolo8n.pt

Повертаємося до нашого файлу 2024-02-19-test-01.py і змінюємо назву на main.py для подальшого спрощення, так як цей скрипт є основним в даному дипломному проекті. Знаходимо алгоритм трекера в коді він знаходиться у класу “OpticalFlow” в методі “target”. Інтегруємо наш детектор до трекеру.

Перевіряємо правильність роботи трекеру і детектору в комбінації. Після запуску програми камера дрону буде намагатися тримати ціль в центрі зображення, але так як ще не створено алгоритм руху камери це буде виглядати не так як хотілось би і унеможливить нормальне керування дрону за допомогою контролера. Головне зараз спостерігати за значенням на екрані “Target at” і “Tracker at”, вони повинні співпадати.

Також перевіримо на практиці здатності інших трекерів з натренованим детектором, а саме: BOOSTING, MIL, KCF, CSRT, MedianFlow, TLD, MOSSE, GOTURN. Для цього створимо відповіду змінну в класі “Experiment” яка буде мати вигляд і містити такі значення:

```
self.trackers = {
        "BOOSTING": cv2.legacy.TrackerBoosting_create(),
        "MIL": cv2.TrackerMIL_create(),
        "KCF": cv2.TrackerKCF_create(),
        "CSRT": cv2.TrackerCSRT_create(),
        "MedianFlow": cv2.legacy.TrackerMedianFlow_create(),
```

```
        "TLD": cv2.legacy.TrackerTLD_create(),
        "MOSSE": cv2.legacy.TrackerMOSSE_create(),
    }
```

Тепер ми злегкістю можем змінювати трекери в коді, це потрібну для аналізу результатів в подальшому.

3.4 Прямування за ціллю, рух дрону і камери відповідно до об’єкту

Для відстеження цілі ми використовуємо значення координат об’єкту які ми отримуємо з камери дрону. Відповідно до координат буде змінюватись і положення камери. За ось такими розрахунками будемо змінювати положення камери дрону:

1. Різниця в положенні між дроном і об'єктом(x) = позиція об’єкту(x) – позиція дрону(x)
2. Різниця в положенні між дроном і об'єктом(y) = позиція об’єкту(y) – позиція дрону(y)
3. Різниця в положенні між дроном і об'єктом(z) = позиція об’єкту(z) – позиція дрону(z)
4. Позиція камери = вектор (Різниця в положенні між дроном і об'єктом(x), Різниця в положенні між дроном і об'єктом(y), Різниця в положенні між дроном і об'єктом(z)))

Наведений вище псевдокод описує алгоритм руху камери відповідно до руху цілі. Окрім відстеження цілі, необхідно також змінювати положення камери та/або дрона. Також тримаємо ціль на фіксованому рівні не лише по осі X, а й по осі Y. Змінимо наш псевдокод для нової задачі:

1. Різниця в положенні між дроном і об'єктом(x) = позиція об’єкту(x) – позиція дрону(x)
2. Різниця в положенні між дроном і об'єктом(y) = позиція об’єкту(y) – позиція дрону(y)

3. Різниця в положенні між дроном і об'єктом(z) = позиція об’єкту(z) – позиція дрону(z)

4. Нова позиція дрону(x) = позиція дрону(x) + Різниця в положенні між дроном і об'єктом(x) * 0,1

5. Нова позиція дрону(y) = позиція дрону(y) + Різниця в положенні між дроном і об'єктом(y) * 0,1

6. Нова позиція дрону(z) = позиція дрону(z) + Різниця в положенні між дроном і об'єктом(z) * 0,1

7. Згідно визначених координат встановлюємо нову позицію дрону

8. Рахуємо кут між позицією дрону та цілі

9. Встановлюємо позицію камери згідно визначених координат та кутів

Для того щоб тримати ціль на фіксованій відстані від дрона використаємо усі наявні методи контролювання дрону кодом за допомогою функцій yaw, throttle, pitch та roll.

Після правильного виконання всих дій отримуємо бажаний результат. Наш дрон та/або камера рухаються відповідно до зміни положення цілі, переконатися в цьому можна завдяки запуску проекту. Розробка програмної системи виконана успішно.

3.5 Встановлені компоненти та бібліотеки

Розділ “Dependencies” містить всі використані утиліти:

Dependencies:

- pip=24.0 - це менеджер пакунків для Python.
- python=3.9=h7a1cb2a_0 - це широко використовувана високорівнева, загальнопризначена, інтерпретована, динамічна мова програмування.

- Unreal Engine 4.27 - ігровий двигун, який розробляє і підтримує компанія Epic Games.
- AirSim - симулятор дронів, автомобілів та іншого, побудований на Unreal Engine
- VS Code - це легкий, але потужний редактор вихідного коду, який працює на вашому робочому столі і доступний для Windows, macOS і Linux.

Розділ “pip” містить список Python-пакунків, які встановлюються через pip, який є стандартним менеджером пакунків для Python. Їх перелік з версіями приведений нижче:

Pip:

- msgpack-rpc-python - міжмовна RPC-бібліотека для клієнтських, серверних та кластерних додатків
- numpy>=1.25.0,<1.26 - це фундаментальний пакет для наукових обчислень на мові Python. Це бібліотека Python, яка надає об'єкт багатовимірного масиву, різні похідні об'єкти (такі як масиви і матриці з масками), а також набір процедур для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з фігурами, сортування, вибірку, введення/виведення, дискретні перетворення Фур'є, базову лінійну алгебру, базові статистичні операції, випадкове моделювання і багато іншого.
- opencv-python>=4.5.1,<4.7
- airsim==1.8.1 - симулятор з відкритим вихідним кодом на базі Unreal Engine для автономних автомобілів від Microsoft AI & Research
- ultralytics==8.2.32 – бібліотека для тренування НМ, містить в свожму арсеналі Yolov8

- PIL ==10.3.0 – утиліта для створення та зберігання зображень

3.6 Структура проекту

При правильному виконанні вище згаданих пунктів наша директорія, що містить проект повинна виглядати таким чином:

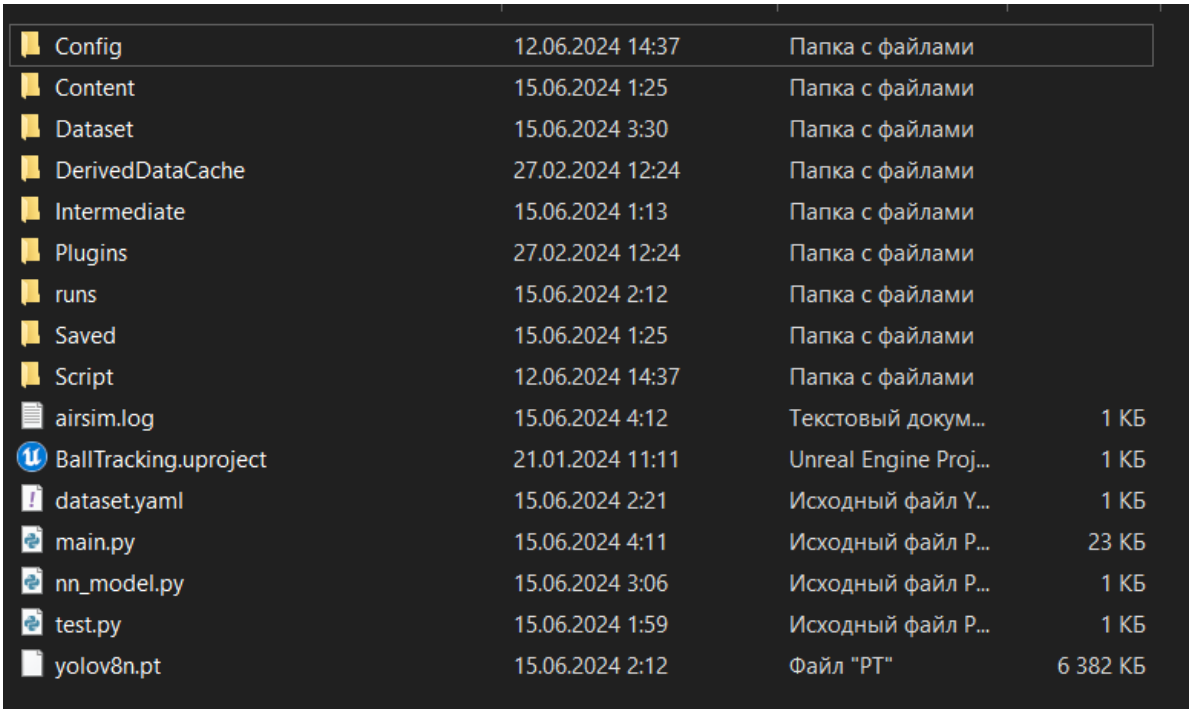


Рис.3.3 - Структура проєкту

- Config – конфігураційні файли проєкту
- Content – директорія для зберігання об’єктів Unreal Engine проєкту
- Dataset – створений під час польотів ДС
- DerivedDataCache – кеш проєкту
- Intermediate – також кеш проєкту
- Plugins – встановленні плагіни Unreal Engine
- runs – директорія для відстеження усіх епох нейронної мережі при навчанні

- Saved – директорія слугує для автозберігання проєкту на Unreal Engine
- Script – скрипти проєкту(Unreal Engine проєкту)
- airsim.log – текстовий файл для відображення роботи системи
- BallTracking.uproject – основний Unreal Engine проєкт
- dataset.yaml -конфігураційний файл для навчання нейронної мережі
- main.py – основний скрипт в основі якого і лежить вся взаємодія з дроном
- nn_model.py – скрипт для навчання нейронної мережі
- test.py – скрипт для перевірки правильності роботи бібліотек, функцій чи методів
- yolov8n.pt – файл, який вміщує в себе натреновану нейронну мережу

Також використовується конфігураційний файл для нашого дрону: setting.json, який розташований у “C:/Users/<ім'я користувача>/Documents/AirSim/ ” .

ВИСНОВКИ ДО РОЗДІЛУ 3

Створено основне програмне забезпечення проєкту. Всі дії описано послідовно. З використанням офіційної документації проблем не виникло. Також проведено тестування початкого запуску і вирішення складностей на даному етапі.

Також ми вже натренували власний детектор і інтегрували з усіма вище розписаними трекерами, що дозволило трішки більше переконатися у їх різниці. На жаль, різницю між трекерами можна побачити лише візуально за допомогою відеозаписів, картинки зовсім не передають тієї різниці, тому їх тут і не прикріплено. Ідея обрати трекер MIL була виправданою, візуальна робота цього трекера одна з найкращих. Описано зміну положення камери та дрону за допомогою псевдокоду, що чітко дає зрозуміти принцип роботи алгоритму. Також в цьому розділі описані залежності та встановленні бібліотеки. Структуризація проекту виконана і кінцевий результат з поясненням було виконано. Програмне забезпечення проекту виконано успішно.

РОЗДІЛ 4

АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Результат тестування трекерів

Перевіримо роботу кожного трекеру, для того щоб підтвердити правильність вибору основного трекеру дослідження(це був KCF трекер). Для цього я виміряв середнє значення швидкості пошуку цілі за 10 спроб, кожним трекером окремо, на практиці. Також я буду робити висновки стосовно трекерів вже не практиці, якщо точність виявлення я знаю приблизно вже від початку і її показники були наведені вище у таблиці, то значення швидкості потрібно було виміряти. Отримав ось такі результати:

Таблиця 4.1 – Результат роботи трекерів

Трекер	Швидкість пошуку цілі, с	Висновок
BOOSTING	0,07	Точність виявлення цілі низька, низька швидкість
MIL	0,06	Точність виявлення низька, низька швидкість
KCF	0,02	Середні показники точності і швидкості
TLD	0,07	Висока точність, низька швидкість
MEDIANFLOW	0,025	Висока точність та середня швидкість
GOTURN	0,1	Висока точність, дуже великі затрати ресурсів процесору, надповільна швидкість

Продовження таблиці 4.1

MOSSE	0,0002	Дуже висока швидкість, середня точність виявлення цілі
CSRT	0,038	Дуже висока точніть виявлення цілі, середня швидкість

Вибір трекеру KCF є виправданим, на практиці ми в цьому переконалися, середні показники швикдості та точності пошуку цілі забезпечують можливість тестування роботи натренованого детектора в повній мірі. Кожен інший трекер має свій значний недолік або низьку точніть виявлення, або низьку швидкість виявлення, або великі затрати ресурсів процесору, або складність розробки та використання. Отже, трекер KCF як з практичної точки зору так і з теоретичної показав середні стабільні результати, без знахних недоліків у використанні.

4.2 Огляд створеного набору даних(датасету)

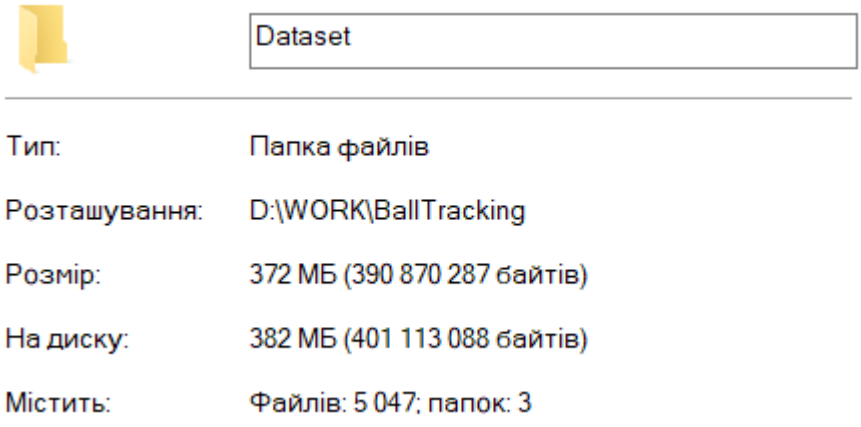


Рис.4.1 - Параметри директорії Dataset

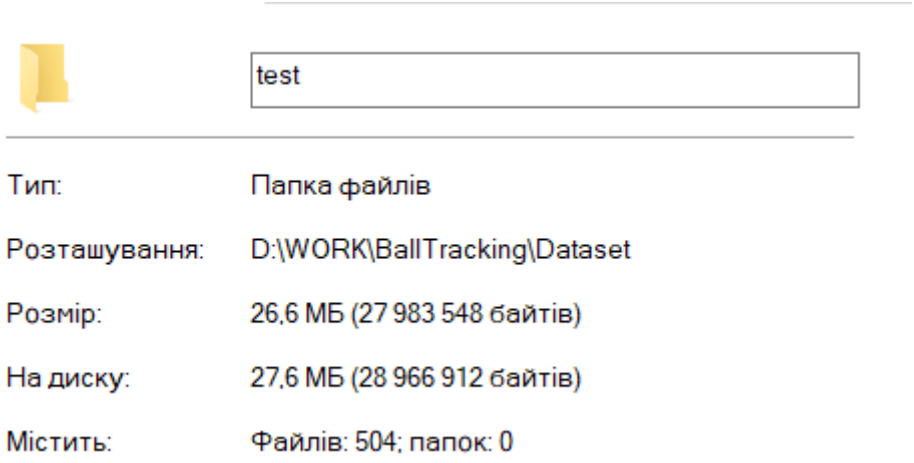


Рис.4.2 - Параметри директорії test

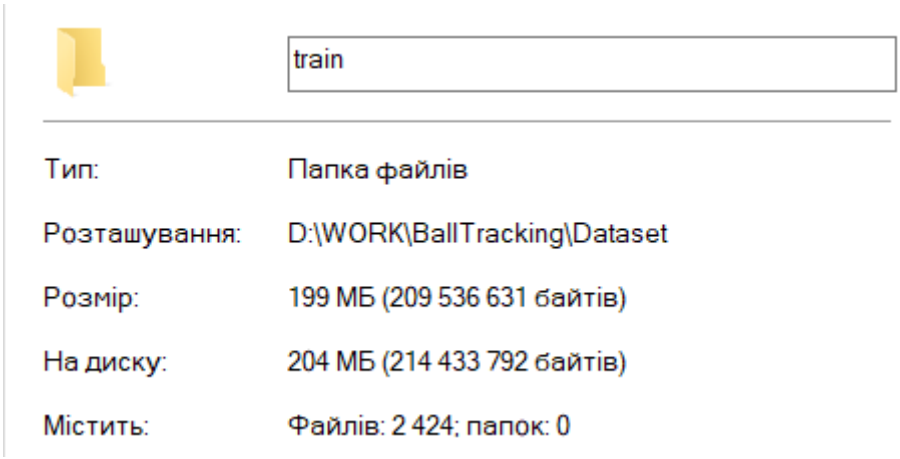


Рис.4.3 - Параметри директорії train

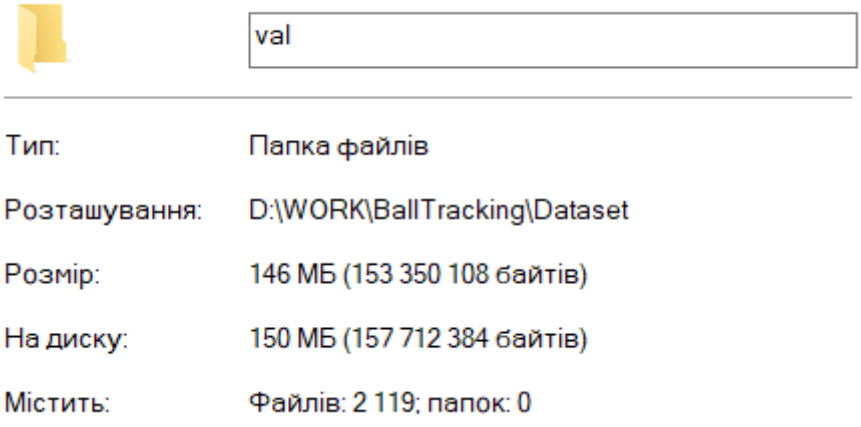


Рис.4.4 - Параметри директорії val

Почнемо аналіз з директорії, яка зображена на Рис.4.1. Папка Dataset є загальною папкою, що містить набір даних. Згідно рисунку, кількість

зображень за допомогою яких тренувалась НМ становить 5047 екземпрялрів. Загальна вага датасету на диску становить всього 382 МБ, що і не так багато.

Розлянемо директорію зображену на Рис.4.2. Папка test містить тестовий набір даних, який використовується для оцінки продуктивності моделі після завершення навчання. Згідно рисунку набір даних цієї директорії становить 504 файли.

Продовжимо аналіз, розглянемо директорію на Рис.4.3. Папка train містить набір навчальних даних, який використовується для тренування моделі. Згідко рисунку кількість даних в цій директорії становить 2424 файли, що є найбільшим показником.

Завершимо аналіз датасету посилаючись на Рис.4.4. Папка val містить набір даних, який використовується для налаштування гіперпараметрів моделі та перевірки її продуктивності під час тренування. Кількість файлів, що зберігається в цій директорії становить 2119 файлів.

Важливість такого поділу даних забезпечує об’єктивну оцінку продуктивності моделі та дозволяє слідкувати за перенавчанням моделі та корегувати гіперпараметри моделі. Без такого поділу варіантивність роботи моделі зменшується і вона не буде загальгою, а буде працювати лише на окремих прикладах навчального набору.

4.3 Результат роботи алгоритму відстеження

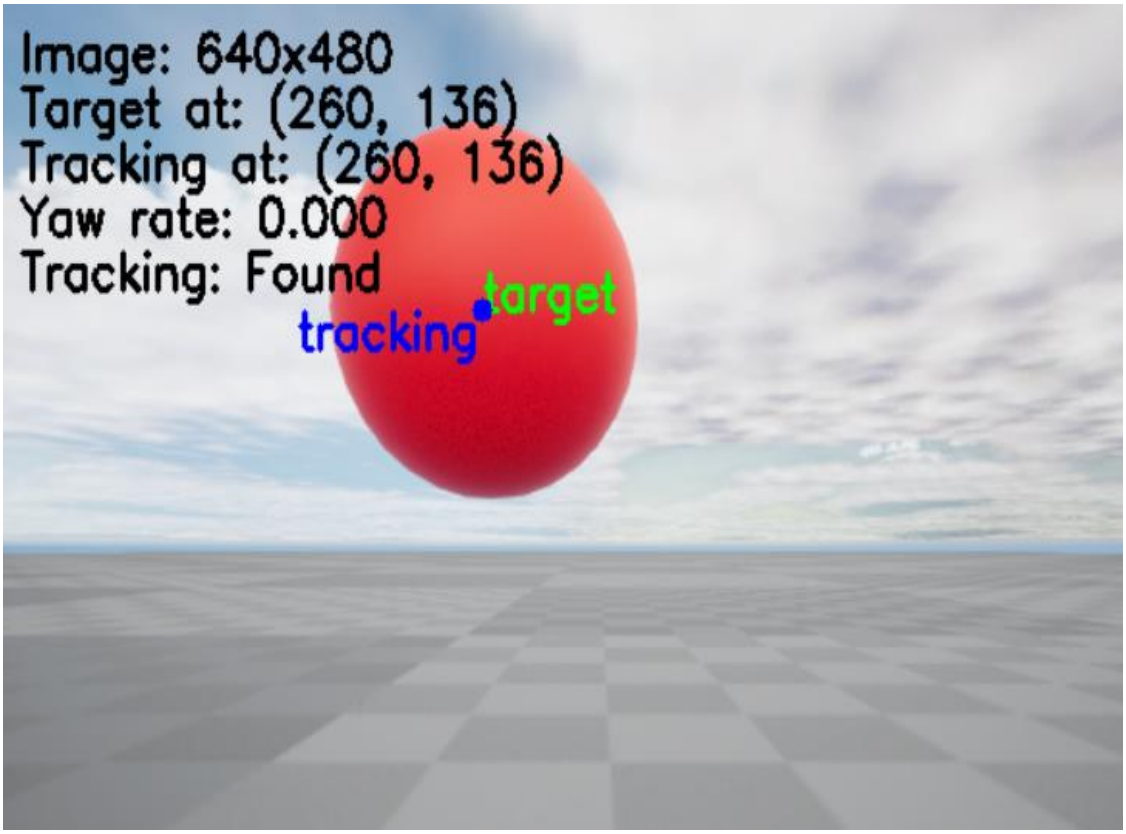


Рис.4.5 – Результат виконання роботи

На Рис.4.5 зображено результат роботи програми дрону, трекер цілі та детектор працюють так як треба, тобто трекер цілі співпадає з центром цілі і відповідно змінює положення камери та/або дрону в залежності від руху кулі. картинками неможливо, передати поворот камери відповідно руху об’єкта, але правильність роботи програми можна перевірити за допомогою лістингу коду який приєднаний у Додатку 4. Звісно трекер має певні похибки у визначенні положення цілі. Вирішенняи цієї проблеми є повторне навчання НМ на значно більшому наборі даних. Під час повторного навчання потрібно слідкувати за метрикою якості моделі:

- абсолютна відносна похибка: ця метрика показує, в середньому, на скільки відсотків прогнози моделі відрізняються від фактичних значень. Наприклад, MAPE = 10% означає, що в середньому прогноз моделі відрізняється від реального значення на 10%.

- середньоквадратична похибка: враховує як величину, так і напрямок помилок, тому великі похибки штрафуються сильніше, ніж маленькі. Це корисно для виявлення моделей, які мають декілька великих помилок.
- квадратична відносна похибка: вимірює середню відносну помилку, але зі штрафом за великі відхилення.
- середня логарифмічна похибка за основою 10: метрика зменшує вплив великих помилок і є корисною, коли важливо враховувати відносну різницю між прогнозами і фактичними значеннями, а не їх абсолютну величину. Вона також зменшує вплив екстремальних значень і є більш стабільною для різноманітних діапазонів даних.

ВИСНОВКИ ДО РОЗДІЛУ 4

В цьому розділі було проведено аналіз важливих для дослідження результатів. Порівняно характеристики трекерів, які були перевірені на практиці під час виконання дипломної роботи. Детальний огляд набору даних, з поясненням поділу датасету на три сегменти. Також було зображено результат виконання програми. Проблема похибки при спрацюванні алгоритму відстеження руху об’єкту полягає в незначному наборі даних, для покращення потрібно перенавчити нейронну мережу на новому значно більшому наборі даних. Головне слідкувати за гіперпараметрами НМ, щоб не було перенавчання та інших проблем, що можуть виникнути при начанні моделі. Отже, аналіз даних та результатів дослідження проведено успішно.

ЗАГАЛЬНІ ВИСНОВКИ

Успішно проведено дослідження на тему методів відстеження об’єкту з зміною положення камери, та були використанні такі основні технології, на яких і базується власне весь проєкт це: БПЛА і його можливості, та нейронні мережі і глибинне навчання. Робота була виконана за допомогою симулятора, геймпаду і мови програмування Python. Виконано огляд найпопулярніших трекерів OpenCV та досліджено їх характеристики. Створено власний детектор за допомогою нейронної мережі, натренованої за допомгою Yolov8.

Отже метод відстження базується, на отриманні зображення, пошуку об’єкту, зміні положення камери або дрону відносно об’єкту так, щоб БПЛА знаходився на однаковій відстані по мірі віддалення цілі. Окрім відстані, камера фіксує об’єкт в центрі зображення. Все це можливо завдяки нейронним мережам, які виконують основну задачу пошуку цілі.

Також були виділені перспективи розвитку проєкту: перенесення проєкту на фізичну платформу. Не мало важливим є значення цієї роботи в наш час, оскільки дана технологія може використовуватись військовими для виконання розвідувальних завдань або ж використання розумних дронів “камікадзе”.

Технічне завдання на дипломний бакалаврський проект виконано повністю.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Build AirSim on Windows [Електронний ресурс] - Режим доступу: https://microsoft.github.io/AirSim/build_windows/
2. NumPy [Електронний ресурс] - Режим доступу: https://numpy.org/doc/stable/user/whatisnumpy.html
3. MessagePack [Електронний ресурс] - Режим доступу: https://github.com/msgpack-rpc/msgpack-rpc#:~:text=MessagePack-RPC%20is%20cross-language,network%20applications%20with%20MessagePack-RPC.
4. Visual Studio Code [Електронний ресурс] - Режим доступу: https://code.visualstudio.com/docs#:~:text=Visual%20Studio%20Code%20is%20a,for%20Windows%2C%20macOS%20and%20Linux.
5. Welcome to AirSim [Електронний ресурс] - Режим доступу: https://microsoft.github.io/AirSim/
6. Wikipedia Unreal Engine [Електронний ресурс] - Режим доступу: https://ru.wikipedia.org/wiki/Unreal_Engine
7. Wikipedia OpenCV [Електронний ресурс] - Режим доступу: https://ru.wikipedia.org/wiki/OpenCV
8. Object Tracking [Електронний ресурс] - Режим доступу: https://pyimagesearch.com/2018/07/30/opencv-object-tracking/
9. OpenCV documentation [Електронний ресурс] - Режим доступу: https://docs.opencv.org/3.4/d1/d1a/classcv_1_1TrackerBoosting.html
10. Python [Електронний ресурс] - Режим доступу: https://www.python.org/downloads/release/python-390/
11. Blueprint [Електронний ресурс] - Режим доступу: Blueprint Overview | Unreal Engine 4.27 Documentation

12. Real-Time Object Tracking System Using OpenCV [Електронний ресурс] - Режим доступу: (PDF) Relevance of apocarotenoids in plant-microorganism communication: an opportunity for sustainable agriculture (researchgate.net)

13. The Guardian - London's CCTV system [Електронний ресурс] - Режим доступу: Latest news, sport and opinion from the Guardian

14. Wikipedia Tesla Autopilot - London's CCTV system [Електронний ресурс] - Режим доступу: Tesla Autopilot - Wikipedia

15. Wikipedia Dataset [Електронний ресурс] - Режим доступу: Дані — Вікіпедія (wikipedia.org)

16. Wikipedia Нейронні мережі [Електронний ресурс] - Режим доступу: Штучна нейронна мережа — Вікіпедія (wikipedia.org)

17. Wikipedia БПЛА [Електронний ресурс] - Режим доступу: Безпілотний літальний апарат — Вікіпедія (wikipedia.org)

ДОДАТОК 1

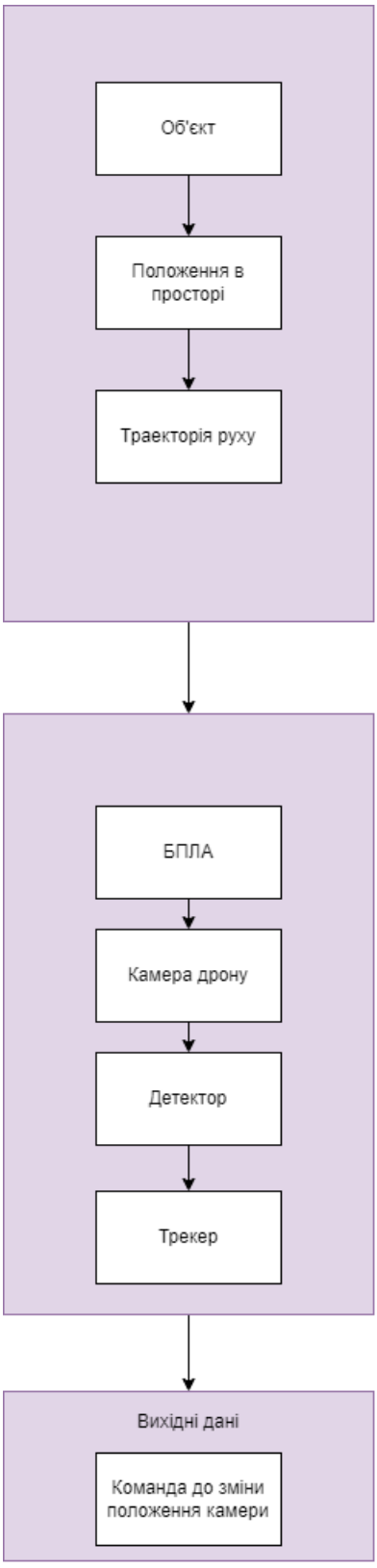
Метод відстеження руху об’єкта з керуванням положення камери

Схема роботи методу для відтсеження руху об’єкту (Схема структурна)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2024 р.



					ІАЛЦ.467200.004 Д1		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Бабич В.Ю.*			**Метод відстеження руху об'єкта з керуванням положення камери**	*Літ.*	*Аркуш* / *Аркушів*
Перевірив		*Гордієнко Ю.Г..*					*1* / *1*
					Схема роботи методу відстеження руху (Схема структурна)	КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-02	
Н. Контр.		*Виноградов Ю.М.*					
Затв.							

ДОДАТОК 2

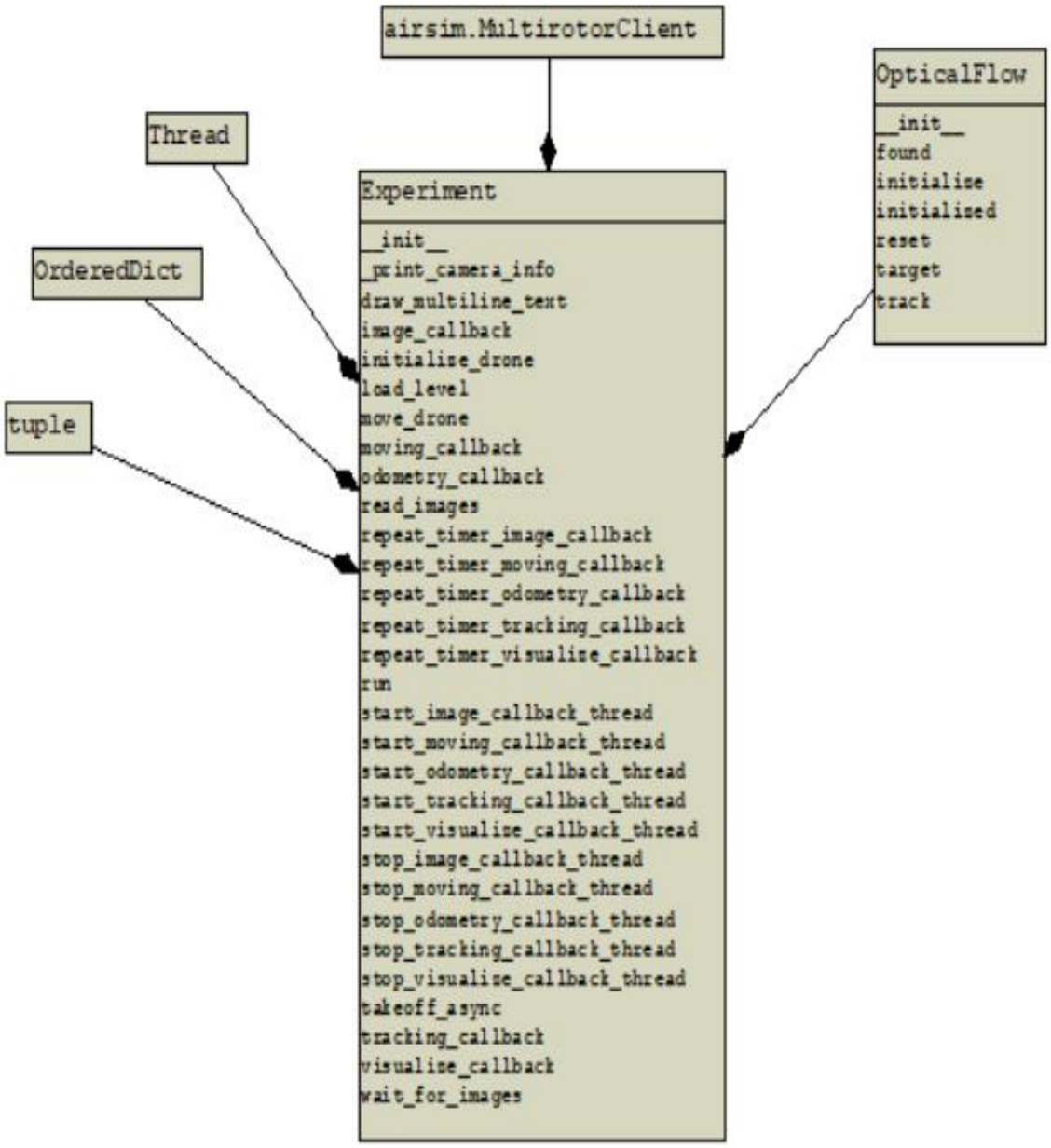
Метод відстеження руху об’єкта з керуванням положення камери

Діаграма класів (Схема функціональна)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2024 р.



					ІАЛЦ.467200.005 Д2		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Бабич В.Ю.*			**Метод відстеження руху об'єкта з керуванням положення камери** ***Діаграма класів (Схема функціональна)***	*Літ.*	*Аркуш* 1 / *Аркушів* 1
Перевірив		*Гордієнко Ю.Г..*					
Н. Контр.		*Виноградов Ю.М.*				КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-02	
Затв.							

ДОДАТОК З

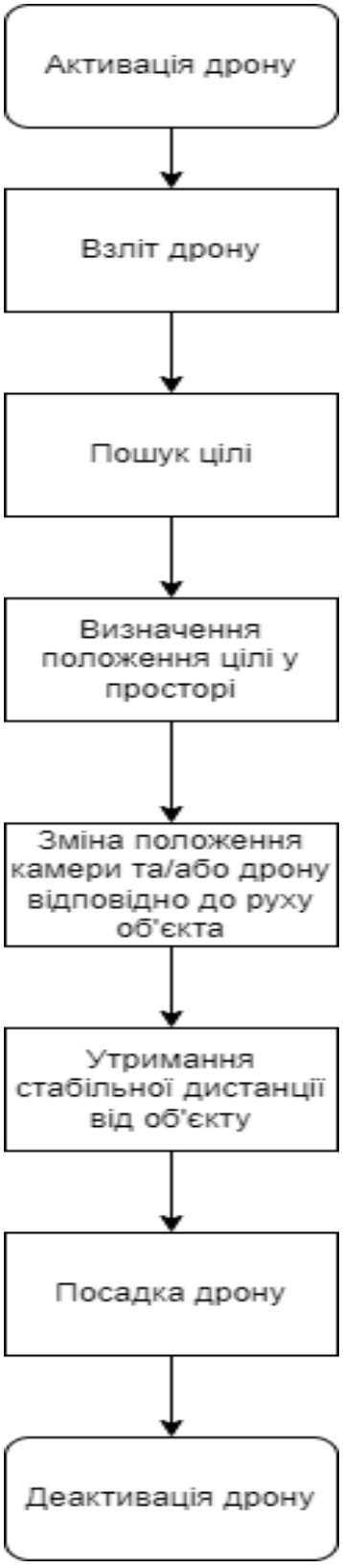
Метод відстеження руху об’єкта з керуванням положення камери

Алгоритм використання (Схема принципова)

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ – 2024 р.



					ІАЛЦ.467200.006 Д3	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Бабич В.Ю.*			**Метод відстеження руху об'єкта з керуванням положення камери**	*Літ.* / *Аркуш* / *Аркушів*
Перевірив		*Гордієнко Ю.Г..*				*1* / *1*
Н. Контр.		*Виноградов Ю.М.*			***Алгоритм використання (Схема функціональна)***	КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-02
Затв.						

ДОДАТОК 4

Метод відстеження руху об’єкта з керуванням положення камери

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 17

Київ – 2024 р.

main.py

```
import logging
from collections import OrderedDict
from threading import Thread
from time import sleep, time
from typing import Any, Dict, List, Tuple

import airsim
import cv2
import numpy as np
from airsim import YawMode
from airsim.types import CameraInfo, ImageResponse
from PIL import Image

logger = logging.getLogger(__name__)
logger_format = "%(asctime)s [%(levelname)s] %(message)s"
logging.basicConfig(
    level=logging.INFO,
    format=logger_format,
    handlers=[logging.FileHandler("airsim.log", mode="w"), log-
ging.StreamHandler()],
)


def radians_to_degrees(radians):
    return radians * 180.0 / np.pi


def degrees_to_radians(degrees):
    return degrees * np.pi / 180.0


class OpticalFlow:
    def __init__(
        self,
        winSize: Tuple[int] = (31, 31),
        maxLevel: int = 3,
        termcrit: Tuple[Any] = None,
    ):
```

```
        self.winSize = winSize
        self.maxLevel = maxLevel
        self.termcrit = (
            termcrit
            if termcrit
            else (cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS, 20,
0.03)
        )

        self.prev_img = None
        self.prev_pts = []

    def initialize(self, image: np.ndarray, x: float, y: float) -> None:
        self.prev_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        self.prev_pts = [np.array((x, y)).reshape((1, 2)).astype(np.float32)]

    def track(self, image: np.ndarray) -> None:
        next_img = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

        next_pts, status, err = cv2.calcOpticalFlowPyrLK(
            self.prev_img,
            next_img,
            self.prev_pts[-1],
            None,
            winSize=self.winSize,
            maxLevel=self.maxLevel,
            criteria=self.termcrit,
        )
        next_pts = next_pts[status.flatten() == 1]

        self.prev_img = next_img
        self.prev_pts = [next_pts]

    def target(self) -> Tuple[float, float]:
        if len(self.prev_pts[0]) == 0:
            return None
        else:
            return self.prev_pts[-1][0]

    def reset(self) -> None:
        self.prev_img = None
        self.prev_pts = []
```

```
    def initialized(self) -> bool:
        return self.prev_img is not None and len(self.prev_pts) > 0

    def found(self) -> bool:
        return len(self.prev_pts[0]) > 0

class Experiment:
    def __init__(
        self,
        drone_name: str = "Drone",
        target_regex: str = ".*Sphere.*",
        target_object_id: int = 20,
        target_color: tuple = (146, 52, 70),
    ):
        self.drone_name = drone_name
        self.target_regex = target_regex
        self.target_object_id = target_object_id
        self.target_color = target_color

        #self.initial_drone_pose = airsim.Pose(
            #airsim.Vector3r(0, 0, 0),
            #airsim.to_quaternion(0, 0, degrees_to_radians(180)),
        #)

        self.client_airsim = airsim.MultirotorClient()
        self.client_airsim.confirmConnection()

        self.client_images = airsim.MultirotorClient()
        self.client_images.confirmConnection()

        self.client_odometry = airsim.MultirotorClient()
        self.client_odometry.confirmConnection()

        self.image_callback_thread = Thread(
            target=self.repeat_timer_image_callback,
            args=(self.image_callback, 0.02),
            daemon=True,
        )
        self.odometry_callback_thread = Thread(
            target=self.repeat_timer_odometry_callback,
            args=(self.odometry_callback, 0.02),
            daemon=True,
        )
```

```
self.tracking_callback_thread = Thread(
    target=self.repeat_timer_tracking_callback,
    args=(self.tracking_callback, 0.02),
    daemon=True,
)
self.moving_callback_thread = Thread(
    target=self.repeat_timer_moving_callback,
    args=(self.moving_callback, 0.02),
    daemon=True,
)
self.visualize_callback_thread = Thread(
    target=self.repeat_timer_visualize_callback,
    args=(self.visualize_callback, 0.02),
    daemon=True,
)

self.is_image_thread_active = False
self.is_odometry_thread_active = False
self.is_tracking_thread_active = False
self.is_moving_thread_active = False
self.is_visualize_thread_active = False

self.trackers = {
    "BOOSTING": cv2.legacy.TrackerBoosting_create(),
    "MIL": cv2.TrackerMIL_create(),
    "KCF": cv2.TrackerKCF_create(),
    "CSRT": cv2.TrackerCSRT_create(),
    "MedianFlow": cv2.legacy.TrackerMedianFlow_create(),
    "TLD": cv2.legacy.TrackerTLD_create(),
    "MOSSE": cv2.legacy.TrackerMOSSE_create(),
}

#self.tracker = OpticalFlow()
self.tracker = self.trackers["BOOSTING"]
#self.tracker = self.trackers["KCF"]
#self.tracker = self.trackers["CSRT"]
#self.tracker = self.trackers["MedianFlow"]
#self.tracker = self.trackers["TLD"]
#self.tracker = self.trackers["MOSSE"]
self.tracker_status = "Not initialized"

# Set the camera requests
self.airsim_camera_requests = OrderedDict(
```

```
            rgb=airsim.ImageRequest(
                camera_name="rgb",
                image_type=airsim.ImageType.Scene,
                pixels_as_float=False,
                compress=False,
            ),
            seg=airsim.ImageRequest(
                camera_name="seg",
                image_type=airsim.ImageType.Segmentation,
                pixels_as_float=False,
                compress=False,
            ),
        )

        self.img_rgb = None
        self.img_seg = None

        self.target_x = None
        self.target_y = None

        self.tracking_x = None
        self.tracking_y = None

        self.drone_state = None

        self.img_counter = 0

        # Set the segmentation object color
        self.client_airsim.simSetSegmentationObjectID(
            mesh_name=target_regex, object_id=target_object_id, is_name_regex=True
        )

        self._print_camera_info()

    def _print_camera_info(self):
        # Get the camera information
        resp =
self.client_images.simGetImages(requests=[self.airsim_camera_requests["rgb"]])[
            0
        ]
        self.rgb_height, self.rgb_width = resp.height, resp.width

        logger.info(f"Camera RGB resolution: {self.rgb_width}x{self.rgb_height}")
```

```
    def initialize_drone(self):
        pass
        #Disarm the drone
        #self.client_airsim.armDisarm(False, vehicle_name=self.drone_name)
        #logger.info(f"Drone disarmed")

        # Disable API control
        #self.client_airsim.enableApiControl(False, vehicle_name=self.drone_name)
        #logger.info(f"API control disabled")

        # Enable API control
        #self.client_airsim.enableApiControl(True, vehicle_name=self.drone_name)
        #logger.info(f"API control enabled")

        # Set the initial drone pose
        #self.client_airsim.simSetVehiclePose(
            #self.initial_drone_pose, ignore_collision=False, vehi-
cle_name=self.drone_name)
        #logger.info(f"Drone pose set to: {self.initial_drone_pose}")

        # Arm the drone
        #self.client_airsim.armDisarm(True, vehicle_name=self.drone_name)
        #logger.info(f"Drone armed")

        # Take off
        self.client_airsim.takeoffAsync(vehicle_name=self.drone_name).join()
        logger.info(f"Drone took off")


    def read_images(self, include_segmenation: bool = True):
        requests = OrderedDict()

        if include_segmenation == True:
            requests["seg"] = self.airsim_camera_requests["seg"]

        requests["rgb"] = self.airsim_camera_requests["rgb"]
        logger.debug(f"Requesting {len(requests)} images")

        responses: Dict[str, ImageResponse] = OrderedDict(
            zip(
                list(requests.keys()),
                self.client_images.simGetImages(requests=list(requests.values())),
```

```
            )
        )
        logger.debug(f"Got {len(responses)} images")

        response = responses["rgb"]
        img1d = np.frombuffer(
            response.image_data_uint8, dtype=np.uint8
        )  # get numpy array
        img_rgb = img1d.reshape(
            response.height, response.width, 3
        )  # reshape array to 4 channel image array H X W X 3

        if include_segmenation:
            response = responses["seg"]
            img1d = np.frombuffer(
                response.image_data_uint8, dtype=np.uint8
            )  # get numpy array
            img_seg = img1d.reshape(
                response.height, response.width, 3
            )  # reshape array to 4 channel image array H X W X 3
            img_seg = cv2.cvtColor(img_seg, cv2.COLOR_RGB2BGR)

            # Resize the segmentation image to the same size as the RGB image
            img_seg = cv2.resize(img_seg, (self.rgb_width, self.rgb_height), interpola-
tion=cv2.INTER_NEAREST)

            target_pts = np.where(np.all(img_seg == self.target_color, axis=-1))

            if len(target_pts[0]) == 0 or len(target_pts[1]) == 0:
                x = None
                y = None
            else:
                x = target_pts[0].mean()
                y = target_pts[1].mean()
        else:
            img_seg = None
            x = None
            y = None

        return dict(
            img_rgb=img_rgb,
            img_seg=img_seg,
            x=x,
```

```
            y=y,
        )

    def image_callback(self):

        response = self.read_images(include_segmenation=True)

        self.img_rgb = response["img_rgb"]
        logger.info(f"Received RGB image: {self.img_rgb.shape}")
        self.img_seg = response["img_seg"]
        logger.debug(f"Received segmentation image: {self.img_seg.shape}")

        # Save image from camera
        img =
self.client_images.simGetImages(requests=[self.airsim_camera_requests["rgb"]])[0]
        if self.img_rgb is None:
            print("Failed to capture image")
        else:
            img1d = np.frombuffer(img.image_data_uint8, dtype=np.uint8)
            img_rgb = img1d.reshape((img.height, img.width, 3))
            image = Image.fromarray(img_rgb)
            image.save(f"Dataset/{self.img_counter}.png")
            self.img_counter += 1
            if self.img_counter == 4999:
                logger.info("STOP STOP STOP STOP STOP")
        logger.info(f"Saved RGB image {self.img_counter}")



        self.target_x = response["y"]
        self.target_y = response["x"]
        logger.debug(f"Target at: {self.target_x}, {self.target_y}")



    def draw_multiline_text(
        self, img, text, color, font, font_scale, thickness, line_type
    ):
        lines = text.split("\n")
        line_width, line_height = cv2.getTextSize(lines[0], font, font_scale, thick-
ness)[0]
        x_offset = round(line_width * 0.1)  # add some padding
        y_offset = round(line_height * 0.4) # add some padding
        for i, line in enumerate(lines, 1):
            img = cv2.putText(
```

```
                img,
                line,
                (x_offset, y_offset + i * (y_offset + line_height)),
                font,
                font_scale,
                color,
                thickness,
                line_type,
            )
        return img

    def visualize_callback(self):
        img_rgb = self.img_rgb.copy()
        img_seg = self.img_seg.copy()

        text_color = (0, 0, 0)
        target_color = (0, 255, 0)
        tracking_color = (255, 0, 0)
        font = cv2.FONT_HERSHEY_SIMPLEX
        thickness = 2
        font_scale = 0.75
        line_type = cv2.LINE_8

        info_schema = (
            "Image: {width}x{height}\n"
            "Target at: ({target_x}, {target_y})\n"
            "Tracking at: ({tracking_x}, {tracking_y})\n"
            "Yaw rate: {yaw_rate}\n"
            "Tracking: {tracking}\n"
        )

        width, height = img_rgb.shape[1], img_rgb.shape[0]
        target_x = round(self.target_x) if self.target_x is not None else None
        target_y = round(self.target_y) if self.target_y is not None else None
        tracking_x = round(self.tracking_x) if self.tracking_x is not None else None
        tracking_y = round(self.tracking_y) if self.tracking_y is not None else None
        yaw_rate = "%.3f" %
self.drone_state.kinematics_estimated.angular_velocity.z_val
        tracking = self.tracker_status

        info = info_schema.format(
            width=width,
            height=height,
```

```
            target_x=target_x,
            target_y=target_y,
            tracking_x=tracking_x,
            tracking_y=tracking_y,
            yaw_rate=yaw_rate,
            tracking=tracking,
        )

        # Combine images
        img = np.concatenate([img_rgb, img_seg], axis=1)

        # draw multiline text using opencv
        img = self.draw_multiline_text(img, info, text_color, font, font_scale, thickness,
line_type)

        has_target = self.target_x is not None and self.target_y is not None
        has_tracking = self.tracking_x is not None and self.tracking_y is not None

        if has_target:
            img = cv2.circle(
                img, (int(self.target_x), int(self.target_y)), 5, target_color, -1
            )
            img = cv2.putText(
                img,
                f"target",
                (int(self.target_x), int(self.target_y)),
                font,
                font_scale,
                target_color,
                thickness,
                line_type,
            )

        if has_tracking:
            img = cv2.circle(
                img, (int(self.tracking_x), int(self.tracking_y)), 5, tracking_color, -1
            )
            text_size = cv2.getTextSize("tracking", font, font_scale, thickness)[0]
            img = cv2.putText(
                img,
                f"tracking",
                (int(self.tracking_x) - text_size[0], int(self.tracking_y) + text_size[1]),
                font,
```

```
                font_scale,
                tracking_color,
                thickness,
                line_type,
            )

        cv2.imshow(self.level_name, img)
        cv2.waitKey(1)

    def odometry_callback(self):
        self.drone_state = self.client_odometry.getMultirotorState()

    def takeoff_async(self):
        self.client_airsim.takeoffAsync(self.drone_name).join()

    def move_drone(
        self, roll: float, pitch: float, yaw: float, throttle: float, duration: float
    ):
        self.client_airsim.moveByManualAsync(
            vx_max=1e9,
            vy_max=1e9,
            z_min=-1e9,
            duration=duration,  # v(x,y,z) represents the velocity in the global coordinate
system
            drivetrain=airsim.DrivetrainType.ForwardOnly,
            yaw_mode=YawMode(is_rate=True, yaw_or_rate=10.0 * (1 if yaw > 0 else -
1)),
        )
        self.client_airsim.moveByRC(
            rcdata=airsim.RCData(
                roll=roll,
                throttle=throttle,
                yaw=yaw,
                pitch=pitch,
                is_initialized=False,
                is_valid=True,
            )
        )
        sleep(duration)

    def wait_for_images(self):
        while self.img_rgb is None:
            sleep(0.1)
```

```
            logger.debug("Waiting for images")

        # Sleep for a while
        sleep(2)

    def repeat_timer_image_callback(self, callback, period_sec):
        self.is_image_thread_active = True
        while self.is_image_thread_active:
            callback()
            sleep(period_sec)

    def repeat_timer_odometry_callback(self, callback, period_sec):
        self.is_odometry_thread_active = True
        while self.is_odometry_thread_active:
            callback()
            sleep(period_sec)

    def repeat_timer_tracking_callback(self, callback, period_sec):
        self.is_tracking_thread_active = True
        while self.is_tracking_thread_active:
            callback()
            sleep(period_sec)

    def repeat_timer_moving_callback(self, callback, period_sec):
        self.is_moving_thread_active = True
        while self.is_moving_thread_active:
            callback()
            sleep(period_sec)

    def repeat_timer_visualize_callback(self, callback, period_sec):
        self.is_visualize_thread_active = True
        while self.is_visualize_thread_active:
            callback()
            sleep(period_sec)

    def moving_callback(self):
        if self.tracking_x is not None and self.tracking_y is not None:
            logger.debug(f"Moving callback")

            # just move yaw using x
            roll = 0.0
            pitch = 0.0
            throttle = 0.0
```

```
        #dx = 0
        #yaw = 0
        #duration = 0
        dx = self.tracking_x - self.rgb_width / 2  # -width/2 to width/2
        yaw = dx / self.rgb_width * 2  # -1 to 1
        duration = 0.010  # 10 ms

        self.move_drone(roll, pitch, yaw, throttle, duration)

    def tracking_callback(self):
        logger.debug(f"Tracking callback")

        if not self.tracker.initialized():
            logger.debug(f"Initializing tracker")

            if self.target_x is None or self.target_y is None:
                logger.debug(f"Target not set, can not initialize tracker")
                self.tracker_status = "Target not set"
                return

            logger.debug(f"Using frame {self.img_rgb.shape} to initialize tracker")
            self.tracker.initialize(
                self.img_rgb,
                self.target_x,
                self.target_y,
            )
            logger.debug(f"Tracker initialized")
            self.tracker_status = "Initialized"

        else:
            self.tracker.track(self.img_rgb)
            logger.debug(f"Tracker tracked")

            if self.tracker.found():
                logger.debug(f"Target found at: {self.tracker.target()}")
                self.tracker_status = "Found"
                target = self.tracker.target()
                self.tracking_x = target[0]
                self.tracking_y = target[1]

            else:
                logger.debug(f"Target not found")
```

```
            self.tracker_status = "Not found"
            self.tracker.reset()
            self.tracking_x = None
            self.tracking_y = None
            logger.debug(f"Tracker reset")

    def load_level(self, level_name, sleep_sec=2.0):
        self.level_name = level_name
        self.client_airsim.simLoadLevel(self.level_name)
        self.client_airsim.confirmConnection()  # failsafe
        sleep(sleep_sec)  # let the environment load completely
        logger.info(f"Level {self.level_name} loaded")

    def start_image_callback_thread(self):
        if not self.is_image_thread_active:
            self.is_image_thread_active = True
            self.image_callback_thread.start()
            logger.info("Started image callback thread")

    def stop_image_callback_thread(self):
        if self.is_image_thread_active:
            self.is_image_thread_active = False
            self.image_callback_thread.join()
            logger.info("Stopped image callback thread.")

    def start_odometry_callback_thread(self):
        if not self.is_odometry_thread_active:
            self.is_odometry_thread_active = True
            self.odometry_callback_thread.start()
            logger.info("Started odometry callback thread")

    def stop_odometry_callback_thread(self):
        if self.is_odometry_thread_active:
            self.is_odometry_thread_active = False
            self.odometry_callback_thread.join()
            logger.info("Stopped odometry callback thread.")

    def start_tracking_callback_thread(self):
        if not self.is_tracking_thread_active:
            self.is_tracking_thread_active = True
            self.tracking_callback_thread.start()
            logger.info("Started tracking callback thread")
```

```
def stop_tracking_callback_thread(self):
    if self.is_tracking_thread_active:
        self.is_tracking_thread_active = False
        self.tracking_callback_thread.join()
        logger.info("Stopped tracking callback thread.")

def start_moving_callback_thread(self):
    if not self.is_moving_thread_active:
        self.is_moving_thread_active = True
        self.moving_callback_thread.start()
        logger.info("Started moving callback thread")

def stop_moving_callback_thread(self):
    if self.is_moving_thread_active:
        self.is_moving_thread_active = False
        self.moving_callback_thread.join()
        logger.info("Stopped moving callback thread.")

def start_visualize_callback_thread(self):
    if not self.is_visualize_thread_active:
        self.is_visualize_thread_active = True
        self.visualize_callback_thread.start()
        logger.info("Started visualize callback thread")

def stop_visualize_callback_thread(self):
    if self.is_visualize_thread_active:
        self.is_visualize_thread_active = False
        self.visualize_callback_thread.join()
        logger.info("Stopped visualize callback thread.")

def run(self):
    self.load_level("DefaultMap")
    self.initialize_drone()
    self.start_image_callback_thread()
    self.start_odometry_callback_thread()

    self.wait_for_images()
    self.start_visualize_callback_thread()
    self.start_tracking_callback_thread()
    self.start_moving_callback_thread()

    sleep(30000)
```

```
        self.stop_image_callback_thread()
        self.stop_odometry_callback_thread()
        self.stop_tracking_callback_thread()
        self.stop_moving_callback_thread()

def main():
    experiment = Experiment()
    experiment.run()

if __name__ == "__main__":
    main()
```

test.py

```
import airsim
print(airsim.__version__)
```

nn_model.py

```
from ultralytics import YOLO

model = YOLO('yolov8n.pt')

model.train(data='dataset.yaml', epochs=100, imgsz=1000)
```

dataset.yaml

```
path: ./Dataset
train: train
val: val
test: test

nc: 1
names: ['ball']
```

setting.json

```
{
    "SettingsVersion": 1.2,
    "SimMode": "Multirotor",
    "CameraDefaults": {
        "CaptureSetting": [{
            "Width": 640,
            "Height": 480,
            "FOV_Degrees": 90
        }]
```

```
},
"Vehicles" : {
   "Drone" : {
      "VehicleType" : "SimpleFlight",
      "X": 0, "Y": 0, "Z": 0,
      "AllowAPIAlways": true,
      "RC": { "RemoteControlID": 0, "AllowAPIWhenDisconnected": true },
      "AutoCreate" : true,
      "Cameras" : {
         "rgb": {
            "CaptureSettings" : [{
               "ImageType" : 0,
               "Width" : 640,
               "Height" : 480
            }],
            "X": 0.50, "Y": 0.00, "Z": 0.10,
            "Pitch": 0.0, "Roll": 0.0, "Yaw": 0.0
         },
         "seg": {
            "CaptureSettings" : [{
               "ImageType" : 5,
               "Width" : 160,
               "Height" : 120
            }],
            "X": 0.50, "Y": 0.00, "Z": 0.10,
            "Pitch": 0.0, "Roll": 0.0, "Yaw": 0.0
         }
      }
   }
}
```