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: Система захоплення та супроводження об’єкту для автопілота

Виконав: студент 4 курсу, групи ІО-04
(шифр групи)

Кубишка Юрій Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст.викл., Алєнін О. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Іваніщев Б.В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент к.т.н., доцент кафедри ІСТ Деведжіогуллари А.В. _____
(посада, вчене звання, науковий ступінь, 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

Кубишки Юрія Сергійовича

1. Тема проєкту Система захоплення та супроводження об’єкту для автопілота
керівник проєкту ст.викл. Алєнін Олег Ігорович,
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Затверджені наказом по університету від 27.05.2024 року № 2112-с
2. Термін здачі студентом закінченого проєкту 03 червня 2024 р.
3. Вихідні дані до проєкту технічна документація, теоретичні дані
4. Зміст пояснювальної записки (перелік завдань, які потрібно розробити)
   Розділ 1 Огляд систем захоплення та супроводження об’єктів
   Розділ 2 Огляд алгоритмів та технологій для роботи системи
   Розділ 3 Деталі розробки системи
   Розділ 4 Дослідження та аналіз розробленої системи

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Іваніщев Б.В.		

7. Дата видачі завдання «30» жовтня 2023 р.

Календарний план

№ п/п	Назва етапу дипломного проєкту	Строк виконання етапу проєкту	Примітка
1	Затвердження теми проєкту	30.10.2023	
2	Вивчення та аналіз завдання	01.11.2023-15.04.2024	
3	Розробка архітектури та загальної структури системи	15.04.2024-05.05.2024	
4	Розробка структур окремих підсистем	15.04.2024-05.05.2024	
5	Програмна реалізація системи	01.05.2024-24.05.2024	
6	Оформлення пояснювальної записки	06.05.2024-30.05.2024	
7	Захист програмного продукту	24.05.2024	
8	Передзахист	03.06.2024	
9	Захист	17.06.2024	24.06.2024

Студент-дипломник _____ Юрій КУБИШКА
(підпис)

Керівник проєкту _____ Олег АЛЄНІН
(підпис)

АНОТАЦІЯ

Дана робота спрямована на вирішення проблеми трекінгу об’єкта в системах з обмеженими обчислювальними потужностями. На основі аналізу існуючих підходів до трекінгу об’єктів, було обрано кілька алгоритмів, що лягли в основу розробки системи. Розроблений програмний продукт дає можливість користувачеві визначати розташування об’єкта у відео-потоці. Проведено дослідження системи на стабільність роботи у різних сценаріях використання. На основі отриманих результатів сформовано рекомендації по застосуванню. Програмний продукт реалізовано мовою Python.

Ключові слова: трекінг, алгоритми стеження, computer vision, threshold, Kalman filter, adaptive threshold, Python.

ANNOTATION

This work is aimed at solving the problem of object tracking in systems with limited computing power. Based on the analysis of existing approaches to object tracking, several algorithms were chosen that formed the basis of the system development. The developed software product enables the user to determine the location of the object in the video stream. A study of the stability of the system in various scenarios of use was carried out. Based on the obtained results, recommendations for application were formed. The software product is implemented in the Python language.

Keywords: tracking, tracking algorithms, computer vision, Kalman filter, adaptive threshold, Python.

Довідки	Формат	Значення	Найменування	Кіл. листів	№ екземпляра	Додаток
			Документація загальна			
			Знову розроблена			
	А4	*ІАЛЦ.ІВ9215.002 ТЗ*	Система захоплення та	4		
			супроводження об'єкту для			
			автопілота			
			Технічне завдання			
	А4	*ІАЛЦ.ІВ9215.003 ПЗ*	Система захоплення та	58		
			супроводження об'єкту для			
			автопілота			
			Пояснювальна записка			
	А4	*ІАЛЦ.ІВ9215.004 Д1*	Система захоплення та	1		
			супроводження об'єкту для			
			автопілота			
			Структурна схема			
	А4	*ІАЛЦ.ІВ9215.005 Д2*	Система захоплення та	2		
			супроводження об'єкту для			
			автопілота			
			Принципова схема			
	А4	*ІАЛЦ.ІВ9215.006 Д3*	Система захоплення та	1		
			супроводження об'єкту для			
			Автопілота			
			Функціональна схема			
	А4	*ІАЛЦ.ІВ9215.007 Д4*	Система захоплення та	4		
			супроводження об'єкту для			
			Автопілота			
			Текст програмного коду			

Зм.	*Лист*	*№ докум.*	*Підп.*	*Дата*	
Розроб.		Кубишка Ю.С.			
Перев.		Алєнін О.І			
Реценз.		Деведжіогулари А.В.			
Н. Контр.		Іваніщев Б.В.			
Затверд.		Стіренко С.Г.			

ІАЛЦ.467200.001 ОА

Система захоплення та супроводження об'єкту для автопілота
Опис альбому

Літ.			Аркуш	Аркушів
			1	1

КПІ ім. Ігоря Сікорського
ФІОТ Група ІО-04

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система захоплення та супроводження об’єкту для автопілота*»

Київ – 2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи захоплення та супроводження об’єкту для автопілота.

Областю застосування програмного забезпечення є будь-яка сфера, де необхідно визначати координати об’єкта слідування в умовах динамічної зміни фону на відео-потоці. Може бути застосовано на автомобілях, дронах, літаках.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня “бакалавр комп’ютерних систем та мереж”, який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України “Київський Політехнічний інститут ім. Ігоря Сікорського”.

3. МЕТА І ПРИЗНАЧЕННЯ

Метою та призначенням даної роботи є розробка крос-платформного програмного забезпечення, що дозволить вести трекінг об’єкта в системах з обмеженими обчислювальними ресурсами.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки дипломного проєкту є науково-технічна література, офіційні документації, публікації та статті в мережі Інтернет на дану тему.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

Розроблюваний продукт має виконувати такі вимоги:

- Простий і інтуїтивно зрозумілий інтерфейс для взаємодії з користувачем.
- Швидке визначення положення об’єкта на відео-потоці.
- Можливість переобирати об’єкт для подальшого відстеження

5.2. Вимоги до програмного забезпечення

Операційна система:

- Windows: Windows 7, 8, 10, або новіше
- Linux: Сучасні дистрибутиви, такі як Ubuntu 20.04 або новіше
- macOS: macOS 10.15 (Catalina) або новіше

Інструменти та бібліотеки:

- Python: Версія 3.6 або новіша (рекомендується Python 3.8 або новіша)
- OpenCV: Версія 4.5.1 або новіше
- Matplotlib(опціонально): Для візуалізації даних та відлагодження

5.3. Вимоги до апаратної частини

Мінімальні параметри системи:

- Двоядерний процесор (Intel Core i3 або еквівалентний).
- Оперативна пам'ять (RAM): 4 ГБ.
- Інтегрована відеокарта з підтримкою OpenGL 2.0 або вище.
- Диск (HDD/SSD): 1 ГБ вільного місця для встановлення та збереження тимчасових файлів.
- Камера: Веб-камера.

Рекомендовані параметри системи:

- Чотириядерний процесор (Intel Core i5 або вище).
- Оперативна пам'ять (RAM): 8 ГБ або більше.
- Дискретна відеокарта (NVIDIA або AMD) з підтримкою CUDA або OpenCL для пришвидшення обробки відео.
- Диск (HDD/SSD): 10 ГБ вільного місця на SSD для швидшого доступу до даних.
- Камера/Відео: Високоякісна камера високої роздільної здатності (HD або Full-HD).

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Вибір теми проєкту	01.09.2023-30.10.2023
Вивчення та аналіз літературних джерел за темою дипломного проєкту	01.11.2023-11.02.2024
Розробка архітектури та загальної структури системи	11.02.2024-04.03.2024
Розробка програмного забезпечення для трекінгу об’єкта з використанням алгоритмів глибокого навчання	04.03.2024-02.04.2024
Розробка програмного забезпечення для інтуїтивної взаємодії користувача з системою	03.04.2024-07.04.2024
Розробка програмного забезпечення для трекінгу об’єкта з використанням мінімальної кількості ресурсів	15.04.2024-05.05.2024
Реалізація прототипу системи супроводження та її тестування	06.05.2024-17.05.2024
Оформлення пояснювальної записки	06.05.2024-08.06.2024
Передзахист	04.06.2024
Захист дипломного проєкту	17.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система захоплення та супроводження об'єкту для автопілота*»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

Системи захоплення та супроводження об'єктів – СЗСО;

Програмне забезпечення – ПЗ;

Region of interest – ROI;

Convolutional neural network – CNN;

Recurrent neural networks – RNN;

ВСТУП

Стрімкий розвиток технологій дає можливість вирішити величезну кількість буденних справ переклавши відповідальність за їх виконання на машини. На відміну від людей вони не мають поганого настрою, страху, амбіцій, а головне – моралі. Машина, всього лише виконує алгоритм, що був їй прописаний. Саме тому, розвиток автономних систем, що здатні майже повністю замінити людину – пріоритетна задача.

Автопілот як складову частину автономних систем впроваджують для громадського транспорту, особистих авто, господарської техніки, та навіть, для роботів-пилососів.

Для покращення роботи автопілота в систему додають алгоритми, що дозволяють реагувати на зміни навколо, це дозволяє:

- оминати перешкоди;
- мінімізувати ризики травмування людей;
- уникати критичних ситуацій на дорогах;
- вести стеження за об’єктами;
- знешкоджувати ворожі цілі.

В даній роботі буде розглянуто існуючі алгоритми трекінгу об’єктів та реалізована спроба створити алгоритм для оптимальної роботи СЗСО на машинах з обмеженою обчислювальною потужністю.

РОЗДІЛ 1. ОГЛЯД СИСТЕМ ЗАХОПЛЕННЯ ТА СУПРОВОДЖЕННЯ ОБ'ЄКТУ ДЛЯ АВТОПІЛОТА

1.1. Актуальність та доцільність розроблення системи

Система захоплення та супроводження об'єкту для автопілота – це набір технологій, що дозволяють автопілоту автомобіля або іншого технологічного засобу фокусуватись на конкретному об'єкті. Це можуть бути інші транспортні засоби, пішоходи, перешкоди або будь-що, що може виділятись на фоні за допомогою наявних сенсорних засобів.

СЗСО для автопілотів є важливою складовою безпілотних засобів спостереження та/або ураження. Існує проблема керування такими пристроями в умовах радіо-електронної боротьби, цей недолік можна частково нівелювати за рахунок використання автономних засобів керування.

Існуючі системи захоплення та супроводження об'єктів (Target Acquisition and Tracking Systems) є важливими компонентами як військових, так і цивільних технологій. Вони використовуються для виявлення, ідентифікації та супроводження різних цілей, таких як літаки, ракети, транспортні засоби.

Існує декілька реалізацій СЗСО в складі принципово різних систем, кожна з яких має свою специфіку, наприклад:

- Радіолокаційні системи (Radar Systems). Використовують радіохвилі для виявлення і визначення місцезнаходження об'єктів. Вони є критично важливими в військовій авіації та протиповітряній обороні для виявлення літаків, ракет і інших повітряних об'єктів.
- Оптико-електронні системи (Electro-Optical Systems) Використовують світлові хвилі (видиме світло, інфрачервоні хвилі) для виявлення і супроводження об'єктів. Вони широко застосовуються в цивільних та військових галузях для спостереження, моніторингу та розвідки.
- Системи з підтримкою штучного інтелекту (AI-assisted Systems) Здатні автоматично виявляти, ідентифікувати та супроводжувати

об'єкти з високою точністю і швидкістю. Вони застосовуються в різних сферах, включаючи безпеку та оборону.

- Системи протиракетної оборони (Missile Defense Systems). Ці системи спеціалізуються на виявленні, супроводженні і знищенні ворожих ракет до того, як вони досягнуть своїх цілей. Вони є важливими для захисту від ракетних атак.
- Морські системи (Naval Systems) Використовуються для виявлення і супроводження морських об'єктів, таких як судна та підводні човни. Вони є важливою частиною морської безпеки та оборони.
- Системи для моніторингу кордонів і безпеки (Border and Security Monitoring Systems) Ці системи забезпечують виявлення і супроводження об'єктів, що перетинають кордони, допомагаючи в забезпеченні безпеки та запобіганні нелегальної діяльності.
- Системи захисту від безпілотників (Anti-Drone Systems). Призначені для виявлення, супроводження і нейтралізації дронів, які можуть становити загрозу для безпеки.

1.2. Алгоритми на основі кореляцій (Correlation Filter-Based Tracking)

1.2.1 Алгоритм KCF (Kernelized Correlation Filters):

Швидкий і ефективний метод, який використовує ядрові трюки для обчислення кореляцій у просторі ознак.

Алгоритм роботи:

1) Створюється шаблон об'єкта на основі початкового зображення. Для цього виділяється область інтересу, яка визначає об'єкт, що відслідковується.
2) Використовується кореляційний фільтр, який обчислює схожість між шаблоном і поточним зображенням. Основна ідея полягає в тому, щоб знайти місце, де кореляція максимальна, що вказує на

положення об'єкта. Кореляція здійснюється у частотній області за допомогою швидкого перетворення Фур'є.

3) KCF використовує ядровий трюк для обчислення кореляцій у просторі ознак, що дозволяє ефективно обробляти великі об'єми даних і підвищує точність. Ядровий трюк полягає в обчисленні кореляції не в початковому просторі ознак, а в багатовимірному просторі, що забезпечує краще розрізнення між різними об'єктами.
4) Після кожного кроку кореляції шаблон оновлюється з урахуванням нової інформації. Це дозволяє алгоритму адаптуватися до змін у зовнішньому вигляді об'єкта під час його руху.
5) На основі отриманих кореляційних значень визначається положення об'єкта в новому кадрі. Місце з найбільшою кореляцією вважається новим центром об'єкта.
6) Алгоритм оновлює свою модель об'єкта, враховуючи нове положення і зовнішній вигляд, щоб покращити відстеження в наступних кадрах[1].

Переваги:

- KCF є дуже швидким методом, який може працювати в режимі реального часу, що робить його придатним для застосувань, де необхідна висока продуктивність.
- Завдяки використанню ядрового трюку, KCF демонструє високу точність у відстеженні об'єктів навіть у складних умовах.
- Метод ефективно обробляє великі об'єми даних, що дозволяє застосовувати його в різних сценаріях.

Недоліки:

- KCF може бути чутливим до значних змін зовнішнього вигляду об'єкта, таких як деформації, обертання або змінення освітлення.

– KCF може вимагати значних обчислювальних ресурсів, особливо при обробці великих зображень.
– У складних сценах з великою кількістю схожих об'єктів KCF може втрачати точність, оскільки метод покладається на кореляцію з шаблоном.

1.2.2 CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability)

Покращена версія KCF, яка враховує просторову надійність.

Алгоритм роботи:

1) Записуються багатоканальні характеристики об'єкта з вхідного зображення. Це можуть бути як канали кольору, так і інші просторові характеристики, такі як HOG (Histogram of Oriented Gradients).
2) Визначається надійність кожного каналу зображення. Це дозволяє алгоритму розподілити вагу кореляційного фільтра між різними каналами, залежно від їхньої надійності.
3) Використовуючи кореляційний фільтр, алгоритм обчислює схожість між багатоканальним шаблоном і поточним зображенням. Просторова надійність дозволяє коригувати кореляцію, приділяючи більше уваги надійним частинам шаблону.
4) На основі отриманих результатів алгоритм оновлює кореляційний фільтр, адаптуючи його до нових умов. Це включає оновлення вагової функції та багатоканального шаблону.
5) Місце з найбільшою кореляцією визначається як нове положення об'єкта в кадрі. Це положення враховується при подальшому оновленні шаблону[2].

Переваги:

- Покращена точність: Завдяки врахуванню просторової надійності та використанню багатоканальних характеристик, CSRT забезпечує високу точність трекінгу.
- Стійкість до змін: Алгоритм добре справляється зі змінами зовнішнього вигляду об'єкта, деформаціями та частковими оклюзіями.
- Гнучкість: Можливість використання різних каналів дозволяє CSRT адаптуватися до різних умов спостереження та характеристик об'єкта.

Недоліки:

- CSRT є більш ресурсозатратним, ніж KCF, і тому повільнішим. Реалізація алгоритму є складнішою через додаткові етапи обробки і адаптації шаблону.
- Використання багатоканальних характеристик і просторової надійності вимагає більше обчислювальних ресурсів, що може бути критично для реальних додатків у режимі реального часу.

1.2.3 Алгоритм MOSSE (Minimum Output Sum of Squared Error)

Застосовує кореляційні фільтри для відстеження об'єктів у відео послідовностях. Ідея полягає в покращені фільтра, який буде максимально збігатися із шаблоном об'єкта на кожному кадрі.

Фільтр h розраховується так, щоб мінімізувати різницю між шаблоном і зображенням об'єкта. Це досягається мінімізацією функції помилки:

$$E = \sum_i |I_i * h - G|^2 ,$$

де - вхідне зображення, *- функція згортки, G – бажаний вихід (гауссовий профіль).

Фільтр оновлюється в процесі виконання, що робить алгоритм більш стійким до змін освітлення, перекриттів та деформацій об'єкта.

Алгоритм роботи:

1) Вибирається початковий кадр, на якому визначається об'єкт для трекінгу. Цей кадр використовується для створення початкового шаблона та навчання фільтра.
2) Для кожного шаблона обчислюється бажаний вихід G, який зазвичай представляється у вигляді гауссового профілю, центрованого на об'єкті.
3) Після кожного кадру фільтр оновлюється з використанням нового зображення та з урахуванням попереднього стану фільтра, що дозволяє алгоритму адаптуватися до змін.

$$h = \alpha h_{new} + (1 - \alpha) h_{old},$$

де h — оновлений фільтр, h_{new} – фільтр, розрахований на основі нового кадру, h_{old} – попередній фільтр α - коефіцієнт навчання, який визначає, наскільки швидко фільтр адаптується до нових даних.

4) На кожному новому кадрі алгоритм обчислює кореляцію між поточним зображенням та фільтром, знаходячи положення об'єкта за допомогою максимального значення кореляції[3].

Переваги:

1) MOSSE є дуже швидким методом, здатним працювати в режимі реального часу.
2) Стійкий до змін освітлення, перекриттів та деформацій об'єкта.
3) Алгоритм легко реалізувати завдяки його математичній простоті.

Недоліки:

1) Алгоритм може бути менш ефективним при значних змінах масштабу об'єкта.

2) Може мати проблеми з відстеженням об'єктів у дуже складних сценах або при значних зміненнях форми об'єкта.

1.3. Алгоритми на основі глибокого навчання

Глибоке навчання дозволяє розробляти моделі, які можуть точно ідентифікувати об'єкти та відстежувати їх навіть у складних умовах. Основні методи глибокого навчання для відстеження об'єктів включають згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та їх комбінації.

Deep SORT є потужним інструментом для трекінгу об'єктів, що поєднує переваги глибокого навчання з класичними методами трекінгу, забезпечуючи високу точність і стабільність роботи в реальних умовах.

Deep SORT складається з двох основних частин:

1) Детектор об'єктів (Object Detector) Використовується для виявлення об'єктів на кожному кадрі відео. В сучасних реалізаціях часто використовуються глибокі нейронні мережі, такі як YOLO (You Only Look Once) або Faster R-CNN, для отримання високоякісних результатів детекції[4].
2) Трекер (Tracker) Відповідальний за зв’язок детекцій між кадрами і побудову траєкторій об'єктів. В основі трекера лежить алгоритм SORT (Simple Online and Realtime Tracking), який використовує лінійне передбачення та асоціацію за допомогою алгоритму Hungarian.

1.3.1 Hungarian Algorithm

Hungarian Algorithm або алгоритм Куна-Мункреса був розроблений Гарольдом Куном у 1955 році на основі попередніх робіт угорських математиків використовується для оптимальної асоціації нових спостережень з існуючими треками, забезпечує мінімізацію сумарної відстані між

прогнозованими та реальними позиціями об'єктів. Це критично важливо для підтримання точного і стабільного трекінгу, особливо у складних сценаріях з великою кількістю об'єктів та перешкодами. Основна ідея алгоритму полягає у знаходженні мінімального покриття матриці витрат шляхом комбінації нулів у матриці, що відповідає оптимальному призначенню.

Принцип роботи Hungarian Algorithm:

1) Створюється матриця розміром $n \times n$, де n — це максимальна кількість елементів у множинах. Елемент $C(i, j)$ вказує на вартість призначення об’єкта i до спостереження j.
2) З кожного рядка матриці витрат віднімається мінімальний елемент цього рядка.
3) З кожного стовпця матриці витрат віднімається мінімальний елемент цього стовпця.
4) Покриття нулів мінімальною кількістю горизонтальних та вертикальних ліній. Якщо кількість ліній дорівнює n – знайдено оптимальне призначення. Перехід до кроку 6.
5) Пошук найменшого елемента, який не покритий лініями.
6) Відняти цей елемент від всіх непокритих елементів, і додати його до елементів, покритих двома лініями. Повторити кроки 4 та 5, доки всі нулі не будуть покриті мінімальною кількістю ліній.
7) Використовуючи покриті нулі, визначається відповідність між об'єктами і спостереженнями[5].

1.3.2 Алгоритм FairMOT (Fair Multi-Object Tracking)

Один із найсучасніших підходів до трекінгу об'єктів, який вирішує проблему одночасного виявлення та трекінгу об'єктів в одному єдиному мережевому модулі.

Опис алгоритму відстеження:

1) Зчитування окремих кадрів з відео-послідовності, на яких необхідно виконати детекцію та відстеження об'єктів.
2) Вилучення ознак з вхідного зображення за допомогою нейронної мережі. Зазвичай використовуються такі мережі як ResNet або DLA (Deep Layer Aggregation) для отримання багатошарових ознак.
3) Мережа прогнозує heatmap'и, що представляють ймовірність присутності центру об'єкта в кожному пікселі зображення. Для кожного центру об'єкта алгоритм прогнозує координати меж об'єкта, що дозволяє визначити точне місцезнаходження об'єкта на зображенні.

4) Одночасно з детекцією об'єктів, мережа генерує векторні ознаки для кожного об'єкта. Ці ознаки використовуються для ідентифікації об'єктів та їхнього відстеження між кадрами.
5) Використовується евклідова або косинусова відстань для порівняння векторів ознак об'єктів у поточному кадрі з ознаками об'єктів у попередніх кадрах. На основі обчислених відстаней виконується асоціація нових детекцій з існуючими треками. Алгоритм асоціює об'єкти з найменшою відстанню між їхніми ознаками, щоб продовжити їхні треки.
6) Оновлення треків:
   – Якщо об'єкт був асоційований з треком, координати треку оновлюються на основі нових даних детекції.
   – Якщо новий об'єкт не був асоційований з існуючим треком, створюється новий трек для цього об'єкта.
   – Якщо трек не був асоційований з новими детекціями протягом певного числа кадрів, він видаляється.
7) алгоритм генерує координати детектованих об'єктів на кожному кадрі та відповідні треки, що дозволяє відслідковувати рух об'єктів у відеопослідовності[6].

1.3. Алгоритми на основі частинок

Методи, які використовують набір випадкових зразків (часток) для оцінювання стану об'єктів та їх траєкторій у просторі і часі. Основна ідея полягає в тому, що частки представляють різні можливі стани об'єктів, а їх ваги відображають ймовірність цих станів.

1.4.1 Алгоритм Particle Filter

Метод для оцінки стану динамічної системи, яка може бути нелінійною та негаусівською. Цей метод належить до класу Монте-Карло методів і використовує набір випадкових зразків (частинок) для представлення апостеріорного розподілу стану системи. Алгоритм широко використовується для трекінгу об'єктів завдяки своїй здатності ефективно працювати з нелінійними та шумовими моделями.

Алгоритм трекінгу з використанням Particle Filter:

1) Модель стану і спостереження

   Модель стану описує динаміку об'єкта, наприклад, його позицію та швидкість. Модель спостереження пов'язує вимірювання (наприклад, координати об'єкта на зображенні) з його станом.

2) Ініціалізація часток

   На першому кроці генерується набір часток на основі початкової оцінки положення об'єкта. Вага всіх часток на початку однакова.

3) Прогнозування

   Для кожної частки прогнозується новий стан об'єкта на основі моделі динаміки. Наприклад, використовується рівняння руху об'єкта з додаванням випадкового шуму.

4) Оновлення ваг

   Після отримання нових даних від сенсорів (наприклад, новий кадр відео), ваги часток оновлюються. Частки, які краще відповідають новим даним, отримують більшу вагу.

5) Пересемплінг

   Виконується пересемплінг, щоб сконцентрувати частки у зонах високої ймовірності. Частинки з малими вагами видаляються, а частинки з великими вагами копіюються.

6) Оцінка стану

   Після кількох ітерацій стан об'єкта оцінюється як середньозважене значення положень часток.

1.4.2. Алгоритм Joint Probabilistic Data Association (JPDA)

Метод для асоціації вимірювань з множинними об'єктами на основі ймовірнісного підходу, тобто допомагає визначити, які вимірювання до яких об'єктів належать, коли ми намагаємося відстежувати кілька цілей одночасно. алгоритм дозволяє системі відстежувати кілька цілей навіть у складних умовах, коли існує багато шуму [7].

Принцип роботи JPDA:

1) Отримання вимірювань від сенсорів $Z = \{z_1, z_2, \dots, z_M\}$;
2) Прогнозування положень об’єктів на основі попереднього стану і моделі руху.
3) Обчислення ймовірностей асоціацій $p(z_i|x_i)$ для кожного вимірювання z_i і кожного об’єкту x_i;
4) Визначення всіх можливих асоціацій між вимірюваннями і цілями. Кожна гіпотеза H_k включає набір відповідностей між вимірюваннями та цілями.
5) Обчислення ймовірності кожної гіпотези $P(H_k)$ на основі ймовірностей асоціацій.
6) Оновлення станів цілей. Використовуючи ймовірності гіпотез, оновлюються оцінки положень цілей. Для кожної цілі x_i

обчислюється зважене середнє всіх вимірювань z_i, враховуючи ймовірності асоціацій [8].

1.5. Алгоритми на основі гібридних методів

1.5.1 Алгоритм Tracklet-Based Tracking

Алгоритм полягає в розбитті довготривалих траєкторій об'єктів на менші відрізки(треклети). Потім треклети з'єднуються до повних траєкторій об'єктів. Добре працює зі складними сценами з великою кількістю об'єктів.

Принцип роботи:

1) На першому етапі виконується детекція об'єктів на кожному кадрі відео. Може бути реалізовано за допомогою різних методів, наприклад, нейронні мережі YOLO, Faster R-CNN або інші алгоритми детекції.
2) Після детекції об'єктів, об'єкти відстежуються протягом декількох послідовних кадрів, утворюючи треклети. Треклети - це короткі послідовності кадрів, де об'єкт відстежується безперервно.
3) Для кожного нового кадру визначається, як з'єднати нові детекції з існуючими треклетами. Це може бути зроблено за допомогою різних метрик, таких як Евклідова відстань, косинусна подібність.
4) Треклети з'єднують між собою, щоб створити довші траєкторії. Це може бути зроблено за допомогою алгоритмів асоціації, таких як Hungarian algorithm або методів оптимізації.
5) Прогноз траєкторій об'єктів і заповнення пропусків детекції [9].

1.5.2 Алгоритм Multi-Hypothesis Tracking (MHT)

Створює і підтримує кілька гіпотез щодо траєкторій об'єктів, що дозволяє краще відстежувати об'єкти у складних умовах. Основна ідея MHT полягає в тому, що для кожної нової серії даних створюється декілька можливих гіпотез

щодо того, які цілі вони представляють. Ці гіпотези оновлюються та перевіряються в наступних кроках, щоб визначити найбільш ймовірні траєкторії [10].

1.6. Алгоритми на основі точок інтересу

1.6.1 Алгоритм Kanade-Lucas-Tomasi (KLT) Tracker

Використовує особливі точки (feature points) для відстеження об'єктів. Алгоритм базується на методі оптичного потоку Лукас-Канаде (Lucas-Kanade) та на критеріях відмінності Томасі (Tomasi) [11].

Основи алгоритму:

- Оптичний потік - відстеження засноване на припущенні, що яскравість пікселя залишається постійною між двома послідовними кадрами. Метод Лукас-Канаде використовує це припущення для оцінки переміщення пікселів у невеликому околі.
- Критерій Томасі: Для вибору точок, які будуть відстежуватись, метод Томасі обирає такі, що легко відслідковуються, тобто "гарні" функції (good features). Це робиться за допомогою аналізу градієнтів зображення. Обираються ті точки, де мінімальні власні значення матриці градієнтів (обчисленої для кожної точки) є достатньо великими.

Принцип роботи:

1) Обираються точки, які добре відслідковуються за допомогою критерію Томасі, зазвичай кути або текстуровані області.
2) Після вибору точок метод Лукас-Канаде використовується для оцінки їхнього переміщення між послідовними кадрами. Це робиться за допомогою розв'язання системи лінійних рівнянь для кожного вікна зображення навколо вибраної точки.

3) Застосовуються ітеративні методи, такі як підхід з найменшими квадратами, для уточнення переміщення,бо оцінка переміщення може бути неточною через шум або великі зміщення.

1.7. Алгоритми на основі баєсових фільтрів

1.7.1 Алгоритм Kalman Filter

Це рекурсивний алгоритм оцінки стану динамічної системи, який мінімізує середньоквадратичну помилку. Він широко використовується для трекінгу об'єктів у різних областях, таких як робототехніка, навігація Використовується для лінійних і гаусових процесів, що забезпечує ефективний трекінг об'єктів з передбаченням їх майбутнього стану.

Принцип роботи:

1) Ініціалізація початкового стану системи x_0 та коваріаційної матриці P_0
2) Прогнозування:
   - Оцінка наступного стану на основі моделі процесу

$$x_{k|k-1} = F_k x_{k-1|k-1} + B_k u_k,$$

де F_k – матриця переходу стану, B_k – матриця управління, u_k – вектор управління

   - Оцінка коваріації похибки прогнозу

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k,$$

де Q_k – коваріаційна матриця шуму процесу.

3) Оновлення:

- Оновлення стану на основі нових вимірювань

$$y_k = z_k - H_k x_{k|k-1},$$

де y_k – вектор різниці(інновації), z_k – вектор вимірювань, H_k – матриця спостереження

- Обчислення матриці інноваційної коваріації

$$S_k = H_k P_{k|k-1} H_k^T + R_k$$

, де R_k – коваріаційна матриця шуму вимірювань

- Обчислення коефіцієнта Калмана

$$K_k = P_{k|k-1} H_k^T S_k^{-1}$$

- Оновлення оцінки стану

$$x_{k|k} = x_{k|k-1} + K_k y_k$$

- Оновлення коваріаційної матриці похибки [12]

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

, де I – одинична матриця

1.7.2 Extended Kalman Filter (EKF)

Алгоритм розширює класичний Калманів фільтр для нелінійних систем. EKF лінеаризує нелінійні функції навколо поточної оцінки стану з використанням розкладу Тейлора.

1) Лінеаризація моделі процесу заключається в обчисленні якобіана матриці переходу стану F_k;
2) Лінеаризація моделі вимірювання заключається в обчисленні якобіана матриці спостереження H_k [13].

Unscented Kalman Filter (UKF): адаптація класичного фільтра Калмана, яка дозволяє ефективно працювати з нелінійними динамічними системами. Основна ідея полягає в тому, щоб замість лінійної апроксимації використовувати апроксимацію за допомогою «Unscented» точок, що краще відображають реальну структуру системи.

Принцип роботи:

1) Створення «Unscented» точкок відповідно до поточного стану системи та її коваріаційної матриці.

2) Передбачення кожної «Unscented» точки через функцію динаміки системи для отримання прогнозувань стану наступного часового кроку.
3) Оцінка прогнозованого стану системи та коваріаційної матриці передбачення на основі прогнозів «Unscented» точок
4) Прогнозування вимірювання для кожної «Unscented» точки
5) Обчислення очікуваного вимірювання та його коваріаційної матриці.
6) Обчислення коефіцієнта Калмана
7) Оновлення стану та коваріаційної матриці на основі фактичних вимірювань [14].

1.8. Інші алгоритми

1.8.1 Mean-Shift Tracking

Ітеративний алгоритм, що використовує статистичні характеристики об'єкта для відстеження його в наступних кадрах. Основна ідея полягає в тому, щоб знаходити області, де густота ймовірності зміни пікселів є найбільшою, що вказує на наявність об'єкта, який слід відстежувати. Чутливий до змін освітлення, шуму.

Принцип роботи:

1) Обчислення гістограми кольорів або функцію щільності для пікселів всередині вікна (початкового вікна для відстеження об'єкта). Це визначить розподіл кольорів або інших ознак в межах вікна.
2) Обчислення вагового центру (середнє) гістограми або функції щільності. Це може бути зважене середнє значення кожної особливості, де вага відповідає значенню гістограми або щільності в цій точці.
3) Переміщення вікна в напрямку центру мас, обчисленого на попередньому кроці, зазвичай робиться шляхом обчислення

вектора зсуву від поточного положення вікна до вагового центру та застосування цього зсуву до вікна.

4) Повторення кроків 2-4 до тих пір, поки вікно не збіжиться до точки максимуму гістограми або досягне певного критерію зупинки (наприклад, зміна відстані між вікнами менше певного порогу) [15].

Коли вікно збігається, воно вказує на нове положення об'єкта на поточному кадрі. Нове положення буде використано для відстеження об'єкта на наступних кадрах відео.

1.8.2 CamShift (Continuously Adaptive Mean Shift)

Розширення Mean-Shift, яке адаптує розмір вікна в залежності від зміни розміру та орієнтації об'єкта в кадрі. Цим забезпечує більш стабільне відстеження об'єктів.

Принцип роботи:

1) Обчислення гістограми кольорів або функції щільності для пікселів всередині вибраного вікна.
2) Виконання ітеративного процесу Mean Shift, який включає обчислення вагового центру гістограми і переміщення вікна в напрямку цього центру, як описано в алгоритмі Mean Shift.
3) Адаптивне масштабування для кращого відстеження об'єкта, зазвичай робиться за допомогою прямокутної апертури, яка охоплює область, що більш відповідає формі та орієнтації об'єкта [15].

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було розглянуто існуючі системи трекінгу об’єктів, алгоритми, що використовують різні підходи до вирішення проблеми супроводу цілей.

Проаналізувавши приведені вище алгоритми стає зрозуміло, що для систем з обмеженою обчислювальною потужністю слід використати алгоритми з невеликою обчислювальною складністю. Такими алгоритмами є:

- CSRT(Discriminative Correlation Filter with Channel and Spatial Reliability)
- MOSSE(Minimum Output Sum of Squared Error)

РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1. Мова програмування

Для розробки системи було обрано мову програмування Python

2.2. Мова Python

Python - високорівнева мова програмування загального призначення, підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване, імперативне, функціональне і процедурне програмування.

Основні переваги:

- Простота і Читабельність Коду
- Абстрагує багато складних деталей, таких як управління пам'яттю, що дозволяє зосередитися на логіці роботи програми.
- Підтримка різних парадигм програмування дозволяє використовувати Python для вирішення різноманітних задач, від скриптів до складних наукових обчислень і веб-розробки.
- Python має велику стандартну бібліотеку, яка включає модулі для роботи з різними протоколами, форматами файлів, інтерфейсами операційної системи, що значно спрощує розробку програм.
- Python легко інтегрується з іншими мовами та технологіями, такими як C/C++, Java, .NET, що значно розширює межі застосування в різноманітних проектах.
- Python є кросплатформенною мовою, тобто програми, написані на Python, можуть виконуватися на різних операційних системах, таких як Windows, macOS, Linux без змін коду.
- Python має велику спільноту розробників, що сприяє постійній появі нових бібліотек та інструментів. Серед популярних бібліотек:

NumPy, Pandas, Matplotlib, SciPy, TensorFlow, Django та багато інших.

Недоліки:

- Низька Швидкість Виконання:

Python є інтерпретованою мовою, що впливає на швидкість виконання коду порівняно з компільованими мовами, такими як C або C++.

- Високе Споживання Пам'яті:

Динамічна типізація та управління пам'яттю в Python можуть призводити до високого споживання оперативної пам'яті, що може бути проблемою для великих додатків.

- Обмежена Підтримка Мобільної Розробки:

Хоча існують бібліотеки для розробки мобільних додатків на Python, такі як Kivy і BeeWare, вони не є настільки зрілими і популярними, як інструменти для Java або Swift.

- Проблеми з багатопоточністю:

Python має Global Interpreter Lock (GIL), який обмежує одночасне виконання потоків, що може бути перешкодою для багатопотокових додатків, що потребують високої продуктивності.

- Молоді Інструменти для Комплексних Систем:

Хоча Python чудово підходить для прототипування та розробки малих і середніх додатків, він може бути не найкращим вибором для дуже великих і комплексних систем, які потребують високої продуктивності і суворого управління ресурсами.

- Відсутність Статичної Типізації:

Динамічна типізація робить Python гнучким і зручним для швидкого прототипування, але також може призводити до помилок, які важко виявити на етапі компіляції.

Сфери застосування мови Python:

- Веб-розробка
- Завдяки фреймворкам, таким як Django та Flask, Python широко використовується для створення веб-додатків і веб-сервісів.
- Наукові обчислення та обробка даних
- Python є однією з найпопулярніших мов для наукових досліджень та аналізу даних завдяки бібліотекам, таким як NumPy, Pandas, Matplotlib, SciPy.
- Інтернет речей (IoT)
- Python використовується для розробки додатків для мікроконтролерів і IoT-пристроїв завдяки простоті і великій кількості бібліотек для роботи з апаратним забезпеченням.
- Розробка ігор

  Бібліотеки, такі як Pygame, дозволяють створювати 2D ігри та навчальні проекти.
- Штучний інтелект і машинне навчання

  Бібліотеки, такі як TensorFlow, Keras, PyTorch, роблять Python потужним інструментом для розробки моделей машинного навчання та глибокого навчання

Worldwide, May 2024 :

Rank	Change	Language	Share	1-year trend
1		Python	28.98 %	+1.2 %
2		Java	15.97 %	+0.1 %
3		JavaScript	8.79 %	-0.6 %
4		C#	6.78 %	-0.0 %
5		C/C++	6.46 %	-0.0 %
6	↑	R	4.76 %	+0.3 %
7	↓	PHP	4.55 %	-0.6 %
8		TypeScript	3.03 %	-0.0 %
9		Swift	2.76 %	+0.4 %
10		Rust	2.6 %	+0.4 %
11		Objective-C	2.41 %	+0.3 %
12		Go	2.25 %	+0.3 %
13		Kotlin	1.97 %	+0.1 %
14		Matlab	1.52 %	-0.1 %
15	↑↑↑↑	Dart	1.0 %	+0.1 %

Рисунок 2.1 – Популярність мов програмування за даними PYPL

На рис. 2.1 видно, що Python – найпопулярніший інструмент сьогодення.[16]

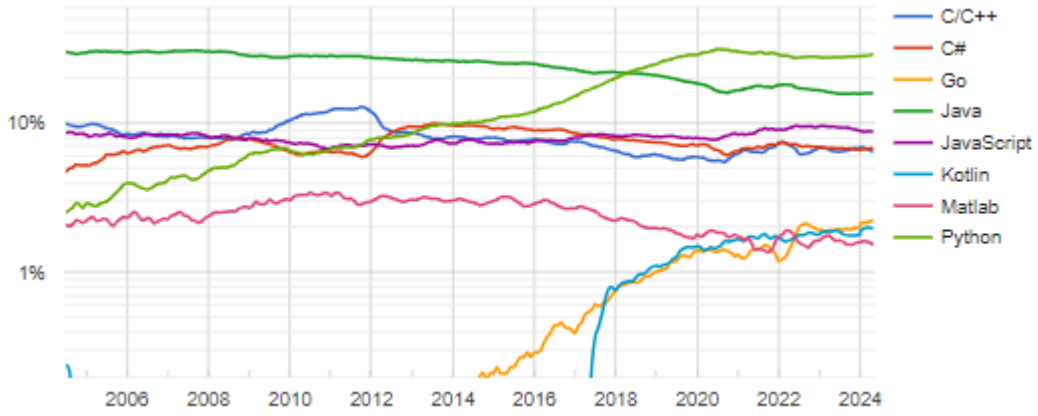


Рисунок 2.2 – Динаміка популярності мов програмування за даними PYPL

На рис.2.2 проілюстровано динаміку популярності мов програмування. Python залишається лідером протягом останніх 5-ти років за даними PYPL[16]

2.3. OpenCV

OpenCV є потужною бібліотекою для обробки зображень та відео, яка пропонує широкий спектр інструментів для виконання різноманітних задач у сфері комп'ютерного зору. Вона інтегрується з іншими науковими бібліотеками Python, що робить її незамінною для дослідників та розробників.

Основні компоненти:

1) Mat: Основний тип даних для зберігання зображень у OpenCV. Це багатовимірний масив, який підтримує різні канали (наприклад, RGB, BGR, Grayscale).
2) CascadeClassifier: Клас для об'єктного детектування на основі каскадів Хаара.
3) KeyPoint: Клас для зберігання ключових точок у алгоритмах детектування особливостей (наприклад, SIFT, SURF).

Переваги:

1) Відкритий код

- Безкоштовна: OpenCV є бібліотекою з відкритим вихідним кодом, доступною для всіх безкоштовно. Це знижує бар'єри для входу в область комп'ютерного зору.
- Спільнота: Велика та активна спільнота користувачів та розробників, яка постійно вдосконалює бібліотеку та надає підтримку новачкам.

2) Кросплатформеність

- Підтримка різних платформ: OpenCV підтримує різні операційні системи, включаючи Windows, Linux, macOS та мобільні платформи (Android, iOS).
- Мультиплатформені API: Однаковий код можна використовувати на різних платформах, що спрощує розробку та перенесення проектів.

3) Широкий набір функцій.

- Обробка зображень: Функції для перетворення кольорів, фільтрації, сегментації, морфологічних операцій тощо.
- Комп'ютерний зір: Функції для розпізнавання облич, детектування об'єктів, трекінгу руху, побудови глибоких нейронних мереж.
- Машинне навчання: Підтримка попередньо навчених моделей та інтеграція з бібліотеками для машинного навчання, такими як TensorFlow, PyTorch, Caffe.

4) Висока продуктивність

- Оптимізація: OpenCV використовує оптимізовані алгоритми для забезпечення високої продуктивності.
- Підтримка апаратного прискорення: Підтримка використання GPU через CUDA та OpenCL для значного прискорення обробки.

5) Інтеграція з іншими інструментами

- Python, C++, Java та інші мови: Підтримка кількох мов програмування, що дозволяє використовувати OpenCV у різних середовищах розробки.
- Інтеграція з науковими бібліотеками: Легка інтеграція з бібліотеками Python, такими як NumPy, SciPy та Matplotlib.

6) Документація та ресурси

- Документація: Вичерпна документація з прикладами використання.
- Навчальні ресурси: Велика кількість онлайн-курсів, туторіалів, книг та відеоуроків.

7) Модульність та розширюваність

- Модульна структура: Бібліотека складається з окремих модулів (наприклад, core, imgproc, video, ml), що дозволяє використовувати лише необхідні компоненти.

8) Розширюваність: Легко додавати нові функції та алгоритми.

- Реальний досвід та застосування

- Використання у промисловості: OpenCV широко використовується в різних галузях, таких як робототехніка, охорона здоров'я, автомобільна промисловість, розпізнавання облич тощо.
- Проекти з відкритим кодом: Багато проектів з відкритим кодом використовують OpenCV, що дозволяє легко знайти приклади реального використання та адаптувати їх до своїх потреб.

2.4. NumPy

NumPy - Numerical Python, це бібліотека Python з відкритим кодом, яка підтримує великі багатовимірні масиви та матриці. Бібліотека також має набір високорівневих математичних функцій для роботи з масивами. Вона забезпечує високопродуктивний об’єкт багатовимірного масиву та інструменти для роботи з цим масивом.

NumPy має різні функції:

- Потужний об'єкт N-вимірного масиву
- Трансляційні функції
- Інструменти для інтеграції коду C/C++ і Fortran
- Корисна лінійна алгебра, перетворення Фур’є та можливості випадкових чисел

Довільні типи даних можна визначати за допомогою NumPy, що дозволяє легко та швидко інтегруватися з широкою різноманітністю баз даних [17].

Переваги NumPy:

- Підходить для аналізу даних
- Масиви NumPy займають менше пам’яті та забезпечують кращу швидкість виконання порівняно з подібними структурами даних у Python (списки та кортежі).

- Ядро реалізовано на C, що дозволяє виконувати операції, близькі до швидкості апаратного забезпечення, що призводить до значного підвищення продуктивності, особливо для великих наборів даних
- NumPy пропонує повну колекцію математичних функцій, включаючи лінійну алгебру, перетворення Фур'є та генерацію випадкових чисел, які є фундаментальними для наукових розрахунків
- Підтримує векторні операції (поелементне додавання та множення), обчислення добутку Кронекера тощо. Списки Python не підтримують ці функції.
- Інтеграція з іншими бібліотеками: NumPy служить основним пакетом для багатьох інших наукових бібліотек на Python, таких як SciPy, Pandas і Matplotlib.

Недоліки NumPy:

- Ускладнене порівняння значень в інтерпретаторі Python через відсутність міжплатформенної підтримки в Python та одночасним використанням "Nan" у NumPy для позначення відсутніх значень
- Потребує безперервної ділянки пам'яті – як наслідок, «дорогі» операції вставки та видалення даних [18].

Бібліотека NumPy підходить для наукових обчислень, оскільки забезпечує достатню кількість ефективних інструментів для роботи зі складними багатовимірними структурами.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було розглянуто питання вибору мови для реалізації проєкту. Найкращим варіантом за сукупністю переваг та недоліків стала мова Python. У розділі детально описані основні вади та переваги мови.

Також було розглянуто бібліотеку OpenCV, що надає потужний інструментарій для реалізації алгоритмів обробки зображень і не тільки.

Як основне середовище розробки було обрано Pycharm.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Встановлення середовища розробки та необхідних для розробки бібліотек

Першим кроком з офіційного сайту[19] завантажуємо актуальну версію python.

Встановимо середовище Pycharm для розробки програмного продукту . На офіційному сайті[20] завантажуємо IDE. Наступним кроком є створення проєкту та завантаження необхідних бібліотек. Використовую командну строку інсталюємо бібліотеки: pip install opencv-contrib-python, pip install numpy

Проєкт створено, середовище готове до розробки.

3.2 Підготовка структури проєкту

Проєкт складається з файла main.py, який є основним виконуваним файлом і буде зберігати в собі код старту системи. та дерикторії videos, в якій буде збережено відео для обробки системою, також буде можливість оброблювати потокове відео з камери ()наприклад з веб-камери ноутбука)

3.3 Розробка графічного інтерфейсу користувача

Система має виводити на екран відео-ряд з заготовленого відео або з камери, при цьому оброблюючи кожен кадр і додаючи на нього прямокутник, який буде показувати місце знаходження об’єкта слідування.

Для цього будемо по черзі зчитувати кожен кард з потоку та для початку просто виводити його на екран. Для цього створюємо спеціальний клас ObjectTracker який буде відповідати за головну логіку системи. Він приймає в конструктор відео, яке ми відразу перетворюємо на VideoCapture, який нам дозволив в основному циклі програми отримувати кадр 1 за 1.

```python
class ObjectTracker:
    def __init__(self, input_source):
        self.video_capture = cv2.VideoCapture(input_source)
    1 usage
    def run(self):
        cv2.namedWindow("Original")
        fps = self.video_capture.get(cv2.CAP_PROP_FPS)
        while True:
            start_time = cv2.getTickCount()
            ret, frame = self.video_capture.read()
            if not ret:
                break

            cv2.imshow( winname: "Original", frame)
            end_time = cv2.getTickCount()
            time_taken = (end_time - start_time) / cv2.getTickFrequency()
            delay_time_ms = max(1, int((1 / fps - time_taken) * 1000))
            key = cv2.waitKey(delay_time_ms)
            if key != -1:
                if key == ord('q'):
                    break

        self.video_capture.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
    INPUT_CAMERA_0 = 0
    INPUT_CAMERA_1 = 'video\\drone1.mp4'
    tracker = ObjectTracker(INPUT_CAMERA_0)
    tracker.run()
```

Рисунок 3.1 – Головний цикл програми

В головному циклі на даний момент отримуємо кадр, перевіряємо чи успішна операція (якщо ні – виходимо з циклу) і далі відображаємо кард на екран і вираховуємо затримку, щоб відео програвалося без прискорень (потрібно лише для тестування, на реальних системах ми хочемо щоб результат був якнайшвидше). Також в кінці циклу перевіряємо чи не натиснута кнопка q, що означає завершення роботи програми.

Наступним кроком буде додавання можливості натискання на відео на виділення регіону інтересу прямокутником, для цього було написано

допоміжні функції draw_bounding_box та draw_tracking_marker, які використовуються для відмалювання прямокутника та маркеру відповідно.

```
1 usage
def draw_tracking_marker(frame, center, color):
    radius = 3
    # Draw the lines
    cv2.line(frame,  pt1: (center[0] - 25, center[1]),  pt2: (center[0] + 25, center[1]), color,  thickness: 2)
    cv2.line(frame,  pt1: (center[0], center[1] - 25),  pt2: (center[0], center[1] + 25), color,  thickness: 2)
    # Draw the circle in the center
    cv2.circle(frame, center, radius,  color: (0, 255, 0), -1)
    cv2.putText(frame,  text: "Tracking",  org: (100, 50), cv2.FONT_HERSHEY_SIMPLEX,  fontScale: 0.7,  color: (75, 255, 75),  thickness: 1)

1 usage
def draw_bounding_box(frame, bounding_box, color):
    point1 = (int(bounding_box[0]), int(bounding_box[1]))
    point2 = (int(bounding_box[0] + bounding_box[2]), int(bounding_box[1] + bounding_box[3]))
    cv2.rectangle(frame, point1, point2, color, 2, 1)
    cv2.putText(frame,  text: "Tracking",  org: (100, 50), cv2.FONT_HERSHEY_SIMPLEX,  fontScale: 0.7,  color: (255, 255, 255),  thickness: 1)
```

Рисунок 3.2 – Допоміжні функції

Також додаємо обробних події натискання мишки щоб зберегти координати зони інтересу.

```
1 usage
def run(self):
    cv2.namedWindow("Original")
    cv2.setMouseCallback( windowName: "Original", self.select_center_roi)
    fps = self.video_capture.get(cv2.CAP_PROP_FPS)
    while True:
        start_time = cv2.getTickCount()
```

Рисунок 3.3 – Реєстрація обробки натискання мишки

```
1 usage
def select_center_roi(self, event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        self.is_tracking = False
        self.center_point = (x, y)

        frame_height, frame_width = self.frame.shape[:2]
        width = frame_width // 4
        height = frame_height // 3
        x0 = max(0, x - (width // 2))
        y0 = max(0, y - (height // 2))
        x0 = min(x0, frame_width - width)
        y0 = min(y0, frame_height - height)

        self.initial_bounding_box = (x0, y0, width, height)
        self.initialize_tracker()
```

Рисунок 3.4 – Обробник натискання мишки

І тепер останнім кроком залишилося тільки відмалювати прямокутник на кадрі

```
1 usage
def run(self):
    cv2.namedWindow("Original")
    cv2.setMouseCallback( windowName: "Original", self.select_center_roi)
    fps = self.video_capture.get(cv2.CAP_PROP_FPS)
    while True:
        start_time = cv2.getTickCount()
        ret, frame = self.video_capture.read()
        if not ret:
            break

        bounding_box = self.initial_bounding_box
        center_point = (int(bounding_box[0] + bounding_box[2] / 2),
                        int(bounding_box[1] + bounding_box[3] / 2))
        draw_tracking_marker(frame, center_point, color: (55, 255, 255))
        draw_bounding_box(frame, bounding_box, color: (255, 255, 0))

        cv2.imshow( winname: "Original", frame)
        end_time = cv2.getTickCount()
        time_taken = (end_time - start_time) / cv2.getTickFrequency()
        delay_time_ms = max(1, int((1 / fps - time_taken) * 1000))
        key = cv2.waitKey(delay_time_ms)
        if key != -1:
            if key == ord('q'):
                break

    self.video_capture.release()
    cv2.destroyAllWindows()
```

Рисунок 3.5 – Відмалюкання прямокутника

3.4 Використання алгоритмів ШІ для захвату об’єкту

Для захвату об’єкту було прийнято рішення використовувати TrackerMOSSE, спираючись на порівняльний аналіз алгоритмів представлених в розділі 1 та аналіз задачі. Для цього додано метод initialize_tracker в якому створюється об’єкт для взаємодії з трекером цього типу та його початкові

конфігурації. Викликатися цей метод буде всередині select_center_roi після ініціалізації початкової точки.

```
1 usage
def initialize_tracker(self):
    self.is_tracking = True
    self.tracker = cv2.legacy.TrackerMOSSE.create()

    self.tracker.init(self.frame, self.initial_bounding_box)

    self.kalman_filter.statePre = np.array( object: [[self.center_point[0]], [self.center_point[1]],
                                            [0], [0],
                                            [self.initial_bounding_box[2]],
                                            [self.initial_bounding_box[3]]], np.float32)
    self.kalman_filter.statePost = np.array( object: [[self.center_point[0]], [self.center_point[1]],
                                             [0], [0],
                                             [self.initial_bounding_box[2]],
                                             [self.initial_bounding_box[3]]], np.float32)
```

Рисунок 3.6 – Ініціалізація трекеру

Наступним кроком буде додавання в цикл обробки використання цього трекера. Тепер регіон захоплення об’єкту будемо отримувати після обробки трекером поточного кадру використовуючи для цього tracker.update.

```
1 usage
def run(self):
    cv2.namedWindow("Original")
    cv2.setMouseCallback( windowName: "Original", self.select_center_roi)
    fps = self.video_capture.get(cv2.CAP_PROP_FPS)
    while True:
        start_time = cv2.getTickCount()
        ret, frame = self.video_capture.read()
        if not ret:
            break

        self.frame = frame

        if self.is_tracking:
            success, bounding_box = self.tracker.update(self.frame)
            if success:
                center_point = (int(bounding_box[0] + bounding_box[2] / 2),
                                int(bounding_box[1] + bounding_box[3] / 2))
                draw_tracking_marker(frame, center_point,  color: (55, 255, 255))
                draw_bounding_box(frame, bounding_box,  color: (255, 255, 0))

            else:
                self.is_tracking = False

        cv2.imshow( winname: "Original", frame)
        end_time = cv2.getTickCount()
        time_taken = (end_time - start_time) / cv2.getTickFrequency()
        delay_time_ms = max(1, int((1 / fps - time_taken) * 1000))
        key = cv2.waitKey(delay_time_ms)
        if key != -1:
            if key == ord('q'):
                break
```

Рисунок 3.7 – Додавання трекінгу в головний цикл програми

3.5 Поліпшення алгоритму трекінгу

Після проведення попереднього тестування було виявлено дві проблеми, а саме: відсутність плавності трекінгу та погіршення точності відстежування при зміні яскравості(наприклад, об’єкт потрапляє в тінь)

Для вирішення першої проблеми було використано Kalman filter, що був розглянутий у розділі 1. Для цього було додано метод update_kalman, який всередині виконує передбачення наступного місцезнаходження об’єкту, що дозволяє зробити рух трекера більш плавним.

```
def update_kalman(kalman_filter, bounding_box):
    center_x = bounding_box[0] + bounding_box[2] / 2
    center_y = bounding_box[1] + bounding_box[3] / 2
    measurement = np.array([[np.float32(center_x)],
                            [np.float32(center_y)],
                            [np.float32(bounding_box[2])],
                            [np.float32(bounding_box[3])]])
    kalman_filter.correct(measurement)
    prediction = kalman_filter.predict()
    predicted_bounding_box = (int(prediction[0] - prediction[4] / 2),
                              int(prediction[1] - prediction[5] / 2),
                              int(prediction[4]),
                              int(prediction[5]))
    return predicted_bounding_box
```

Рисунок 3.8 – Фільтр Калмана

Також для більш наочного результату додано можливість динамічно вмикати та вимикати роботу фільтра під час виконання програми.

```
if self.is_tracking:
    success, bounding_box = self.tracker.update(self.frame)
    if success:
        if self.use_kalman:
            predicted_bounding_box = update_kalman(self.kalman_filter, bounding_box)
        else:
            predicted_bounding_box = bounding_box
        center_point = (int(predicted_bounding_box[0] + predicted_bounding_box[2] / 2),
                        int(predicted_bounding_box[1] + predicted_bounding_box[3] / 2))
        draw_tracking_marker(frame, center_point, color: (55, 255, 255))
        draw_bounding_box(frame, predicted_bounding_box, color: (255, 255, 0))

    else:
        self.is_tracking = False
```

Рисунок 3.9 – Використання фільтру Калмана в головному циклі програми

Для вирішення другої проблеми було застововано adaptive threshold в чорно-білих кольорах. Це дозволяє привести зображення в стан, коли на ньому нівелюються перепади яскравості та найбільш виразними стають межі об’єктів. Для наочності також було додано можливість динамічно вмикати та вимикати його.

```
while True:
    start_time = cv2.getTickCount()
    ret, frame = self.video_capture.read()
    if not ret:
        break

    if self.use_adaptive_threshold:
        self.frame = cv2.adaptiveThreshold(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), maxValue: 255,
                                           cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, self.threshold, C: 3)
        if self.show_threshold_frame:
            frame = self.frame
    else:
        self.frame = frame

    cv2.putText(frame, text: "Status: ", org: (15, 50), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.7, color: (255, 255, 255), thickness: 1)

    cv2.putText(frame, text: f"kalman={self.use_kalman}, adaptive_threshold={self.use_adaptive_threshold} ", org: (15, 25),
                cv2.FONT_HERSHEY_SIMPLEX, fontScale: 0.4, color: (255, 255, 255), thickness: 1)
```

Рисунок 3.10 – Використання адаптивного трешхолду в головному циклі програми

Для динамічного перемикання параметрів використано обробники натискання відповідних клавіш.

```
key = cv2.waitKey(delay_time_ms)
if key != -1:
    if key == ord('q'):
        break
    elif key == ord('k'):
        self.use_kalman = not self.use_kalman
    elif key == ord('t'):
        self.use_adaptive_threshold = not self.use_adaptive_threshold
    elif key == ord('s'):
        self.show_threshold_frame = not self.show_threshold_frame
    elif key == ord('+'):
        self.threshold += 2
    elif key == ord('-'):
        self.threshold = self.threshold - 2 if self.threshold > 3 else 3
```

Рисунок 3.11 – Обробник натискання клавіш клавіатури

Також було додано виведення параметрів на зображення

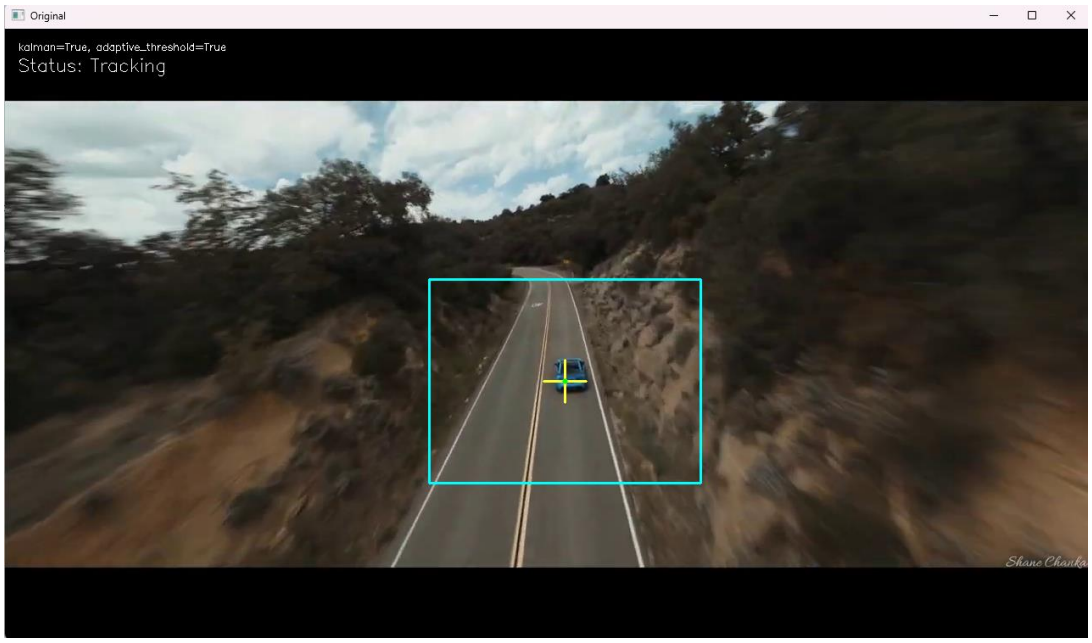


Рисунок 3.12 – Приклад роботи системи. Звичайний режим

Рисунок 3.13 – Приклад роботи системи. Режим відображення адаптивного трешхолду

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було проведено налаштування робочого середовища, а саме Pycharm, було спроєктовано та імплементовано систему захоплення та супроводження об’єкта для автопілота.

Для цього першим кроком стала розробка графічного інтерфейсу користувача за допомого бібліотеки opencv, далі за допомогою цієї бібліотеки було реалізовано покадрову обробку відео для захоплення об’єкту.

Проведено попереднє тестування та виявлено деякі недоліки, які були усунуті за допомого фільтра Калмана та адаптивного трешхолду.

Останнім кроком було додавання можливості динамічної зміни параметрів для наочної демонстрації роботи системи та впливу параметрів на її точність.

РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

Для тестування та аналізу розробленої системи складемо список тестових випадків. Для кожного тестового випадку проведемо ряд тестів щоб визначити коректність робити системи захоплення та супроводження об’єкту для автопілота.

4.1. Формування тест-кейсів

Таблиця 4.1 – Тест-кейси

ID	Тестові дані	Характер руху
1	Відео –people.mp4 Точка захвату – пара, що йде пляжем Довжина відео – 00:26 Роздільна здатність - 1366 на 720 Частота кадрів - 24	Повільних рух, без різкої зміни відносно меж кадру
2	Відео – car.mp4 Точка захвату – машина, що їде гірською дорогою Довжина відео – 00:47 Роздільна здатність - 1280 на 720 Частота кадрів - 24	Швидкий рух по дорозі, автомобіль час від часу покидає кадр
3	Відео – moto.mp4 Точка захвату – мотоцикл, що їде по дорозі Довжина відео – 00:10 Роздільна здатність - 1280 на 720 Частота кадрів - 24	Статичний кадр, де мотоцикл по центру, в кінці кард піднімається вгору
4	Відео – truck.mp4 Точка захвату – фура що їде магістраллю Довжина відео – 00:12 Роздільна здатність - 1280 на 720 Частота кадрів - 24	Фура досить далеко від точки зйомки, повільно їде по дорозі
5	Відео – car1080.mp4 Точка захвату – фура що їде магістраллю Довжина відео – 03:29 Роздільна здатність - 1920 на 1080 Частота кадрів - 30	Машина досить далеко від точки зйомки, повільно їде по дорозі
6	Відео – drone.mp4 Точка захвату – танк, що стоїть Довжина відео – 00:15 Роздільна здатність - 360 на 640	Відео дуже поганої якость, танк стоїть, на нього летить дрон

	Частота кадрів - 30	
7	Відео – parus.mp4 Точка захвату – парсник Довжина відео – 00:17 Роздільна здатність - 640 на 360 Частота кадрів - 30	Парусник пливе можем від одного краю екрану до іншого

Кінець таблиці 4.1

4.2. Перевірка тест-кейсів

Для перевірки кожного тест-кейса було проведено по 10 тестів. Для кожного запуска завват цілі за допомогою нажаття клавішею мишки по об’єкту відбувався впродовж перших 2 секунд відео. Далі представлені результати по кожному тестовому випадку.

1. Тестовий випадок 1

Таблиця 4.2 – Результати тестів для тест-кейса 1

№	Позитивний/негативний	Коментар
1	Позитивний	
2	Позитивний	
3	Позитивний	
4	Середній	Точка була поставлено занадно низько і під кінець відео захват переїхав на тінь
5	Позитивний	
6	Позитивний	
7	Позитивний	
8	Позитивний	
9	Позитивний	
10	Позитивний	

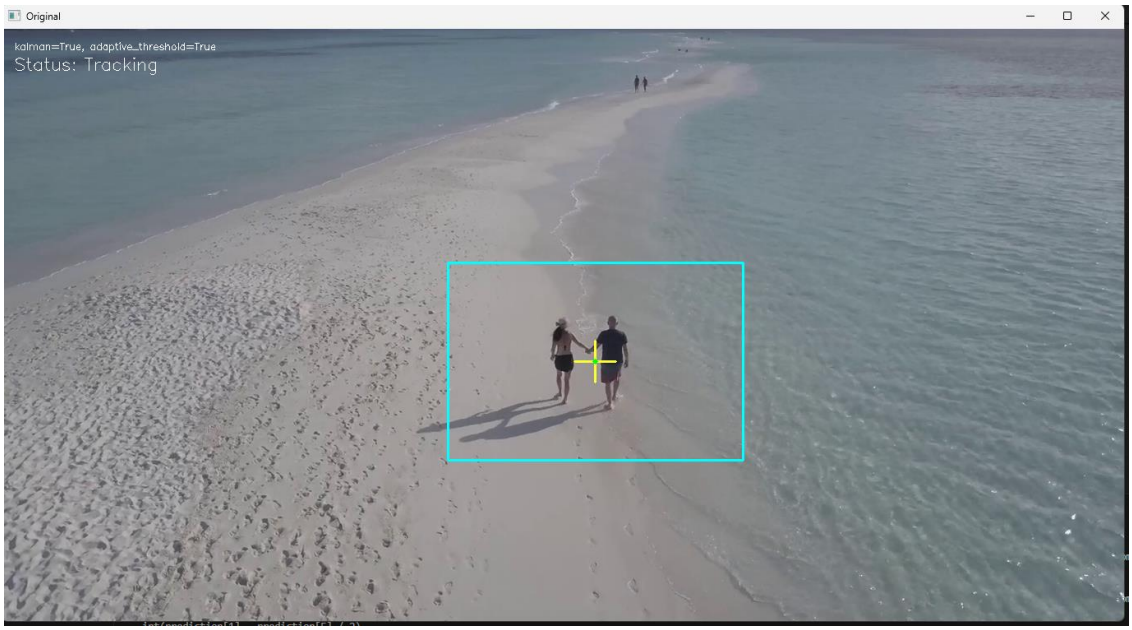


Рисунок 4.1 – Виконання системи на тест-кейсі 1. Приклад 1

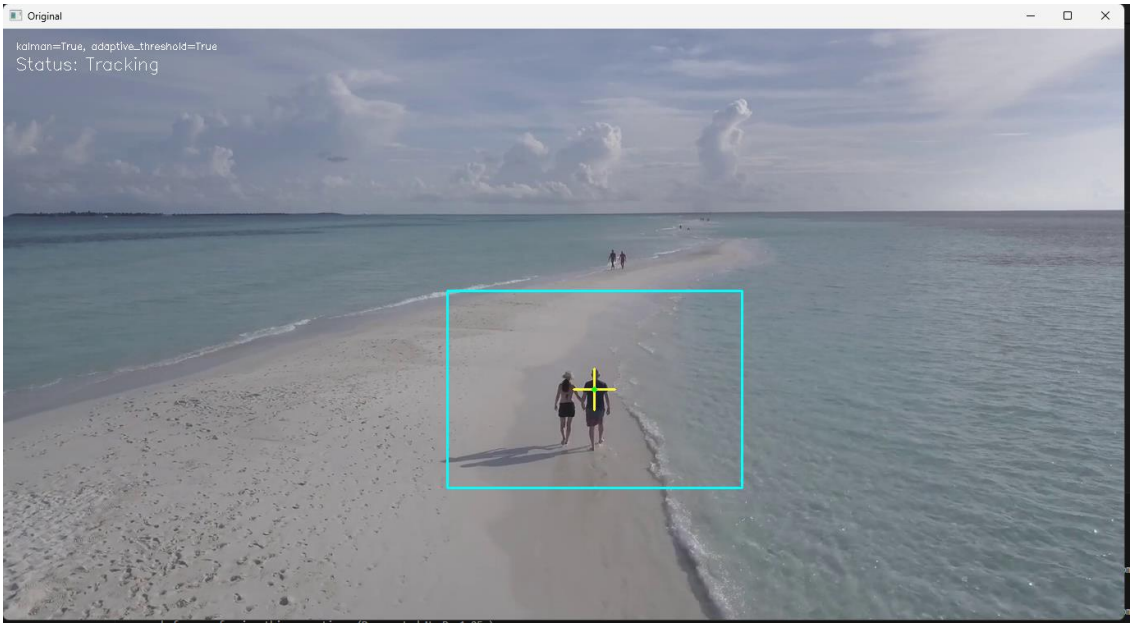


Рисунок 4.2 – Виконання системи на тест-кейсі 1. Приклад 2

2. Тестовий випадок 2

Таблиця 4.3 – Результати тестів для тест-кейса 2

№	Позитивний/негативний	Коментар
1	Позитивний	
2	Негативний	Захват пропав після 9 секунд
3	Позитивний	

4	Середній	Фокус з’їхав вліворуч
5	Позитивний	
6	Негативний	Після пропадання з кадру захват пропав
7	Негативний	Після пропадання з кадру захват залишився на дорозі
8	Позитивний	
9	Середній	Захват трохи з’їхав після різкого переміщення камери
10	Позитивний	

Кінець таблиці 4.3

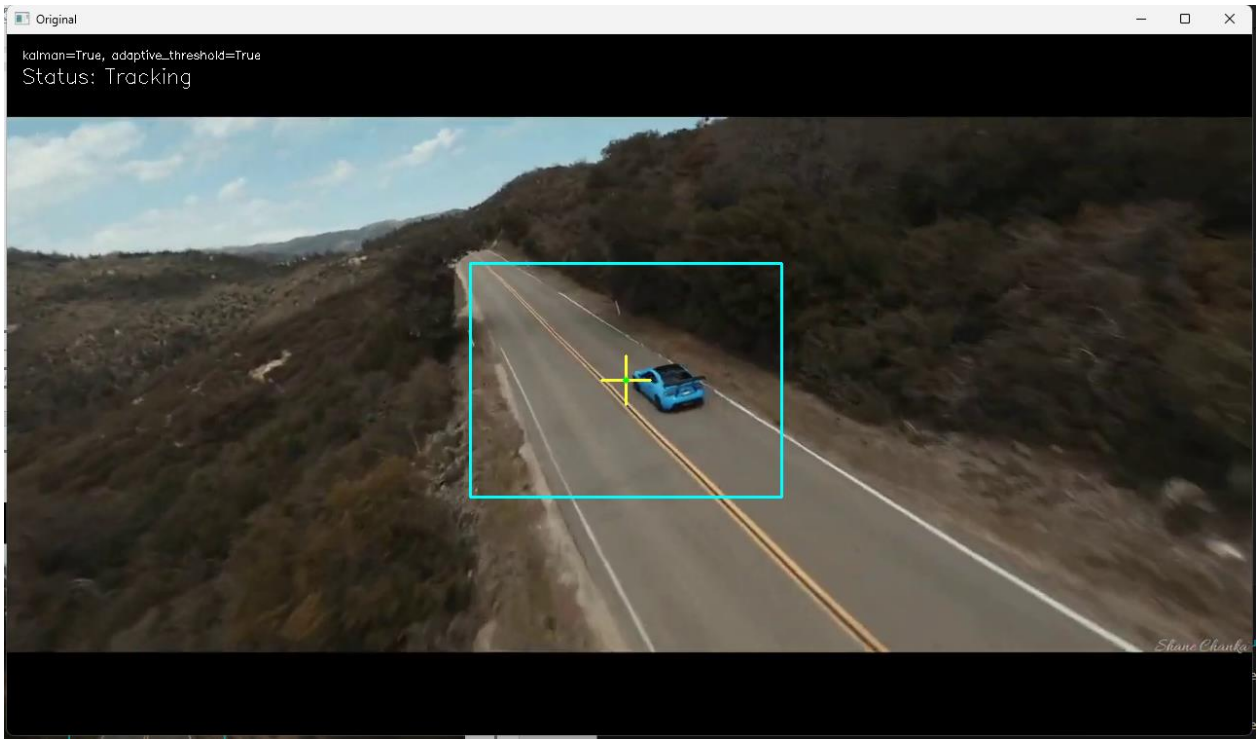


Рисунок 4.3 – Виконання системи на тест-кейсі 2. Приклад 1

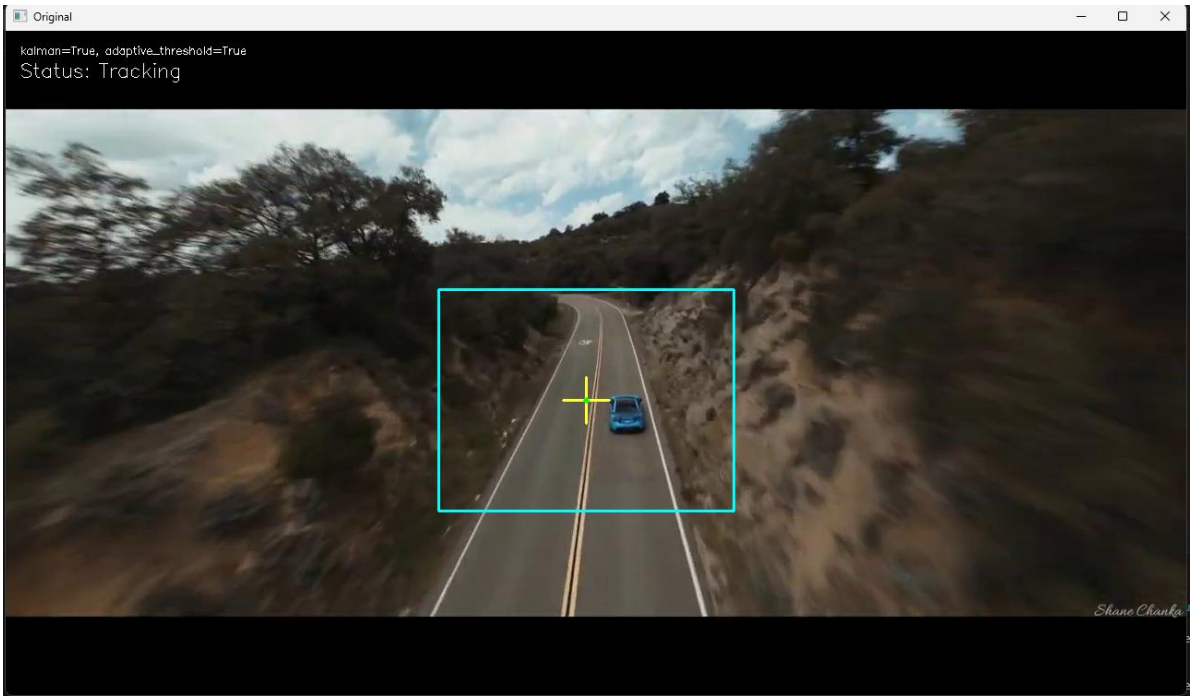


Рисунок 4.4 – Виконання системи на тест-кейсі 2. Приклад 2

3. Тестовий випадок 3

Таблиця 4.4 – Результати тестів для тест-кейса 3

№	Позитивний/негативний	Коментар
1	Позитивний	
2	Позитивний	
3	Позитивний	
4	Позитивний	
5	Позитивний	
6	Позитивний	
7	Позитивний	
8	Позитивний	
9	Позитивний	
10	Позитивний	

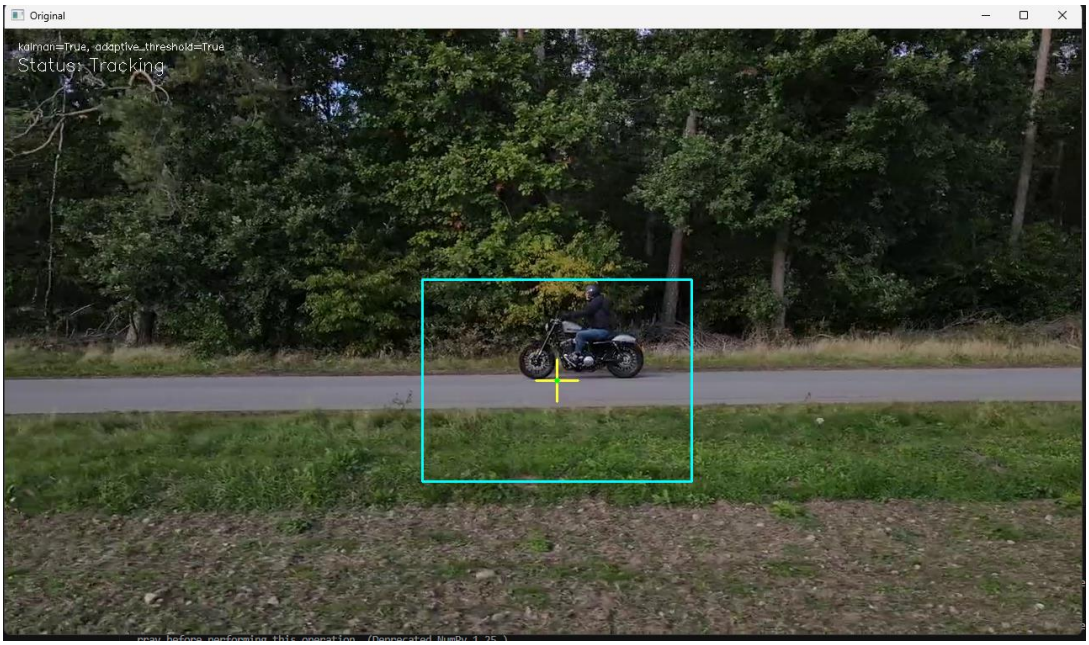


Рисунок 4.5 – Виконання системи на тест-кейсі 3. Приклад 1

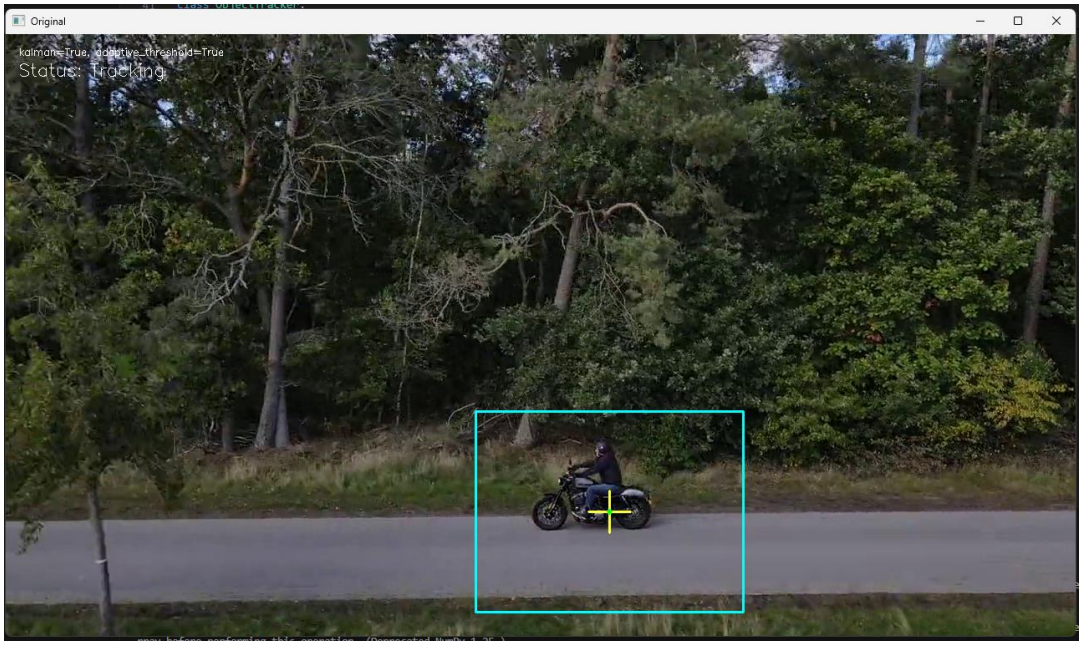


Рисунок 4.6 – Виконання системи на тест-кейсі 3. Приклад 2

4. Тестовий випадок 4

Таблиця 4.5 – Результати тестів для тест-кейса 4

№	Позитивний/негативний	Коментар
1	Середній	Завхат залишається на дорозі через приблизно 6 секунд руху
2	Середній	
3	Середній	
4	Середній	

5	Позитивний	
6	Середній	Завхат залишається на дорозі через приблизно 6 секунд руху
7	Середній	
8	Середній	
9	Середній	
10	Середній	

Кінець таблиці 4.5

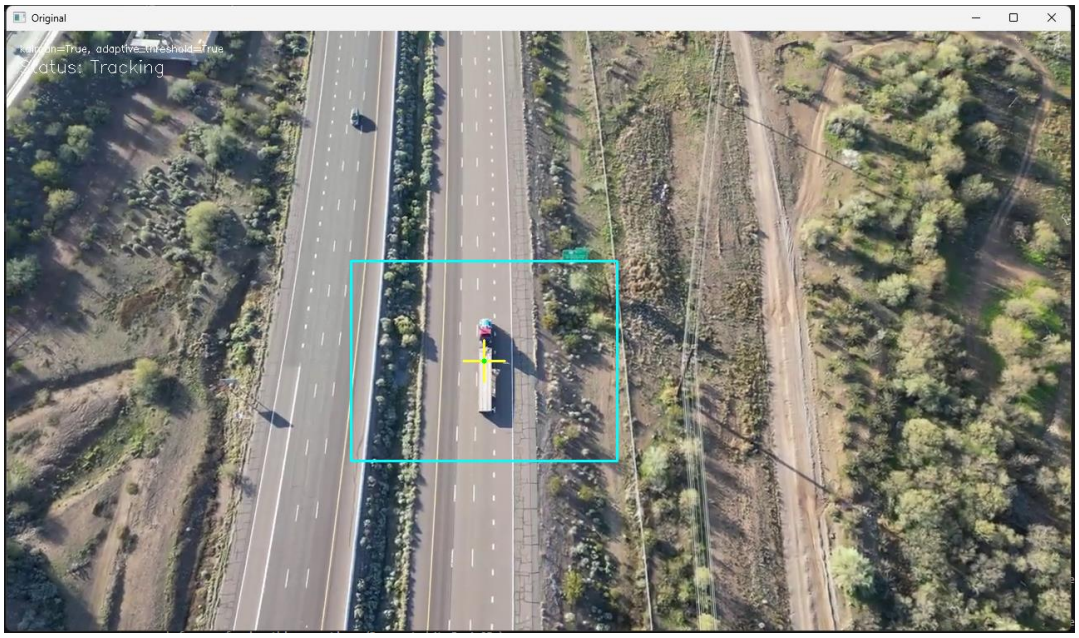


Рисунок 4.7 – Виконання системи на тест-кейсі 4. Приклад 1

Рисунок 4.8 – Виконання системи на тест-кейсі 4. Приклад 2

5. Тестовий випадок 5

Таблиця 4.6 – Результати тестів для тест-кейса 5

№	Позитивний/негативний	Коментар
1	Негативний	Захват залишається на дорозі, тримається на машині не більше 3-4 секунд, після чого просто переходить на захват деякої місцевості
2	Негативний	
3	Негативний	
4	Негативний	
5	Негативний	
6	Негативний	
7	Негативний	
8	Негативний	
9	Негативний	
10	Негативний	

Рисунок 4.9 – Виконання системи на тест-кейсі 5. Приклад 1

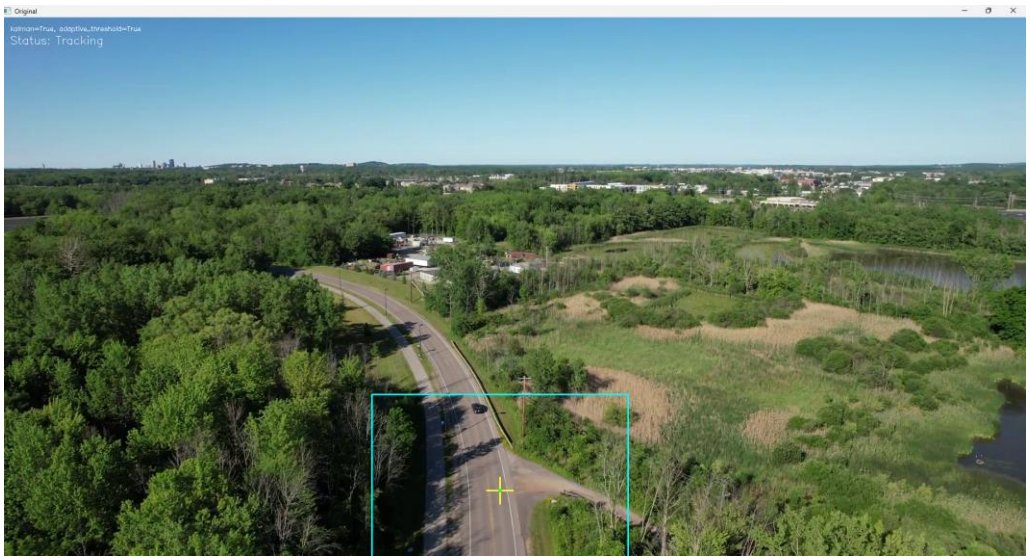


Рисунок 4.10 – Виконання системи на тест-кейсі 5. Приклад 2

6. Тестовий випадок 6

Таблиця 4.7 – Результати тестів для тест-кейса 6

№	Позитивний/негативний	Коментар
1	Середній	Захват в якийсь момент залишився на дорозі
2	Позитивний	
3	Середній	Захват в якийсь момент залишився на дорозі
4	Середній	На початку була неточність
5	Негативний	Захват спав
6	Негативний	Відбувся захват не правильного об’єкту
7	Позитивний	
8	Середній	Захват в якийсь момент залишився на дорозі
9	Позитивний	
10	Негативний	Відбувся захват не правильного об’єкту

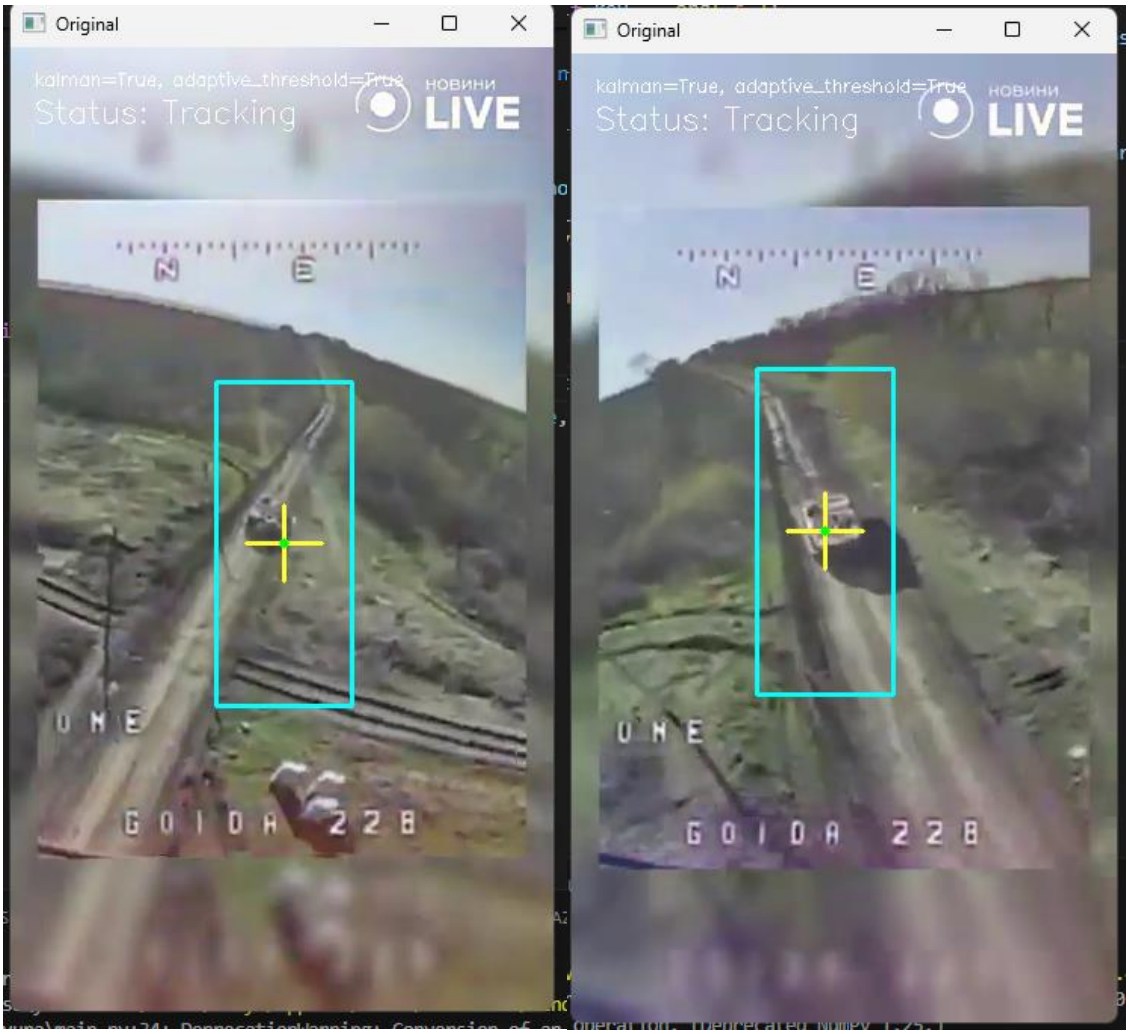


Рисунок 4.11 – Виконання системи на тест-кейсі 6. Приклади 1 та 2

7. Тестовий випадок 7

Таблиця 4.8 – Результати тестів для тест-кейса 7

№	Позитивний/негативний	Коментар
1	Позитивний	
2	Позитивний	
3	Позитивний	
4	Позитивний	
5	Позитивний	
6	Позитивний	
7	Позитивний	
8	Позитивний	

9	Позитивний	
10	Позитивний	

Кінець таблиці 4.8

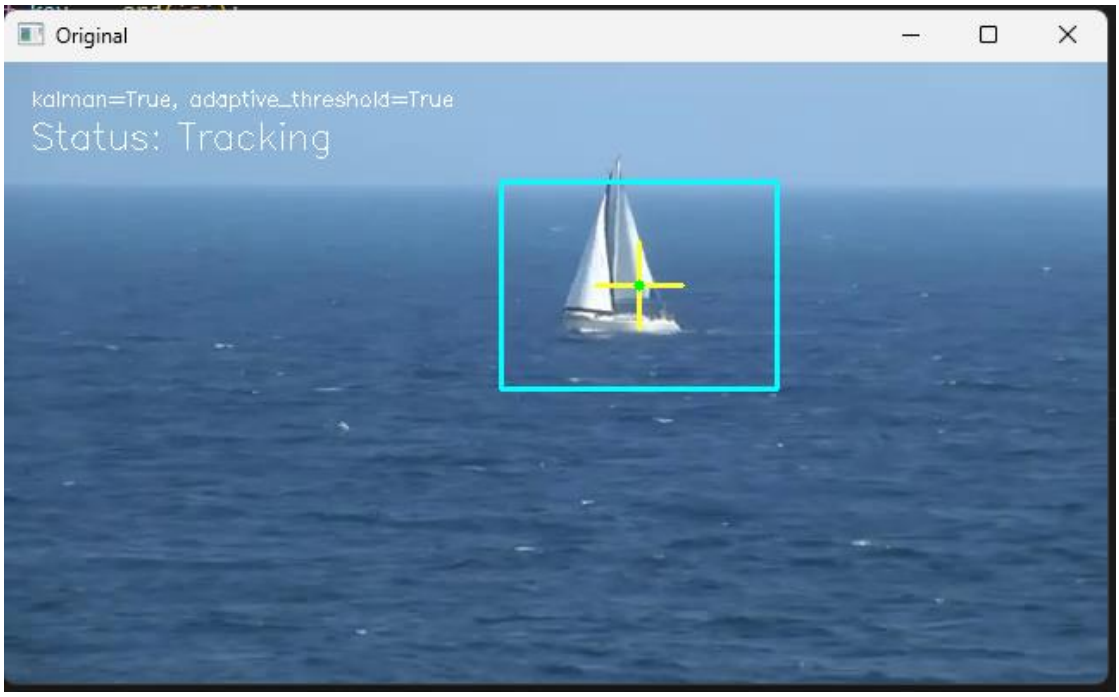


Рисунок 4.12 – Виконання системи на тест-кейсі 7. Приклад 1

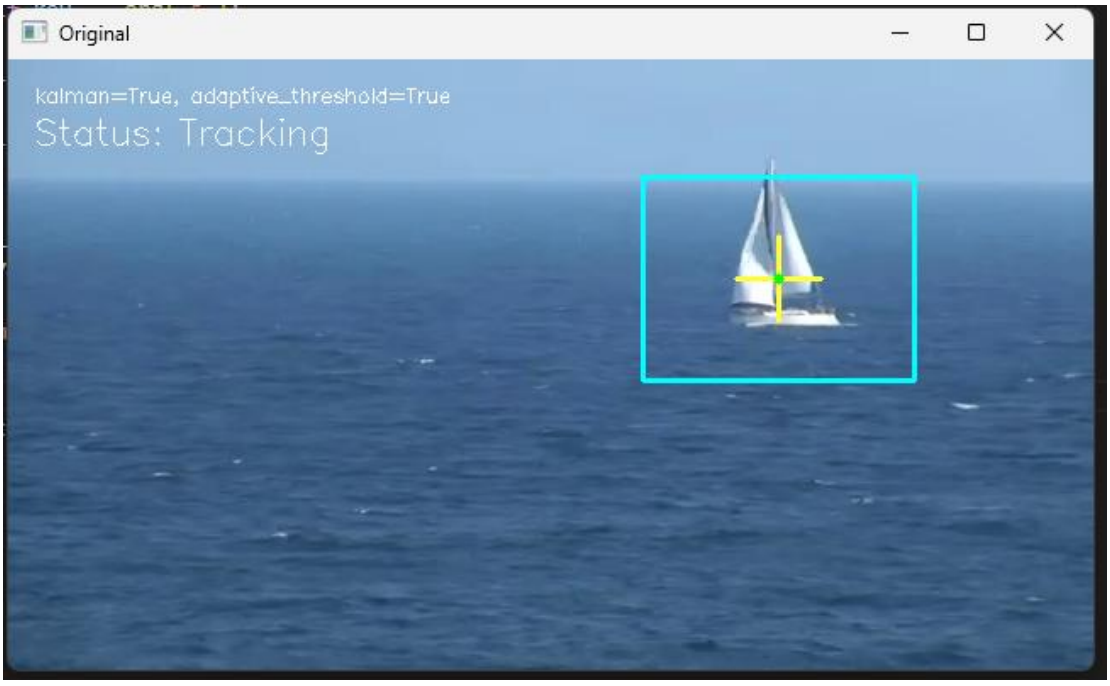


Рисунок 4.13 – Виконання системи на тест-кейсі 7. Приклад 2

4.3. Аналіз результатів тестування

Аналізуючи результати тестування треба враховувати проблеми які можуть бути пов’язані з візуальних відстеженням.

Візуальне відстеження є складним через варіації зовнішнього вигляду об’єкта та його оточення на відео. Зовнішній вигляд об'єкта змінюється через пози, освітлення, природні варіації або нежорстку трансформацію. Об’єкти можуть бути закриті або виходити з кадру відео. Ці варіації ускладнюють відстеження. Поза об'єкта може змінюватися, наприклад, положення живої істоти, яке змінюється під час діяльності або під час різних видів діяльності. Освітлення також змінюється, наприклад, коли тінь падає на предмет, змінюється джерело світла або кут, або відбувається зміна інтенсивності чи яскравості світла. Природні варіації включають зміни зовнішнього вигляду об’єктів у групі, наприклад м’ячі різного розміру або люди з різними фізичними характеристиками. У нежорстких трансформаціях форма та/або розмір об’єкта змінюються, наприклад, коли м’яч віддаляється від камери, м’яч виглядає меншим на відео. Розтягнення або звуження є нежорсткими перетвореннями. Ці зміни можуть спричинити втрату трекером оригінального об’єкта. Коли об’єкт, який відстежується, закритий або виходить за межі кадру, трекер може втратити об’єкт і припинити відстеження, або трекер може почати стежити за об’єктом, який нагадує оригінальний об’єкт. Трекер може припинити оновлення фільтра, коли об’єкт більше не знаходиться в кадрі або його закрито, але він може відновити відстеження та оновлення фільтра, коли об’єкт знову потрапить у поле зору.

Роблячи висновок з тестування можна побачити що система працює досить по різному для різних вхідних даних. Для одних відео (1, 3, 7) система захоплює об’єкт з високою точністю, а для інших (5) по суті не працює і є безкорисним. Також є ряд відео (2, 4, 6) для яких результат є середнім, тобто в залежності від початкових умов (області трекінгу, старту трекінгу і т. д.) результат може вийти досить точним, так і може перестати відстежувати об’єкт чи почати відстежувати інший.

Порівнюючи різні групи відео з різними результатами можна побачити наступне: система дуже добре працює для об’єктів, що займають приблизно ¼ розміру кадру та дуже погано для маленьких об’єктів, бо в систему потрапляє занадто великий обсяг зовнішнього середовища і система починає відстежувати його, також система є стійкою до об’єктів що покидають кадр і повертаються з тої ж сторони (80% успіху). Також очевидно на точність відстежування впливає роздільна здатність камери та частота кадру, але найбільший вплив дає розмір об’єкту і наскільки від вирізняється на фоні (кольором, межами і т. д.)

Для поліпшення результату систему можна поєднувати з іншими системами ШІ і комп’ютерного зору зокрема, наприклад проблему розміру можна було б вирішити якщо після захвату спочатку працювала інша система, яка визначила б межі об’єкту після чого можна було уточнити (збільшити або зменшити) область трекінгу.

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі було складено список тест-кейсів, які дають змогу побачити як працює система на різних тестових даних. Тестування продемоструватo сильні і слабкі сторони розробленої системи. На основі тестування було проведено аналіз який визначив найкращі умови використання системи і визначив напрямок подальших можливих поліпшень системи.

Загалом було проведено 70 тестів для 7 різних відео. Розподіл по результатам:

- Позитивний – 38 тестів
- Середній – 16 тестів
- Негативний – 16 тестів

ВИСНОВКИ

У дипломній роботі було вирішено проблему захоплення і супроводження об’єкту для автопілота у системах з обмеженими обчислювальними ресурсами. На сьогодення ця проблема є актуальною. Оскільки така система може бути автономною і дешевою, також вона може бути частиною автономної системи керування безпілотним літальним апаратом в умовах радіо-електронної боротьби.

Першим етапом вирішення цієї проблеми був аналіз існуючих і загальнодоступних методів трекінгу об’єктів. Було досліджено та порівняно такі методи та підходи: Kernelized Correlation Filters, Discriminative Correlation Filter with Channel and Spatial Reliability, Minimum Output Sum of Squared Error, Deep SORT з використанням Hungarian Algorithm, Fair Multi-Object Tracking, Particle Filter, Joint Probabilistic Data Association, Tracklet-Based Tracking, Multi-Hypothesis Tracking, Kanade-Lucas-Tomasi Tracker, Kalman Filter, Extended Kalman Filter, Continuously Adaptive Mean Shift, Mean-Shift Tracking.

Після проведення порівняльного аналізу було обрано трекер MOSSE та додаткові обробники: фільтр Калмана та адаптивний трешхолд. Такий варіант обробки був реалізований за допомогою бібліотеки opencv та мови програмування python. Для наочної демонстрації роботи програми було реалізовано графічний інтерфейс користувача, та також додано можливість динамічно змінювати параметри системи та включати/виключати додаткові обробки для порівняння їх вкладу у кінцевий результат.

Після розробки були підготовлені тестові відео матеріали та складено список тест-кейсів на основі яких було проведено тестування системи. Для кожного тест-кейса було проведено 10 тестів, та проаналізовано кожний середній або негативний результат. Тестування показало недоліки та переваги алгоритму та виявлено найкращі умови застосування. Після чого було зроблено висновки щодо того як можна покращити систему в наступних дослідженнях, а саме можливість поєднати її ще з іншими алгоритмами машинного навчання та

штучного інтелекту, що б дозволило системі працювати точно при гірших умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Henriques, J. F., Caseiro, R., Martins, P., & Batista, J. (2015). "High-Speed Tracking with Kernelized Correlation Filters." [Електронний ресурс] – https://ieeexplore.ieee.org/document/7001050 (дата звернення 10.05.2024).
2. Bhat, G., Danelljan, M., Gool, L. V., & Timofte, R. (2019). "Learning Discriminative Model Prediction for Tracking." [Електронний ресурс] – https://link.springer.com/chapter/10.1007/978-3-030-01258-8_7 (дата звернення 20.05.2024).
3. Bolme, D. S., Beveridge, J. R., Draper, B. A., & Lui, Y. M. (2010). "Visual Object Tracking using Adaptive Correlation Filters." [Електронний ресурс] – https://www.cs.colostate.edu/~draper/papers/bolme_cvpr10.pdf (дата звернення 21.05.2024).
4. "Object Tracking with DeepSORT and YOLO-NAS: Practitioner's Guide." [Електронний ресурс] – https://deci.ai/blog/object-tracking-with-deepsort-and-yolo-nas-practitioners-guide/ (дата звернення 23.05.2024).
5. "Hungarian Algorithm." [Електронний ресурс] – https://en.wikipedia.org/wiki/Hungarian_algorithm (дата звернення 20.05.2024).
6. Zhang, Yifu, et al. "FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking." International Journal of Computer Vision, vol. 129, no. 11, 2021, pp. 3069-3085.
7. “Joint Probabilistic Data Association Filter” [Електронний ресурс] - https://en.wikipedia.org/wiki/Joint_Probabilistic_Data_Association_Filter (дата звернення 20.05.2024).
8. “Joint probabilistic data association tutorial” [Електронний ресурс] - https://stonesoup.readthedocs.io/en/latest/auto_tutorials/08_JPDATutorial.html (дата звернення 21.05.2024).

9. Hung-Min Hsu, Tsung-Wei Huang, Gaoang Wang, Jiarui Cai, Zhichao Lei, and Jenq-Neng Hwang. Multi-camera tracking of vehicles based on deep features re-id and trajectory-based camera link models. In CVPR Workshops, pages 416–424, 2019.
10. Reid, D.B.:An algorithm for tracking multiple targets.IEEE Transactions on Automatic Control 24, 843–854, 1979.
11. Jean-Yves Bouguet, “Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm”, Intel Corporation Microprocessor Research Labs [Електронний ресурс] - http://robots.stanford.edu/cs223b04/algo_tracking.pdf (дата звернення 11.06.2024).
12. Tony Lacey , “Tutorial: The Kalman Filter” [Електронний ресурс] - https://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf (дата звернення 05.06.2024).
13. “Extended Kalman Filter” [Електронний ресурс] – https://ahrs.readthedocs.io/en/latest/filters/ekf.html (дата звернення 18.05.2024).
14. S. J. Julier and J. K. Uhlmann. Unscented Filtering and Nonlinear Estimation. Proceedings of the IEEE, 92(3):401–422, 2004
15. “Mean-Shift” [Електронний ресурс] - https://docs.opencv.org/4.x/d7/d00/tutorial_meanshift.html (дата звернення 16.05.2024).
16. "PYPL Popularity of Programming Language." [Електронний ресурс] – https://pypl.github.io/PYPL.html (дата звернення 20.05.2024).
17. “NumPy Introduction” [Електронний ресурс] - https://www.geeksforgeeks.org/introduction-to-numpy/ (дата звернення 12.06.2024).
18. “Introduction to NumPy” [Електронний ресурс] - https://medium.com/analytics-vidhya/introduction-to-numpy-279bbc88c615 (дата звернення 12.06.2024).

19."Download Python." [Електронний ресурс] – https://www.python.org/downloads/ (дата звернення 19.05.2024).
20."PyCharm: The Python IDE for Professional Developers." [Електронний ресурс] – https://www.jetbrains.com/pycharm/ (дата звернення 19.05.2024).

ДОДАТОК 1

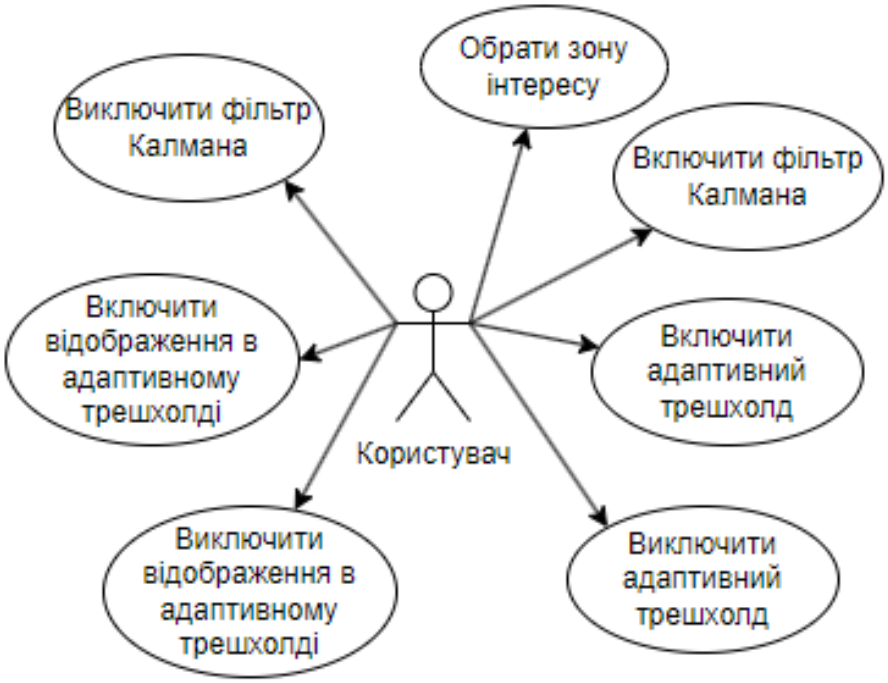
Система захоплення та супроводження об'єкту для автопілота.

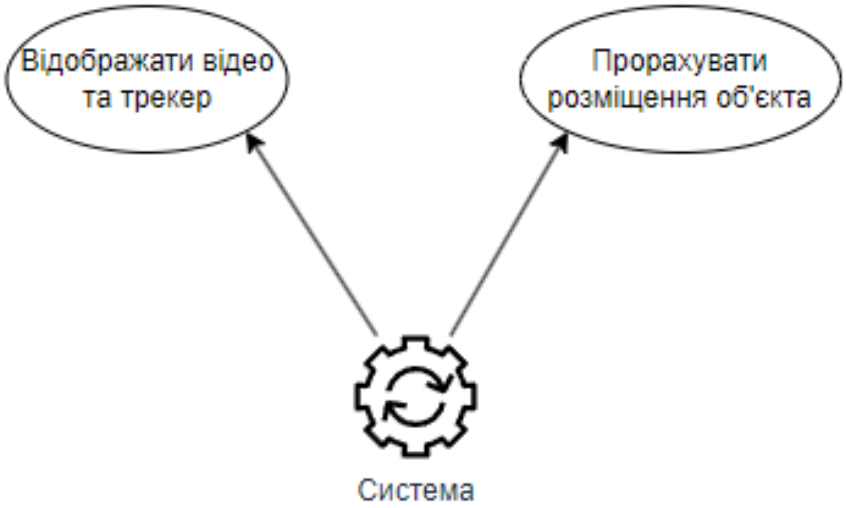
Структурна схема системи (діаграма прецедентів)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ – 2024





					ІАЛЦ.467200.004 Д1	
Зм.	Арк.	*№ докум.*	*Підпис*	*Дата*		
Розроб.		*Кубишка Ю.С.*			*Система захоплення та супроводження об'єкту для автопілота. Структурна схема системи (діаграма прецедентів)*	*Літ.* / *Аркуш* / *Аркушів*: 1 / 1
Перевір.		*Алєнін О.І.*				
Реценз.		Деведжіогуллари А.В.			КПІ ім. Ігоря Сікорського ФІОТ *Група ІО-04*	
Н. Контр.		*Іваніщев Б.В.*				
Затверд.		*Стіренко С.Г.*				

ДОДАТОК 2

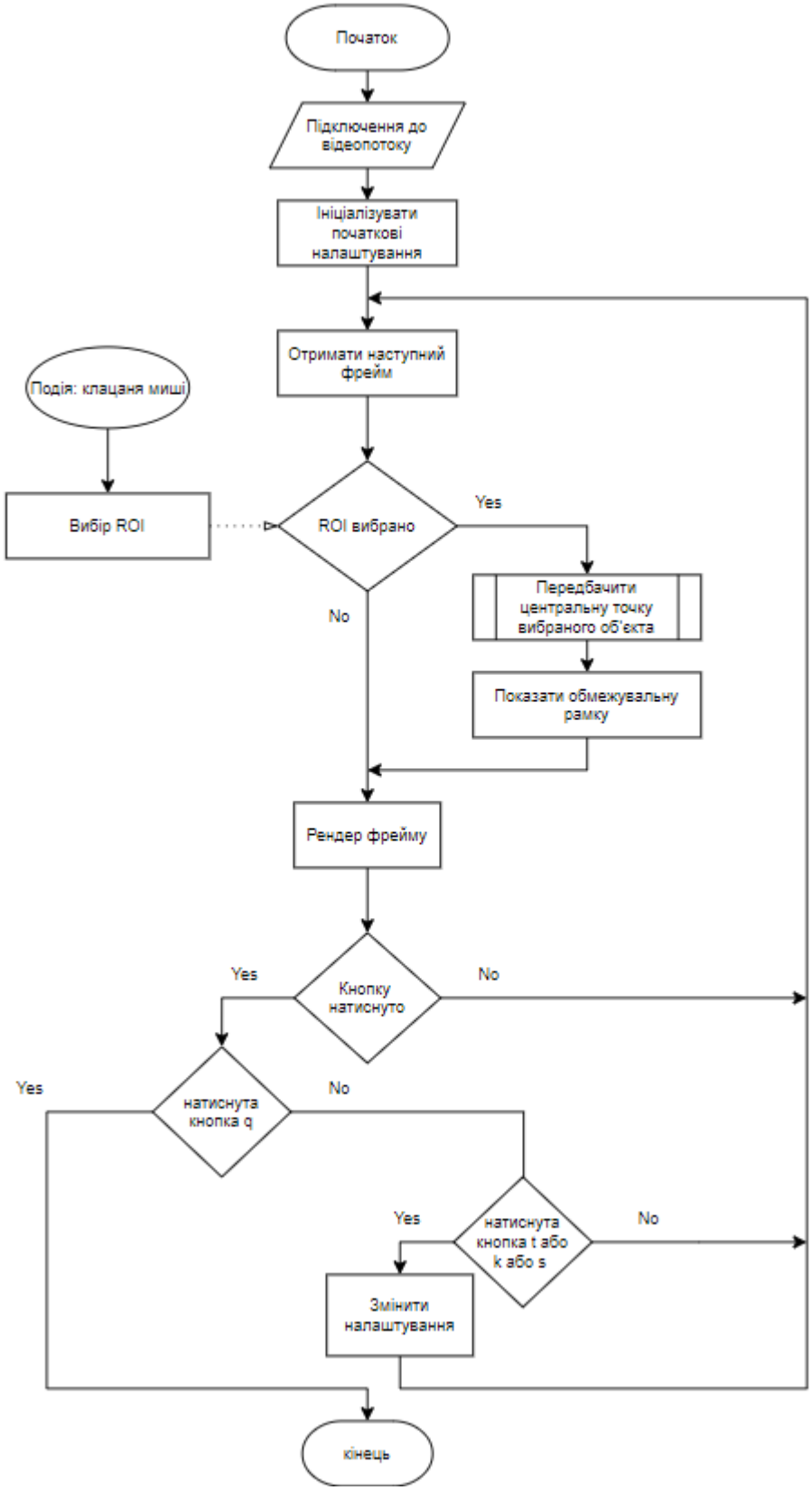
Система захоплення та супроводження об'єкту для автопілота.

Принципова схема роботи програми

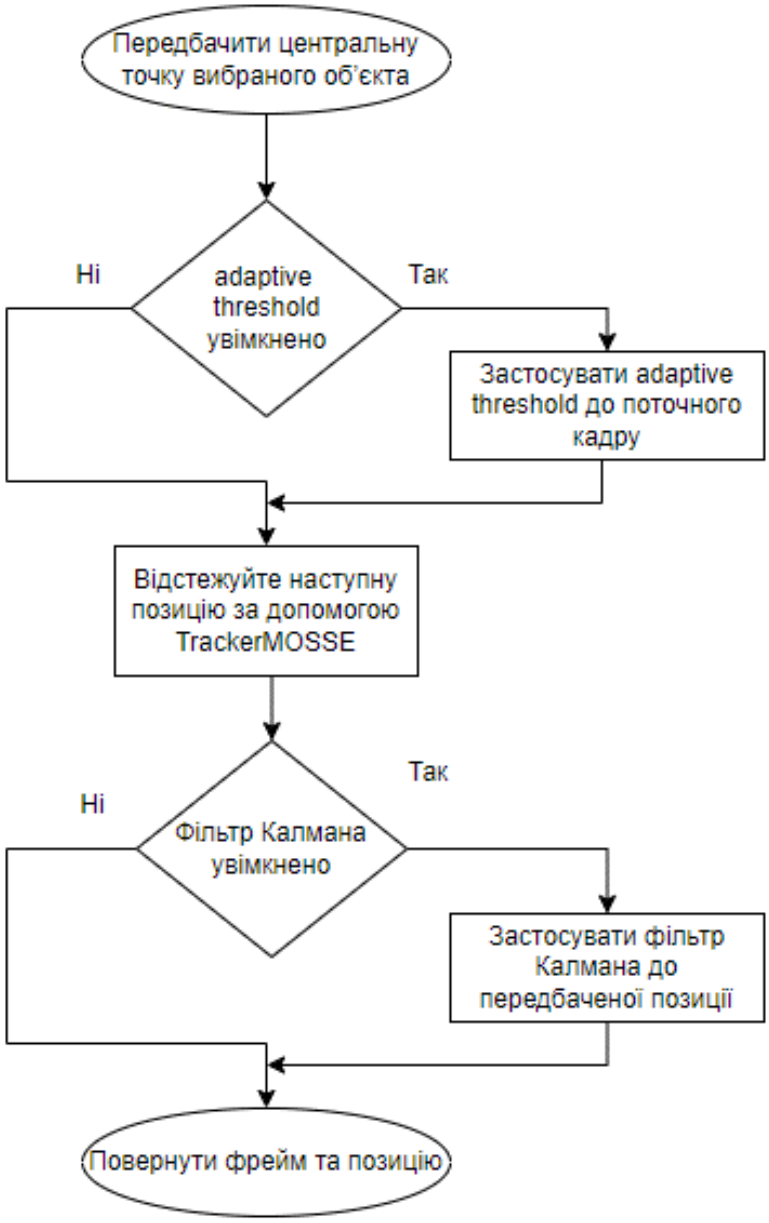
ІАЛЦ.467200.005 Д2

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.005 Д2	
Зм.	Арк.	*№ докум.*	*Підпис*	*Дата*		
Розроб.		*Кубишка Ю.С.*			*Система захоплення та супроводження об'єкту для автопілота. Принципова схема роботи програми*	*Літ.* / *Аркуш* / *Аркушів*: 1 / 2
Перевір.		*Алєнін О.І.*				
Реценз.		Деведжіогуллари А.В.				КПІ ім. Ігоря Сікорського ФІОТ *Група ІО-04*
Н. Контр.		*Іваніщев Б.В.*				
Затверд.		*Стіренко С.Г.*				


Передбачити центральну точку вибраного об'єкта
adaptive threshold увімкнено
Ні
Так
Застосувати adaptive threshold до поточного кадру
Відстежуйте наступну позицію за допомогою TrackerMOSSE
Фільтр Калмана увімкнено
Ні
Так
Застосувати фільтр Калмана до передбаченої позиції
Повернути фрейм та позицію

ДОДАТОК 3

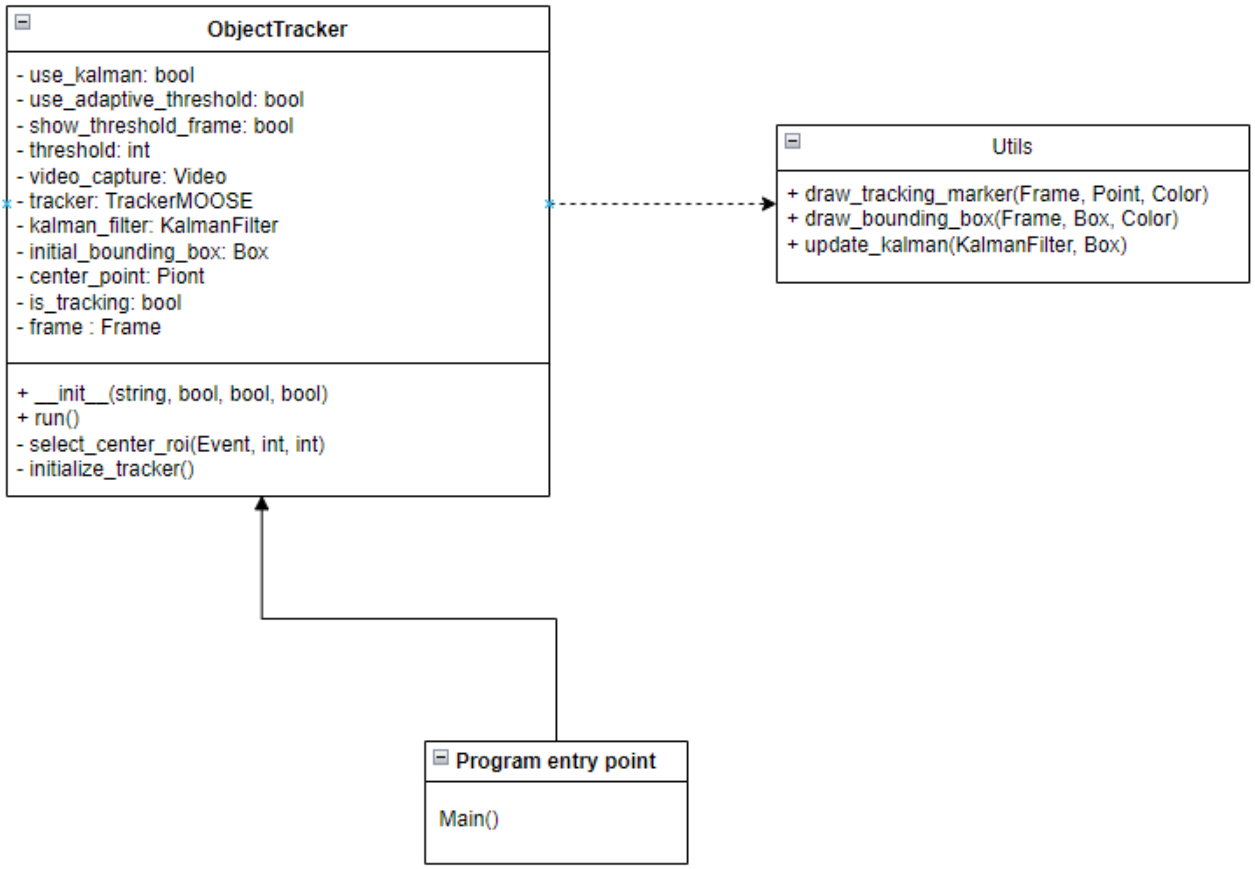
Система захоплення та супроводження об'єкту для автопілота.

Функціональна схема (діаграма класів)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.006 ДЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Кубишка Ю.С.			Система захоплення та супроводження об'єкту для автопілота. Функціональна схема (діаграма класів)	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Алєнін О.І.					
Реценз.		Деведжіогуллари А.В.				КПІ ім. Ігоря Сікорського ФІОТ Група ІО-04	
Н. Контр.		Іваніщев Б.В.					
Затверд.		Стіренко С.Г.					

ДОДАТОК 3

Система захоплення та супроводження об'єкту для автопілота.

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 4

Київ – 2024

main.py

```
import cv2
import numpy as np


def draw_tracking_marker(frame, center, color):
    radius = 3

    # Draw the lines
    cv2.line(frame, (center[0] - 25, center[1]), (center[0] + 25, center[1]),
color, 2)
    cv2.line(frame, (center[0], center[1] - 25), (center[0], center[1] + 25),
color, 2)

    # Draw the circle in the center
    cv2.circle(frame, center, radius, (0, 255, 0), -1)

    cv2.putText(frame, "Tracking", (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(75, 255, 75), 1)


def draw_bounding_box(frame, bounding_box, color):
    point1 = (int(bounding_box[0]), int(bounding_box[1]))
    point2 = (int(bounding_box[0] + bounding_box[2]), int(bounding_box[1] +
bounding_box[3]))
    cv2.rectangle(frame, point1, point2, color, 2, 1)
    cv2.putText(frame, "Tracking", (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255, 255, 255), 1)


def update_kalman(kalman_filter, bounding_box):
    center_x = bounding_box[0] + bounding_box[2] / 2
    center_y = bounding_box[1] + bounding_box[3] / 2
    measurement = np.array([[np.float32(center_x)],
                            [np.float32(center_y)],
                            [np.float32(bounding_box[2])],
                            [np.float32(bounding_box[3])]])
    kalman_filter.correct(measurement)
    prediction = kalman_filter.predict()
    predicted_bounding_box = (int(prediction[0] - prediction[4] / 2),
                              int(prediction[1] - prediction[5] / 2),
                              int(prediction[4]),
                              int(prediction[5]))
    return predicted_bounding_box


class ObjectTracker:
    def __init__(self, input_source, use_kalman=False,
use_adaptive_threshold=False, show_threshold_frame=False):
        self.use_kalman = use_kalman
        self.use_adaptive_threshold = use_adaptive_threshold
        self.show_threshold_frame = show_threshold_frame
        self.threshold = 9
        self.video_capture = cv2.VideoCapture(input_source)
```

```
        self.tracker = cv2.legacy.TrackerMOSSE.create()
        self.kalman_filter = cv2.KalmanFilter(6, 4)
        self.kalman_filter.measurementMatrix = np.array([[1, 0, 0, 0, 0, 0],
                                                          [0, 1, 0, 0, 0, 0],
                                                          [0, 0, 0, 0, 1, 0],
                                                          [0, 0, 0, 0, 0, 1]],
np.float32)
        self.kalman_filter.transitionMatrix = np.array([[1, 0, 1, 0, 0, 0],
                                                        [0, 1, 0, 1, 0, 0],
                                                        [0, 0, 1, 0, 0, 0],
                                                        [0, 0, 0, 1, 0, 0],
                                                        [0, 0, 0, 0, 1, 0],
                                                        [0, 0, 0, 0, 0, 1]],
np.float32)
        self.kalman_filter.processNoiseCov = np.eye(6, dtype=np.float32) * 0.05

        self.initial_bounding_box = None
        self.center_point = None
        self.is_tracking = False
        self.frame = None

    def select_center_roi(self, event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            self.is_tracking = False
            self.center_point = (x, y)

            frame_height, frame_width = self.frame.shape[:2]
            width = frame_width // 4
            height = frame_height // 3
            x0 = max(0, x - (width // 2))
            y0 = max(0, y - (height // 2))
            x0 = min(x0, frame_width - width)
            y0 = min(y0, frame_height - height)

            self.initial_bounding_box = (x0, y0, width, height)
            self.initialize_tracker()

    def initialize_tracker(self):
        self.is_tracking = True
        self.tracker = cv2.legacy.TrackerMOSSE.create()

        self.tracker.init(self.frame, self.initial_bounding_box)

        self.kalman_filter.statePre = np.array([[self.center_point[0]],
[self.center_point[1]],
                                                [0], [0],
                                                [self.initial_bounding_box[2]],
                                                [self.initial_bounding_box[3]]],
np.float32)
        self.kalman_filter.statePost = np.array([[self.center_point[0]],
[self.center_point[1]],
                                                 [0], [0],
                                                 [self.initial_bounding_box[2]],

[self.initial_bounding_box[3]]], np.float32)

    def run(self):
        cv2.namedWindow("Original")
        cv2.setMouseCallback("Original", self.select_center_roi)
        fps = self.video_capture.get(cv2.CAP_PROP_FPS)
        while True:
            start_time = cv2.getTickCount()
            ret, frame = self.video_capture.read()
```

```
            if not ret:
                break
            if self.use_adaptive_threshold:
                self.frame = cv2.adaptiveThreshold(cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY), 255,
                                                   cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY, self.threshold, 3)
                if self.show_threshold_frame:
                    frame = self.frame
            else:
                self.frame = frame

            cv2.putText(frame, "Status: ", (15, 50), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (255, 255, 255), 1)

            cv2.putText(frame, f"kalman={self.use_kalman},
adaptive_threshold={self.use_adaptive_threshold} ", (15, 25),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4, (255, 255, 255), 1)

            if self.is_tracking:
                success, bounding_box = self.tracker.update(self.frame)
                if success:
                    if self.use_kalman:
                        predicted_bounding_box =
update_kalman(self.kalman_filter, bounding_box)
                    else:
                        predicted_bounding_box = bounding_box
                    center_point = (int(predicted_bounding_box[0] +
predicted_bounding_box[2] / 2),
                                    int(predicted_bounding_box[1] +
predicted_bounding_box[3] / 2))
                    draw_tracking_marker(frame, center_point, (55, 255, 255))
                    draw_bounding_box(frame, predicted_bounding_box, (255, 255,
0))

                else:
                    self.is_tracking = False

            cv2.imshow("Original", frame)
            end_time = cv2.getTickCount()
            time_taken = (end_time - start_time) / cv2.getTickFrequency()
            delay_time_ms = max(1, int((1 / fps - time_taken) * 1000))
            key = cv2.waitKey(delay_time_ms)
            if key != -1:
                if key == ord('q'):
                    break
                elif key == ord('k'):
                    self.use_kalman = not self.use_kalman
                elif key == ord('t'):
                    self.use_adaptive_threshold = not
self.use_adaptive_threshold
                elif key == ord('s'):
                    self.show_threshold_frame = not self.show_threshold_frame
                elif key == ord('+'):
                    self.threshold += 2
                elif key == ord('-'):
                    self.threshold = self.threshold - 2 if self.threshold > 3
else 3

        self.video_capture.release()
        cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    INPUT_CAMERA_0 = 0
    INPUT_CAMERA_1 = 'video\\drone1.mp4'
    tracker = ObjectTracker(INPUT_CAMERA_0, True, True)
    tracker.run()
```