Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей виявлення і спостереження рухомих об'єктів.
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-3

Лазаренко С.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник проф. Смеляков К.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук (або центр післядипломної освіти, або навчально-науковий центр заочної форми навчання)
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Лазаренко Станіславу Валентиновичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження моделей виявлення і спостереження рухомих об’єктів»
Затверджена наказом по університету від 29.03.2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2022
3. Вихідні дані до роботи досліджуваних моделей для виявлення та спостереження рухомих об'єктів, вимоги до розробки алгоритмів трекінгу та детекції об'єктів, специфікації для налаштування системи трекінгу, мови програмування Python, технології та бібліотеки OpenCV, YOLOv8, DeepSORT, середовища розробки PyCharm, операційні системи macOS
4. Перелік питань, що потрібно опрацювати в роботі
аналіз предметної галузі, огляд наявних математичних моделей, модифікація базових алгоритмів, дослідження можливості прискорення базових моделей, створення плану експерименту для дослідження, опис імплементації алгоритмів, дослідження результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	23.01 – 14.02.24	*виконано*
2	Аналіз та вибір моделей для дослідження	15.02 – 24.02.24	*виконано*
3	Аналіз та моделювання предметної області	17.02 – 28.02.24	*виконано*
4	Планування експериментів	25.02 – 28.02.24	*виконано*
5	Програмна реалізація кожної з обраних для дослідження моделі	25.02 – 01.04.24	*виконано*
6	Експериментальні дослідження	02.04 – 20.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	20.04 – 23.04.24	*виконано*
8	Написання та оформлення статті та тез доповіді	17.04 – 23.04.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 26.04.24	*виконано*
10	Підготовка презентації та доповіді	26.04 – 2.05.24	*виконано*
11	Нормоконтроль	15.06 – 18.06.24	*виконано*
12	Рецензування	10.06 – 14.06.24	*виконано*
13	Занесення диплома в електронний архів	18.06.2024	*виконано*
14	Попередній захист	19.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	19.06.2024	*виконано*

Дата видачі завдання 20 січня 2023р.

Студент (ка) _____ Лазаренко С.В.
(підпис)

Керівник роботи _____ проф. Смеляков К.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 66 стор., 14 рис., 1 табл., 16 джерел.

КОМП’ЮТЕРНИЙ ЗІР, ВІДЕОДЕТЕКЦІЯ, КЛАСИФІКАЦІЯ, ОПТИМІЗАЦІЯ КОМП’ЮТЕРНОГО ЗОРУ.

Об’єкт дослідження – системи відеодетекції та класифікації.

Мета роботи – Розробка та аналіз систем відеодетекції з використанням Raspberry Pi для ідентифікації рухомих літаючих об’єктів у складних умовах.

Результат роботи – Розроблено концепцію та описано алгоритми відеодетекції, а також розроблено прототип веб додатку для системи детекції.

COMPUTER VISION, VIDEO DETECTION, VIDEO ANALYSIS, RASPBERRY PI.

Object of research - video detection and classification systems.

Purpose - Development and analysis of video detection and classification systems using Raspberry Pi for flying object identification.

The result of the work - Developed the concept and described the algorithms of video detection.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Лазаренко Станіслав Валентинович , студент(ка) гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження моделей виявлення і спостереження рухомих об’єктів.», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

YOLO – You Look Only Once

FPS – Frames Past Second

KCF – Kernelized Correlation Filters

CSRT – Discriminative Correlation Filter with Channel and Spatial Reliability

MOSSE – Minimum Output Sum of Squared Error

ВСТУП

У сучасному світі технології розвиваються з неймовірною швидкістю, пропонуючи нові можливості для розв'язання складних завдань. Однією з таких областей є комп'ютерний зір, який відіграє ключову роль у розробці інноваційних систем відео детекції. Комп'ютерний зір дозволяє машинам інтерпретувати та обробляти візуальну інформацію, що відкриває нові горизонти у таких сферах, як безпека, навігація, і навіть у розвитку розважальних технологій.

Цей дипломний проект зосереджений на розробці та аналізі систем відео детекції з використанням Raspberry Pi для ідентифікації складних рухомих об'єктів. Особлива увага приділяється розробці ефективних алгоритмів, які здатні точно та швидко ідентифікувати ці об'єкти в різних умовах. У контексті зростаючих вимог до безпеки та працездатності різних систем, розробка надійних систем детекції рухомих об'єктів є актуальною і важливою задачею.[2]

Проект також включає розробку алгоритмів класифікації об'єктів, які допомагають визначити тип рухомого об'єкта, що значно підвищує ефективність системи детекції. Це дозволить створити більш комплексну систему, здатну ефективно працювати у різних умовах і відповідати високим вимогам сучасного технологічного світу.

На завершення, проект передбачає розробку мобільного додатку, який забезпечує користувачам можливість моніторингу і управління системою детекції в реальному часі. Цей аспект роботи підкреслює практичне застосування розробленої системи та її значення в сучасному технологічному світі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної області

Сучасний світ вимагає ефективних та надійних систем безпеки, особливо у контексті збільшення використання складних рухомих об'єктів, таких як безпілотні літальні апарати, автономні транспортні засоби, роботи і т.д. Ці об'єкти можуть використовуватися як для цивільних, так і для військових цілей, що ставить питання про необхідність ефективних методів їх виявлення та ідентифікації.

Детекція та відстеження рухомих об'єктів є складною задачею через їхні різні розміри, високу маневреність та різноманітність форм. Тому розробка ефективних алгоритмів комп'ютерного зору для детекції та відстеження цих об'єктів є актуальною та важливою задачею.

У цьому дослідженні ми прагнемо розробити та аналізувати систему детекції складних рухомих об'єктів, яка використовує алгоритми комп'ютерного зору на базі Raspberry Pi. Основна увага буде приділена створенню ефективних та точних алгоритмів для виявлення та відстеження об'єктів, які можуть бути впроваджені у різних умовах експлуатації.

1.2 Огляд сучасних методів та інструментів

Сьогодні існує багато методів та інструментів для автоматичної детекції та відстеження рухомих об'єктів. Вони включають, наприклад, методи оптичного потоку, методи виявлення зміни фону, методи машинного навчання, такі як глибоке навчання, тощо.

Однак, кожен з цих методів має свої обмеження та вимоги до обчислювальної потужності, якість вхідних даних, освітлення, швидкість руху об'єктів і т.д. Тому важливо вибрати найбільш підходящий метод для конкретних умов експлуатації.

Також існує ряд програмних бібліотек та інструментів, які можуть бути використані для розробки системи детекції та відстеження рухомих об'єктів, такі як OpenCV, TensorFlow, Keras, YOLO, тощо. Вони надають широкий спектр

алгоритмів та функцій для обробки зображень та відео, машинного навчання, глибокого навчання, тощо.

В рамках цього дослідження ми плануємо використовувати Raspberry Pi як низьковартісну і енергоефективну платформу для реалізації нашої системи детекції та відстеження рухомих об'єктів.[3] Raspberry Pi має достатню обчислювальну потужність для реалізації більшості сучасних алгоритмів детекції та відстеження, а також підтримує широкий спектр периферійних пристроїв, включаючи камери, сенсори, тощо.

Одним з головних викликів цього дослідження є намагання досягнути максимальної швидкості та ефективності детекції та відстеження рухомих об'єктів у реальному часі. Реальний час відстеження вимагає високої швидкості обробки та мінімальної затримки, що є важливим для багатьох застосувань, особливо у сфері безпеки та навігації. Однак, це також ставить високі вимоги до обчислювальної потужності та ефективності використаних алгоритмів.

У цьому контексті, використання Raspberry Pi як платформи для реалізації системи детекції та відстеження може бути викликом, оскільки ця платформа має обмеження щодо обчислювальної потужності та ресурсів пам'яті. Однак, Raspberry Pi також пропонує ряд переваг, таких як низька вартість, мале споживання енергії, гнучкість та велика підтримка спільноти, що робить його привабливим вибором для багатьох проектів. Тому, ключовим аспектом цього дослідження буде розробка та оптимізація алгоритмів детекції та відстеження, які можуть ефективно працювати на обмеженому обладнанні Raspberry Pi.

На сьогоднішній день існує чимало алгоритмів для відстеження об'єктів у відео, від простих математичних моделей до складних архітектур глибокого навчання. Кожен з цих алгоритмів має свої переваги та недоліки, а також вимоги до обчислювальної потужності та якості вхідних даних. У даній роботі розглянуто декілька сучасних методів детекції та трекінгу об'єктів, а також проведено експериментальне порівняння їх ефективності. Зокрема, перевірені такі алгоритми:

- BoT-SORT

  BoT-SORT зберігає треки між кадрами, що підвищує точність.

- DeepSORT

  DeepSORT використовує глибокі нейронні мережі для асоціації об'єктів між кадрами.

- Kalman Filter with MeanShift

  Комбінація фільтру Калмана для передбачення положення об'єктів та алгоритму MeanShift для корекції положення. Використовує гістограму кольорів для ідентифікації об'єктів (див. рис. 1.1).

- ByteTrack

  ByteTrack підвищує точність відстеження за рахунок використання передбачень трекера.

- CSRT

  CSRT забезпечує високу точність трекінгу.

- KCF

  KCF забезпечує швидкий трекінг з середньою точністю.

- MOSSE

  MOSSE забезпечує найвищий FPS серед усіх розглянутих алгоритмів, але знижує точність.

Videos	**KCF**		**CSRT**		**KCF + CSRT**	
	Highest FPS	**Average FPS**	**Highest FPS**	**Average FPS**	**Highest FPS**	**Average FPS**
Video 1	48	42	36	32	47	41
Video 2	44	39	33	29	46	36
Video 3	49	45	36	30	48	42
Video 4	41	35	31	28	40	34
Video 5	45	40	32	28	46	40

Рисунок 1.1 – Порівняння швидкості алгоритмів KCF, CSRT, KCF+CSRT (за даними [1])

У рамках цього дослідження ми плануємо зосередитися на використанні OpenCV та YOLOv8. OpenCV - це відкрита бібліотека алгоритмів комп'ютерного зору та машинного навчання, яка має широкий спектр функцій для обробки зображень та відео. YOLOv8 - це алгоритм глибокого навчання для виявлення об'єктів, який використовує конволюційну нейронну мережу для одночасного прогнозування класу об'єкта та його положення на зображенні (див. рис. 1.2).

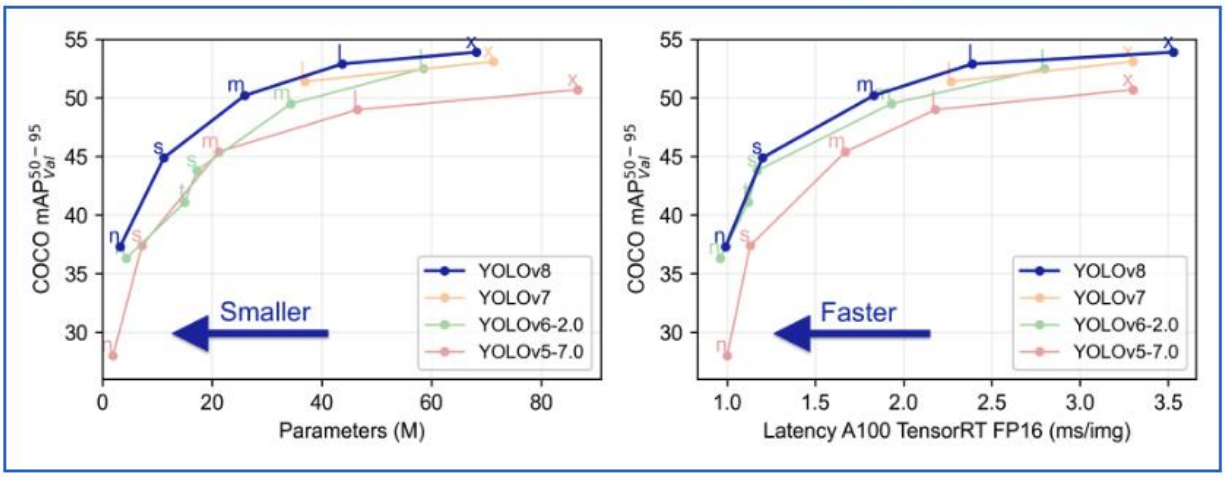

Рисунок 1.2 – Порівняння швидкості алгоритмів YOLO (за даними [2])

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Огляд математичних моделей

Перед тим як перейти до модифікації базових алгоритмів детекції та трекінгу об'єктів, розглянемо їх математичне представлення. Розглянемо модулі детекції руху, групування bounding boxes, оновлення та управління трекерами, а також класифікацію за допомогою YOLO[4]. Додатково розглянемо функції градієнтної корекції та сегментації зображення.

2.2 Модуль детекції руху

Модуль детекції руху використовує різницю між двома послідовними кадрами для виявлення руху. Це досягається за допомогою обробки зображень.

Функція motion_detection(frame1, frame2) виконує наступні кроки:

- Обчислення різниці між двома кадрами за формулою 2.1:

$$D = | F_1 - F_2 | \tag{2.1}$$

де F_1 і F_2 - це послідовні кадри відео,

D – різниця між двома послідовними кадрами.

- Перетворення в градації сірого за формулою 2.2:

$$G = \text{Gray}(D) \tag{2.2}$$

де D – різниця між двома кадрами

G – кадр в градаціях сірого

- Гауссове згладжування за формулою 2.3:

$$B = \text{GaussianBlur}(G, ksize = 5) \tag{2.3}$$

де B – згладжене зображення.

G– кадр в градаціях сірого.

ksize – розмір ядра для згладжування (в даному випадку 5).

- Порогова обробка за формулою 2.4:

$$BT = \begin{cases} 255 & \text{якщо } B > 20 \\ 0 & \text{інакше} \end{cases} \quad (2.4)$$

де BT – бінарне зображення після порогової обробки.

B – згладжене зображення.

- Дилатація за формулою 2.5:

$$D_T = \text{dilate}(T, \text{kernel}, \text{iterations} = 3) \quad (2.5)$$

де D_T – зображення після дилатації,

T – бінарне зображення після порогової обробки,

kernel – ядро для дилатації,

iterations – кількість ітерацій дилатації (в даному випадку

- Пошук контурів за формулою 2.6:

$$C = \text{findContours}(D_T) \quad (2.6)$$

де C – знайдені контури на зображенні,

D_T – зображення після дилатації.

2.3 Групування bounding boxes

Групування bounding boxes виконується для об'єднання перекриваючих або вкладених прямокутників.

Функція merge_boxes(boxes) виконує наступні кроки:

- Перевірка перекриття двох прямокутників за формулою 2.7:

$$\text{overlap}(A, B) = \text{True якщо} A \cap B \neq \emptyset \quad (2.7)$$

де A – перший прямокутник,

B – другий прямокутник,

overlap(A,B) – логічне значення, яке вказує, чи перекриваються прямокутники.

- Об'єднання двох прямокутників за формулою 2.8:

$$\begin{aligned} \text{new\_box} = (&\min(x_A, x_B), \min(y_A, y_B), \max(x_A + w_A, x_B + w_B) \\ &- \min(x_A, x_B), \max(y_A + h_A, y_B + h_B) \\ &- \min(y_A, y_B)) \end{aligned} \quad (2.8)$$

де new_box – новий об'єднаний прямокутник,

x_A, y_A- координати верхнього лівого кута першого прямокутника A,

x_B, y_B - координати верхнього лівого кута другого прямокутника B,

w_A, h_A - ширина та висота першого прямокутника A,

w_B, h_B- ширина та висота другого прямокутника B

2.4 Оновлення та управління трекерами

Оновлення та управління трекерами забезпечують відстеження рухомих об'єктів протягом відеопотоку.

Функція update_trackers_with_overlap(trackers, grouped_boxes, frame_number) виконує наступні кроки:

- Оновлення позиції трекера за формулою 2.9:

$$T_i^{(k+1)} = \begin{cases} T_i^{(k)} \cup \{(B_j, f_k)\} & \text{якщо overlap}(T_i^{(k)}, B_j) \\ T_i^{(k)} & \text{інакше} \end{cases} \quad (2.9)$$

де $T_i^{(k)}$ - трекер i на кадрі k,

B_j - bounding box j на кадрі k,

f_k - номер кадру.

- Ініціалізація нового трекера за формулою 2.10:

$$T_{new} = \{(B_j, f_k)\} \quad (2.10)$$

де T_{new} - новий трекер,

B_j- bounding box об'єкта j,

f_k- номер кадру k.

2.5 Класифікація трекерів за допомогою YOLO

Класифікація трекерів за допомогою моделі YOLO дозволяє визначити, чи є відстежуваний об'єкт об’єктом.

Функція yolo_classify_drones(trackers: Set[Any], current_frame: Any) виконує наступні кроки:

- Детекція за допомогою моделі YOLO за формулою 2.11:

$$\mathcal{D} = \text{YOLO}(F) \quad (2.11)$$

де D- множина детекцій,

F - поточний кадр.

- Оновлення статусу трекера за формулою 2.12:

$$\text{update\_status}(T_i, \mathcal{D}) \rightarrow T_i \quad (2.12)$$

де T_i трекер i, оновлюється на основі детекцій D.

2.6 Градієнтна корекція зображення

Градієнтна корекція зображення використовується для покращення якості зображення шляхом нормалізації інтенсивності пікселів.

Функція apply_gradient_correction(image, min_val, max_val) виконує наступні кроки:

- Перетворення зображення у формат з плаваючою комою за формулою 2.13:

$$I_f = \text{float}(I) \tag{2.13}$$

де I_f - зображення у форматі з плаваючою комою,

I – вхідне зображення.

- Нормалізація інтенсивності за формулою 2.14:

$$I_n = \frac{I_f - \min(I_f)}{\max(I_f) - \min(I_f)} \tag{2.14}$$

де I_n- нормалізоване зображення,

I_f - зображення у форматі з плаваючою комою.

- Кліппінг нормалізованого зображення за формулою 2.15:

$$I_c = \text{clip}(I_n, 0{,}1) \times 255 \tag{2.15}$$

де I_c - кліповане нормалізоване зображення,

I_n - нормалізоване зображення.

- Перетворення назад у 8-бітний формат за формулою 2.16:

$$I_{out} = \text{uint8}(I_c) \tag{2.16}$$

де I_{out} - вихідне зображення у 8-бітному форматі,

I_c - кліповане нормалізоване зображення.

Функція find_optimal_min_max_values(image, percentile=1) визначає оптимальні мінімальні та максимальні значення інтенсивності для нормалізації:

- Обчислення гістограми за формулою 2.17:

$$h = \text{histogram}(I, bins = 256) \tag{2.17}$$

де h – гістограма інтенсивності,

I – вхідне зображення.

- Обчислення кумулятивної функції розподілу (CDF) за формулою 2.18:

$$cdf = \text{cumsum}(h) \tag{2.18}$$

де cdf – кумулятивна функція розподілу.

- Нормалізація CDF за формулою 2.19:

$$cdf_n = \frac{cdf}{\max(cdf)} \times 100 \tag{2.19}$$

де cdf – кумулятивна функція розподілу,

cdf_n - нормалізована кумулятивна функція розподілу.

- Знаходження оптимальних значень за формулою 2.20:

$$\text{max\_val} = \text{percentile}(cdf_n, 99) \tag{2.20}$$

де max_val – оптимальне максимальне значення інтенсивності,

cdf_n - нормалізована кумулятивна функція розподілу.

2.7Сегментація зображення

Функція get_predictions(frame) виконує сегментацію зображення для виявлення зон, які не будуть використовуватися для детекції.

- Виявлення країв за формулою 2.21:

$$E = \text{detect\_edges}(F, 3) \tag{2.21}$$

де E – знайдені краї.

F – кадр відео.

- Отримання сирих прогнозів за формулою 2.22:

$$P = \text{get\_raw\_predictions}(F, E) \tag{2.22}$$

де P – сирі прогнози.

F – кадр відео.

E – знайдені краї.

- Фільтрація прогнозів для визначених класів за формулою 2.23:

$$S = \{(mask, class) \in P \mid class \in \{\text{"tree","structure"}\}\} \tag{2.23}$$

де S – відфільтровані прогнози.

P – сирі прогнози.

mask – маска області.

class – клас об'єкта (наприклад, дерево або структура).

2.8 Висновок

Розглянуті математичні моделі детекції та трекінгу рухомих об'єктів, градієнтної корекції та сегментації зображення дозволяють зрозуміти, як працюють алгоритми в рамках обробки відеопотоків. Кожна з функцій має чітке математичне представлення, що допомагає в подальшій розробці та вдосконаленні алгоритмів.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі буде описано реалізацію алгоритму детекції та трекінгу рухомих об’єктів, який використовує модель YOLOv8 для детекції, а також додаткові операції обробки зображення, такі як корекція градієнта та сегментація зображення.

3.1 Загальна архітектура

Реалізація алгоритму детекції та трекінгу рухомих об’єктів складається з декількох основних модулів:

Завантаження та попередня обробка відео:

- Відео розділяється на окремі кадри.
- Застосовується корекція градієнта для покращення якості зображення.

Детекція рухомих об’єктів:

- Використовується модель YOLOv8 для детекції рухомих об’єктів на кожному кадрі.

Сегментація зображення:

- Використовується механізм класифікації для видалення зон, які не повинні братися до уваги при детекції.

Трекінг об'єктів:

- Використовується кастомний алгоритм трекінгу для відстеження об’єктів між кадрами.

3.2 Завантаження та попередня обробка відео

Для завантаження відео використовується клас VideoCapture, який реалізує патерн Singleton для забезпечення єдиного екземпляра об'єкта. Попередня обробка кадрів включає корекцію градієнта для нормалізації інтенсивності пікселів.

```
class VideoCapture(metaclass=SingletonMeta):
    def __init__(self, file_path: Optional[str] = None) -> None:
        # Initialization code here...

    def open(self) -> None:
        # Open video source...

    def read(self) -> Optional[np.ndarray]:
        # Read a frame from the video source...

def apply_gradient_correction(image, min_val, max_val):
    image_float = image.astype(np.float32)
    normalized_image = (image_float - min_val) / (max_val - min_val)
    normalized_image = np.clip(normalized_image, 0, 1)
    corrected_image = (normalized_image * 255).astype(np.uint8)
    return corrected_image

def initial_frames_process(video_capture, min_val, max_val):
    previous_frame = apply_gradient_correction(video_capture.frame1,
min_val, max_val)
    current_frame = apply_gradient_correction(video_capture.frame2,
min_val, max_val)
    return previous_frame, current_frame

def find_optimal_min_max_values(image, percentile=1):
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * (100.0 / cdf.max())
    optimal_min_val = np.searchsorted(cdf_normalized, percentile)
    optimal_max_val = np.searchsorted(cdf_normalized, 100 -
percentile)
    return optimal_min_val, optimal_max_val
```

3.3 Детекція об’єктів

Детекція об’єктів виконується за допомогою моделі YOLOv8. Кадри обробляються моделлю для отримання bounding boxes об’єктів.

```
def predict_boxes(current_frame: Any) -> Any:
    result = model.predict(current_frame)[0]
    boxes = extract_boxes(result)
    return boxes
```

3.4 Сегментація зображення

Сегментація зображення використовується для видалення непотрібних зон з кадрів, таких як дерева чи будівлі (див. рис. 3.1).

```
def segmentation_predictions(frame):
    dilated_edges = detect_edges(frame, 3)
    predictions = get_raw_predictions(frame, dilated_edges)
    return [(mask, class_name) for mask, class_name in predictions if
class_name in ["tree", "structure"]]

def is_overlapping_with_segment(box, segmentation_mask):
    x, y, w, h = box
    box_mask = np.zeros_like(segmentation_mask)
    cv2.rectangle(box_mask, (x, y), (x + w, y + h), 255, -1)
    overlap = cv2.bitwise_and(box_mask, segmentation_mask)
    return np.any(overlap)
```

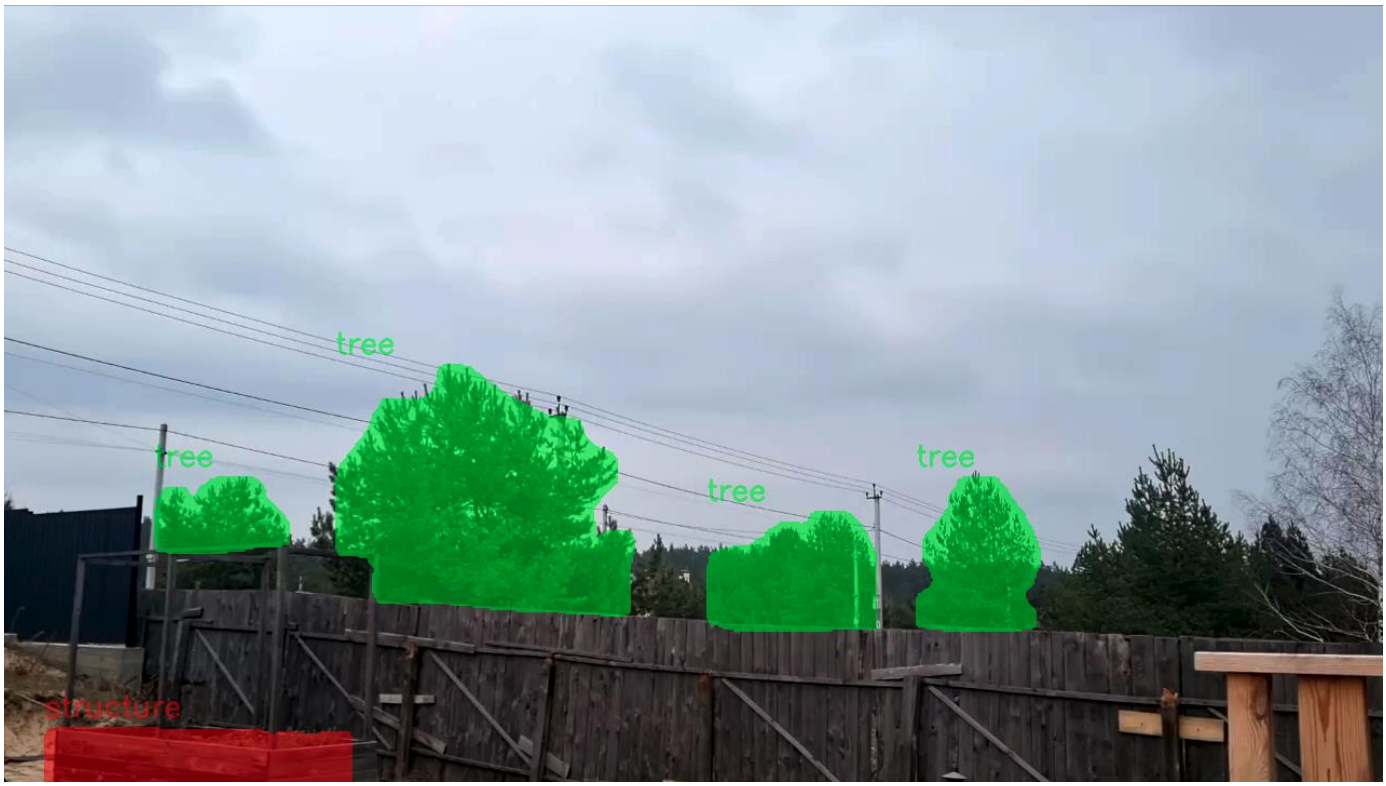

Рисунок 3.1 – Сегментація зображення і накладення зон (виконано самостійно)

3.5 Трекінг об'єктів

Для трекінгу об'єктів використовується кастомний алгоритм, який включає ініціалізацію трекерів, оновлення їх позицій та класифікацію трекерів за допомогою YOLO.

```
class DroneTracker:
    def __init__(self):
```

```
            self.positions = []
            self.is_drone = False
            self.kcf_tracker = None
            self.is_active = False

    def init_tracker(box, frame_number):
        new_tracker = DroneTracker()
        new_tracker.positions.append((box, frame_number))
        return new_tracker

    def update_trackers_with_overlap(trackers, grouped_boxes,
frame_number):
        new_trackers = set()
        for box in grouped_boxes:
            matched = False
            for tracker in trackers:
                last_box, _ = tracker.positions[-1]
                if check_overlap(last_box, box):
                    tracker.positions.append((box, frame_number))
                    new_trackers.add(tracker)
                    matched = True
                    break
            if not matched:
                new_tracker = init_tracker(box, frame_number)
                new_trackers.add(new_tracker)
        trackers.update(new_trackers)
        return trackers

    def yolo_classify_drones(trackers: Set[Any], current_frame: Any) ->
None:
        for tracker in trackers:
            if not tracker.is_drone and len(tracker.positions) >= 5:
                last_position, _ = tracker.positions[-1]
                predicted_boxes = predict_boxes_for_roi(current_frame,
last_position)
                if predicted_boxes is not None:
                    update_tracker_status(tracker, predicted_boxes,
trackers)
```

Приклад трекінгу:

Початок трекінгу:

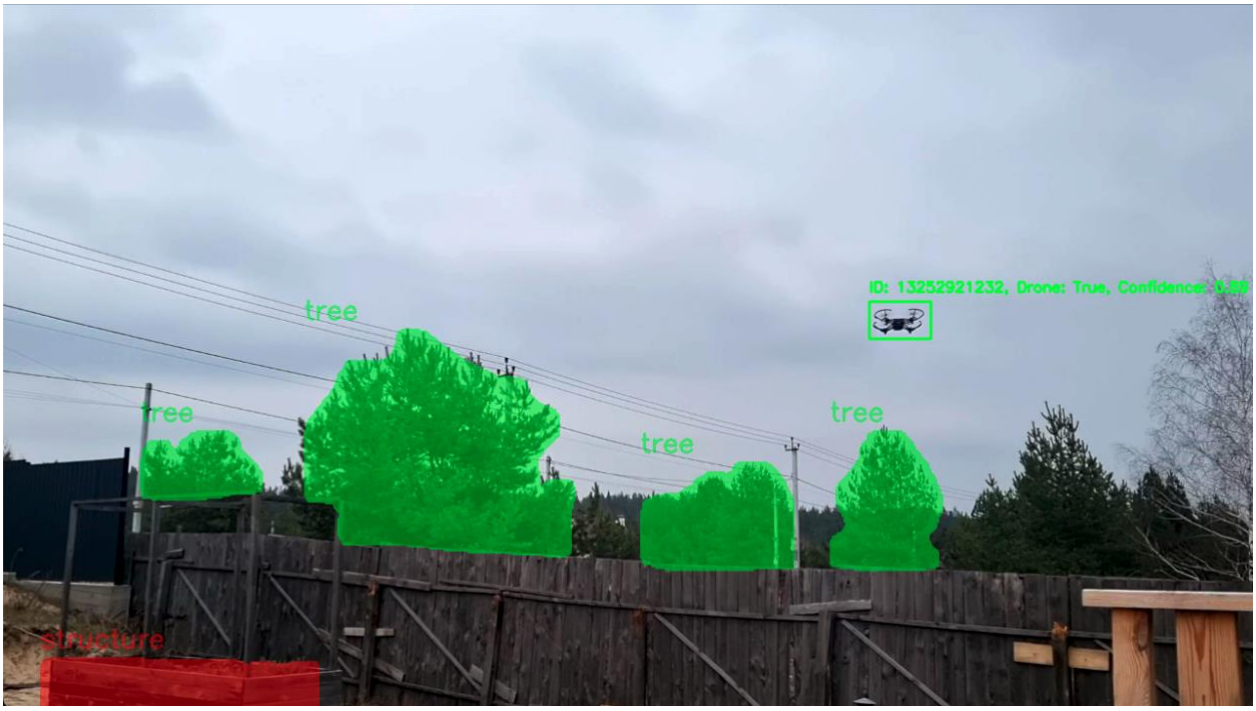

Рисунок 3.2 – Приклад візуалізації трекінгу (виконано самостійно)

На першому зображенні показано, як об'єкт (об’єкт) було вперше виявлено та ініціалізовано трекер (див. рис. 3.2).

Коли об’єкт залітає в зону, яка позначена як дерево:

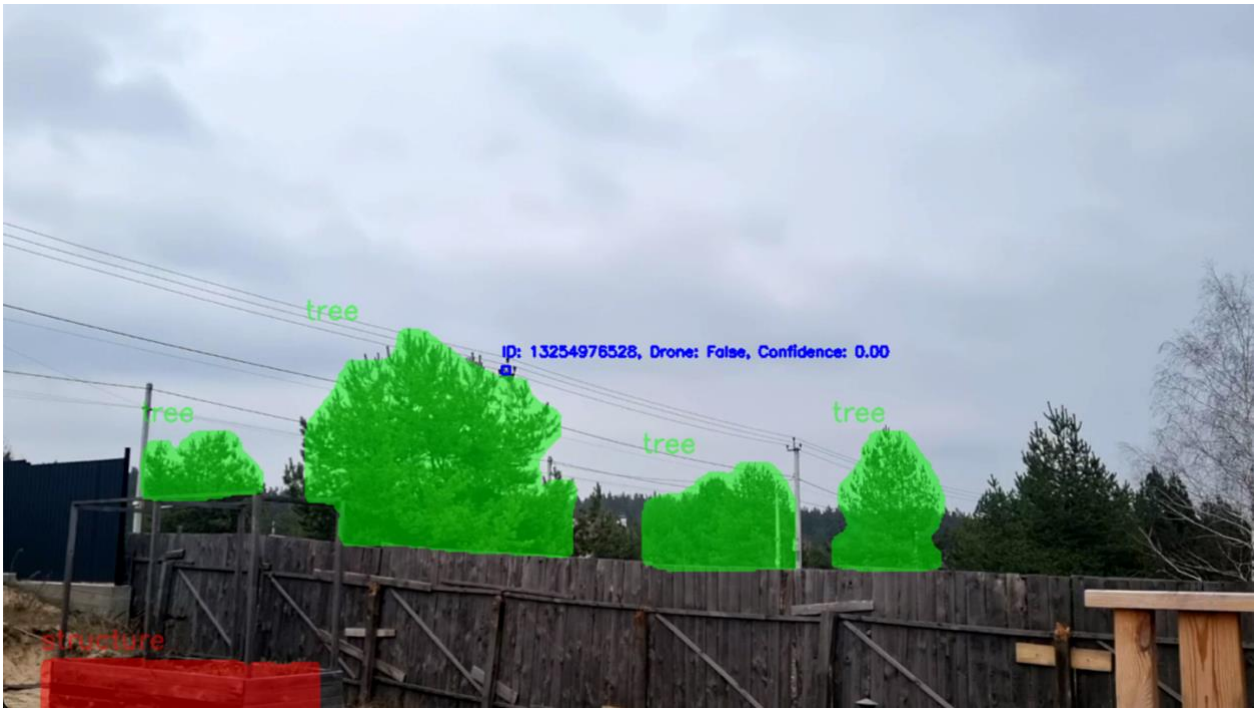

Рисунок 3.3 – Об’єкт втрачено із-за того що він залетів в зону дерев (виконано самостійно)

На цьому зображенні показано, як об’єкт залітає в зону, позначену як дерево, де трекінг не відбувається (див. рис. 3.3).

Коли об’єкт вилітає із зони дерев:

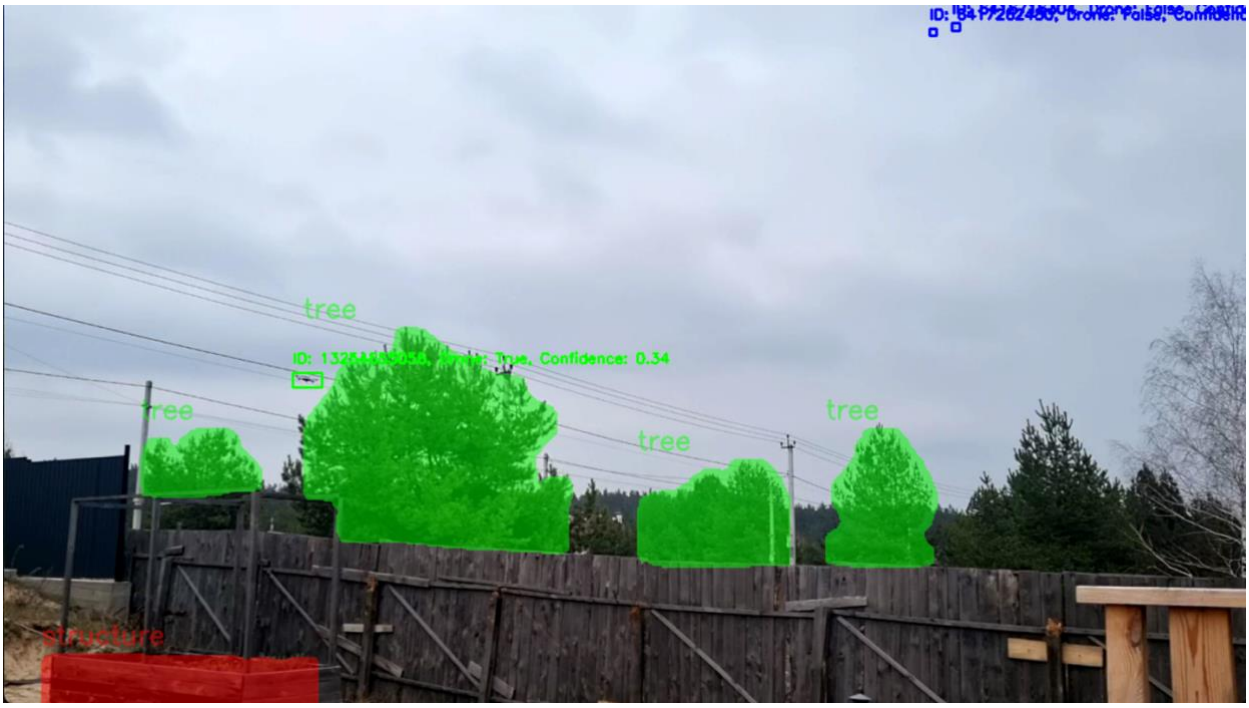

Рисунок 3.4 – Повторна детекція об’єкту (виконано самостійно)

На цьому зображенні показано, як об’єкт вилітає із зони дерев, після чого його знову виявлено та класифіковано (див. рис. 3.4).

Кожен з цих кроків ілюструє певний етап процесу трекінгу об'єктів. На першому етапі об'єкт виявляється та ініціалізується трекер. На другому етапі об'єкт залітає в зону, де його не трекають (наприклад, через наявність дерев), і трекінг призупиняється. На третьому етапі об'єкт знову виявляється після виходу із зони, де трекінг не здійснювався.

3.6 Візуалізація результатів

Для візуалізації результатів використовується функція draw_detections_and_trackers, яка малює bounding boxes та статус трекерів на кадрі.

```
def draw_detections_and_trackers(frame, trackers):
    local_frame = frame.copy()
    for tracker in trackers:
```

```
            box, _ = tracker.positions[-1]
            color = Colors.BLUE if not tracker.is_drone else Colors.GREEN
            cv2.rectangle(local_frame, (box[0], box[1]), (box[0] +
box[2], box[1] + box[3]), color, 2)
            text = f"ID: {id(tracker)}, Drone: {tracker.is_drone}"
            position = (box[0], box[1] - 10)
            cv2.putText(local_frame, text, position,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
        return local_frame
```

3.7 Основний цикл обробки відео

Основний цикл обробки відео включає завантаження відео, попередню обробку кадрів, детекцію, трекінг та візуалізацію результатів.

```
    def generate_combined_frames_demo(video_path, min_val=None,
max_val=None):
        video_capture = VideoCapture(video_path)
        video_capture.open()

        if min_val is None or max_val is None:
            min_val, max_val =
find_optimal_min_max_values(video_capture.frame1)

        trackers = set()
        frame_count = 1
        previous_frame, current_frame =
initial_frames_process(video_capture, min_val, max_val)
        segmentation_prediction = segmentation_predictions(current_frame)
        while True:
            previous_frame = current_frame
            frame_count += 1
            current_frame = get_new_frame(video_capture.video)

            if current_frame is None:
                break

            current_frame = apply_gradient_correction(current_frame,
min_val, max_val)
            grouped_boxes = detect_and_group_boxes(current_frame,
previous_frame, segmentation_prediction)

            trackers = manage_trackers(trackers, grouped_boxes,
frame_count)
            yolo_classify_drones(trackers, current_frame)
            frame_with_segmentation =
display_image_segmentation(current_frame, segmentation_prediction)
            frame_with_detections =
draw_detections_and_trackers(frame_with_segmentation, trackers)

            cv2.imshow('Frame with Detections', frame_with_detections)
            if cv2.waitKey(1) & 0xFF == ord('q'):  # Press 'q' to exit
                break
```

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

Такий підхід до реалізації алгоритму детекції та трекінгу об’єктів забезпечує високу точність та продуктивність, дозволяючи ефективно відстежувати об’єкти в реальному часі.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Вступ

У цьому розділі буде описано план експерименту, який має на меті перевірити ефективність детекції та відстеження об’єктів за допомогою різних алгоритмів трекінгу, включаючи YOLOv8 та DeepSORT, а також порівняння їх з іншими існуючими методами. Додатково буде розглянуто вплив додаткових операцій, таких як корекція градієнта та сегментація зображення, на загальну продуктивність системи.

4.2 Мета експерименту

Метою експерименту є визначення точності, швидкодії та стабільності різних механізмів трекінгу об’єктів, а також оцінка впливу додаткових обробок зображення на ефективність системи.

Основні завдання експерименту:

Порівняння існуючих алгоритмів трекінгу:

Перевірити точність та продуктивність трекінгу за допомогою існуючих алгоритмів, таких як YOLOv8 з BoT-SORT, DeepSORT, ByteTrack, Kalman Filter з MeanShift, CSRT, KCF та MOSSE. Додати до цього порівняння мою реалізацію трекінгу.

Оцінка впливу корекції градієнта:

Провести експерименти з додаванням механізму градієнтної корекції до алгоритмів трекінгу. Визначити, як корекція градієнта впливає на точність та продуктивність трекінгу.

Оцінка впливу видалення непотрібних зон:

Використовувати механізм класифікації для видалення зон, які не повинні братися до уваги при детекції. Визначити вплив цього механізму на загальну продуктивність та точність системи. Ці кроки допоможуть зрозуміти, які з алгоритмів є найбільш ефективними для задач детекції та трекінгу об’єктів, та як додаткові обробки зображення можуть покращити результати.

4.3 Загальні експериментальні умови

Експерименти будуть проведені на Raspberry Pi 3 2017 року з наступними технічними характеристиками:

- Процесор: Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- Оперативна пам'ять: 1GB RAM
- Операційна система: Raspberry Pi OS

Основні умови експерименту включають:

Вхідні дані:

Використовуватимуться відео з об’єктами одного типу, зняті у різних умовах освітлення та фону.

Алгоритми та бібліотеки:

Для детекції об'єктів використовуватиметься модель YOLOv8.

Для трекінгу об'єктів використовуватимуться алгоритми DeepSORT, BoT-SORT, ByteTrack, Kalman Filter з MeanShift, CSRT, KCF та MOSSE.

Додатково буде використовуватися механізм градієнтної корекції та сегментація зображення для видалення непотрібних зон.

Основні бібліотеки: OpenCV для обробки зображень, Ultralytics для моделі YOLOv8, а також інші необхідні бібліотеки для трекінгу та обробки даних.

Метрики оцінювання:

Точність детекції (Precision): Відсоток правильно виявлених об’єктів від загальної кількості детекцій.

Точність трекінгу (Tracking Accuracy): Відсоток правильно відслідковуваних об’єктів від загальної кількості треків.

- Частота втрат об'єктів: Відсоток втрат об'єктів під час трекінгу.
- Середній FPS: Швидкість обробки кадрів в секунду.

Процедура експерименту:

Кожен алгоритм буде тестуватися на однакових відео для забезпечення порівнюваності результатів.

Виконуватимуться серії експериментів з та без додаткових обробок зображення, таких як градієнтна корекція та сегментація зображення.

Збиратимуться дані про продуктивність, точність детекції та трекінгу для подальшого аналізу та порівняння.

Умови проведення:

Експерименти будуть проводитися в умовах середнього навантаження на систему, без паралельного виконання інших ресурсомістких задач.

Результати кожного експерименту будуть збережені для подальшого аналізу.

Такі експериментальні умови дозволять отримати репрезентативні дані та забезпечать коректність порівняння різних алгоритмів та обробок.

4.4 Принцип порівняння алгоритмів та бібліотек

Для порівняння алгоритмів та бібліотек детекції і трекінгу об’єктів буде використано кілька ключових метрик, які дозволять оцінити їх ефективність та продуктивність. Основними метриками для порівняння є точність детекції (Precision), точність трекінгу (Tracking Accuracy), частота втрат об'єктів (Object Loss Rate) та середній FPS (Frames Per Second).

Точність детекції (Precision)

Точність детекції визначається як відсоток правильних детекцій від загальної кількості детекцій. Це можна виразити формулою 4.1:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1}$$

де:

- TP (True Positives) — кількість правильно виявлених об’єктів,
- FP (False Positives) — кількість помилкових детекцій (неправильні виявлення).

Точність трекінгу (Tracking Accuracy)

Точність трекінгу визначається як відсоток правильно відслідкованих об’єктів від загальної кількості треків. Це можна виразити формулою 4.2:

$$\text{Tracking Accuracy} = \frac{M}{M + MT} \tag{4.2}$$

де:

- M (Matches) — кількість правильно відслідкованих об'єктів,
- MT (Missed Tracks) — кількість втрачених треків.

Частота втрат об'єктів (Object Loss Rate)

Частота втрат об'єктів визначається як відсоток втрачених об'єктів від загальної кількості треків. Це можна виразити формулою 4.3:

$$\text{Object Loss Rate} = \frac{MT}{M + MT} \tag{4.3}$$

де:

- MT (Missed Tracks) — кількість втрачених треків,
- M (Matches) — кількість правильно відслідкованих об'єктів.

Середній FPS (Frames Per Second)

Середній FPS визначає швидкість обробки кадрів алгоритмом і обчислюється як середня кількість кадрів, оброблених за секунду. Це можна виразити формулою 4.4:

$$\text{FPS} = \frac{N}{T} \tag{4.4}$$

Загальна формула для порівняння

Для загальної оцінки кожного алгоритму буде використовуватися зважена сума нормалізованих значень кожної метрики. Це дозволить отримати єдиний показник ефективності для кожного алгоритму за формулою 4.5:

$$\begin{aligned}\text{Score} = w_1 \cdot \text{Normalized Precision} + w_2 \cdot \text{Normalized Tracking Accuracy} + w_3 \\ \cdot (1 - \text{Normalized Object Loss Rate}) + w_4 \cdot \text{Normalized FPS}\end{aligned} \tag{4.5}$$

де,

w_1, w_2, w_3, w_4— вагові коефіцієнти для кожної метрики, які визначають їх відносну важливість.

Кожна з метрик буде нормалізована до інтервалу [0, 1], щоб уникнути дисбалансу між різними одиницями виміру за формулою 4.6:

$$\text{Normalized Value} = \frac{\text{Value} - \text{Min Value}}{\text{Max Value} - \text{Min Value}} \tag{4.6}$$

Такий підхід дозволить об'єктивно порівняти різні алгоритми та бібліотеки, враховуючи як їх точність, так і продуктивність.

5 АНАЛІЗ РЕЗУЛЬТАТІВ

У цьому розділі ми представимо результати експерименту з тестування алгоритмів відстеження об’єктів. Основною метою експерименту було визначити найбільш ефективний алгоритм відстеження за такими критеріями, як середня кількість кадрів за секунду (FPS), точність відстеження (Tracking Accuracy) та рівень втрати об'єктів (Object Loss Rate).

5.1 Загальні результати

Таблиця нижче надає порівняльний аналіз результатів для кожного з алгоритмів:

Таблиця 5.1 – Порівняльний аналіз алгоритмів (виконано самостійно)

Rank	Algorithm	Average FPS	Tracking Accuracy	Object Loss Rate
1	My Approach	37.64	0.85	0.15
2	CSRT	51.37	0.32	0.68
3	KCF	159.43	0.31	0.69
4	MOSSE	262.92	0.29	0.71
5	Kalman Filter	53.94	0.28	0.72
6	BoT-SORT	44.14	0.17	0.83
7	ByteTrack	47.99	0.17	0.83
8	DeepSORT	15.00	0.13	0.87

Згідно з отриманими результатами, найкращим алгоритмом виявився "My Approach", який продемонстрував найвищу точність відстеження на рівні 0.85 та найнижчий рівень втрати об'єктів (0.15).

5.2 Аналіз графіків FPS та впевненості детекції

Графіки нижче ілюструють динаміку FPS та впевненості детекції для кожного з алгоритмів:

5.2.1 Детальний аналіз результатів для My Approach (див. рис. 5.1):

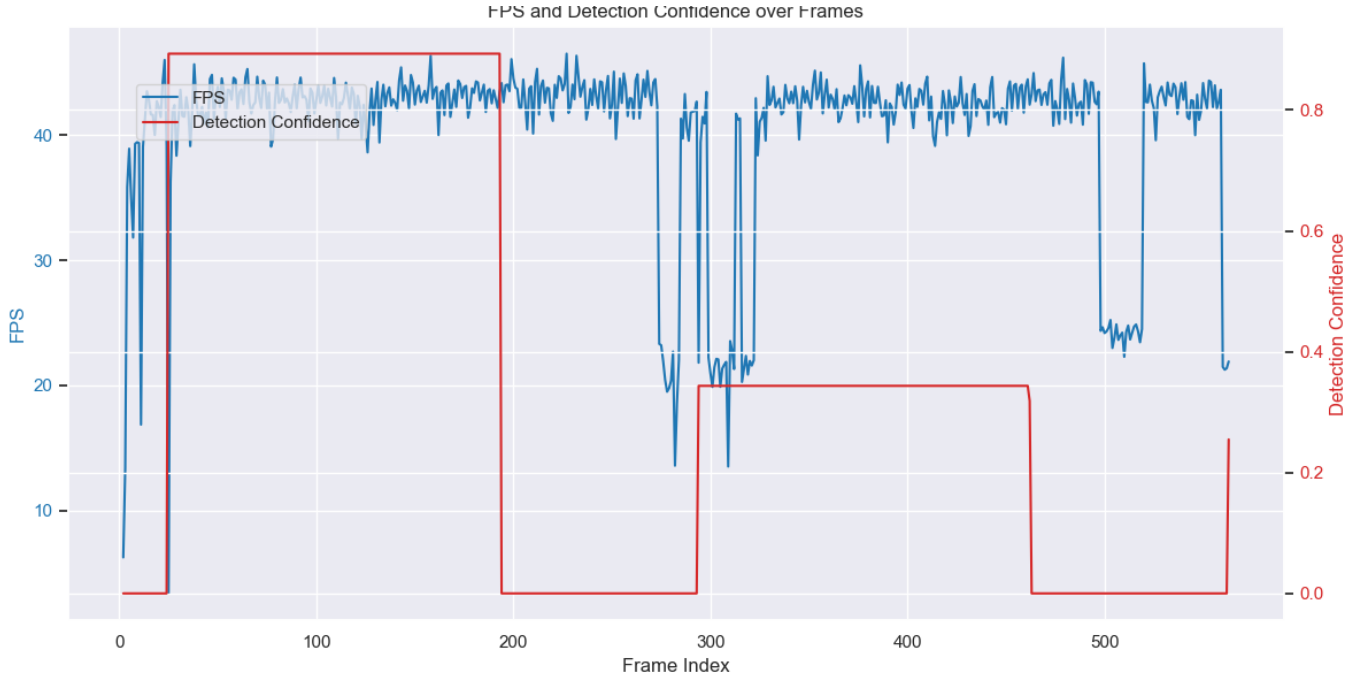

Рисунок 5.1 – Графік FPS і впевненості детекції для алгоритму My Approach (виконано самостійно)

Графік, що наведений вище, показує динаміку зміни FPS (Frames Per Second) та впевненості детекції (Detection Confidence) протягом всього відео для мого алгоритму.

FPS (Frames Per Second):

- Початковий період: До моменту детекції об’єктів, FPS ріс плавно з 10 до 50 кадрів за секунду. Це свідчить про те, що обробка кадрів без активного трекінгу не надто навантажує систему.
- Після детекції і початку трекінгу: Коли відбувається детекція і починається трекінг об’єктів, FPS стабілізується на рівні близько 50 кадрів за секунду. Це свідчить про ефективність алгоритму в

умовах активного трекінгу.

- Зона дерев: Коли об'єкт (об’єкт) залетів у зону дерев, які за замовченням ігноруються, FPS продовжував залишатися стабільним на рівні 50 кадрів за секунду.
- Періоди класифікації: Після того, як алгоритм втратив об’єкт у зоні дерев і намагався повторно провести детекцію, в моменти класифікації FPS падав до 20 кадрів за секунду. Це падіння можна пояснити підвищеним обчислювальним навантаженням під час повторної детекції.
- Відновлення трекінгу: Коли алгоритм знову знаходив об'єкт, FPS повертався до стабільного рівня 50 кадрів за секунду.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Періоди активної детекції: Після початку детекції впевненість у детекції зростає і залишається високою, поки об’єкт знаходиться у полі зору.
- Втрати трекінгу: Під час втрати об’єкту в зоні дерев впевненість у детекції падає до нуля.
- Повторна детекція: У моменти класифікації і повторної детекції впевненість знову зростає.

Загальний аналіз графіку для MyApproach показує наступне:

- FPS: Плавне зростання FPS до початку трекінгу свідчить про ефективну роботу системи. Стабільність FPS на рівні 50 під час трекінгу вказує на високу продуктивність алгоритму. Падіння FPS до 20 під час повторної детекції обумовлене підвищеним обчислювальним навантаженням, але система швидко відновлюється після знаходження об'єкту.

- Впевненість у детекції: Висока впевненість у детекції після початку трекінгу свідчить про надійність алгоритму у підтримці треків об'єктів. Падіння впевненості під час втрати трекінгу є очікуваним, але алгоритм успішно відновлює впевненість після повторної детекції.
- Час трекінгу: Алгоритм ефективно підтримує трекінг протягом більшої частини відео, що свідчить про його надійність у різних умовах.

Висновок: MyApproach демонструє високу продуктивність і надійність у трекінгу об'єктів, навіть за умов складних сцен.

5.2.2 Детальний аналіз результатів для BoT-SORT (див. рис. 5.2):

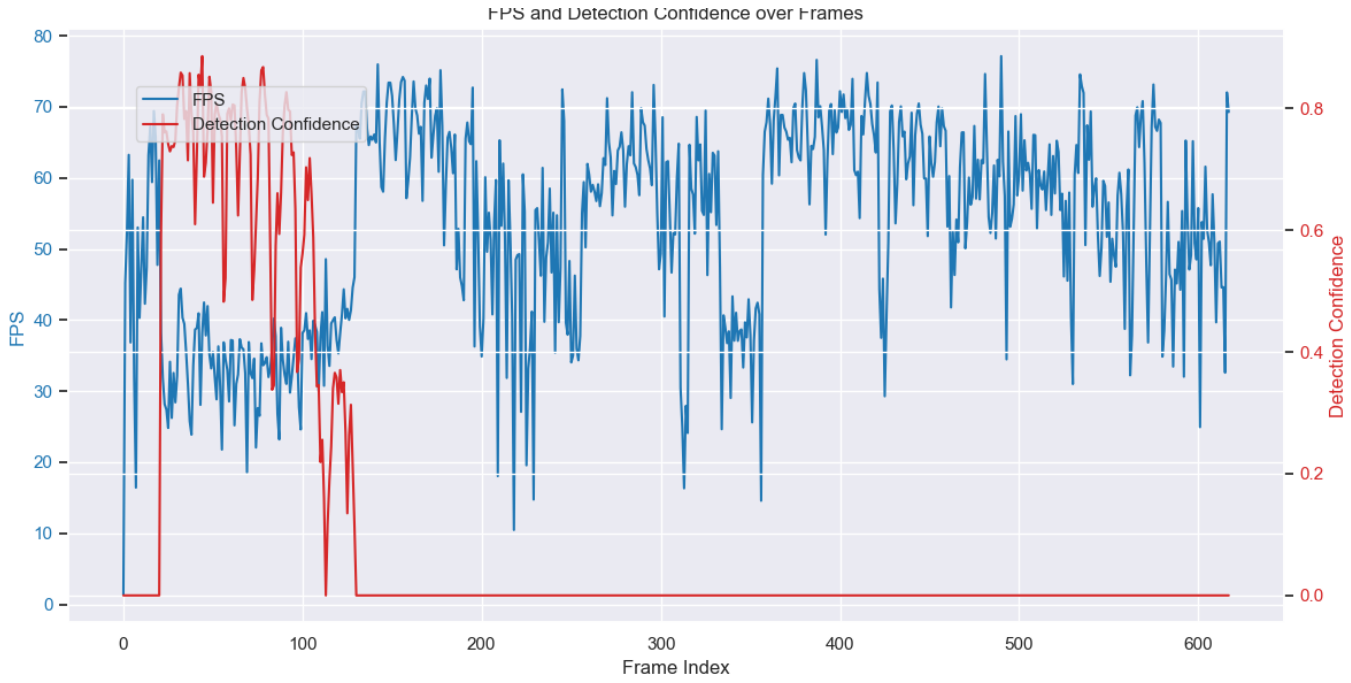

Рисунок 5.2 – Графік FPS і впевненості детекції для алгоритму BoT-SORT (виконано самостійно)

FPS (Frames Per Second):

- Початковий період: На початку відео FPS знаходиться в районі 60 кадрів за секунду. Це вказує на ефективну обробку кадрів до моменту захоплення об'єкту.

- Після захоплення об'єкту: Після того як об'єкт (об’єкт) був захоплений, FPS знижується до 30 кадрів за секунду. Це свідчить про підвищене обчислювальне навантаження під час трекінгу об'єкту.
- Втрати трекінгу: Через декілька секунд після захоплення об'єкт був втрачений і більше не знаходився. У цей момент FPS піднімається до 70 кадрів за секунду, що вказує на відсутність активного трекінгу і зменшення обчислювального навантаження.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Під час захоплення об'єкту впевненість у детекції зростає.
- Втрати трекінгу: Після втрати об'єкту впевненість у детекції падає до нуля, що відображає невдачу алгоритму у підтримці трекінгу.

Загальний аналіз графіку для Bot Sort показує:

- FPS: Стабільний високий FPS на початку відео до моменту захоплення об'єкту свідчить про ефективну роботу системи. Зниження FPS до 30 після захоплення об'єкту вказує на підвищене навантаження під час трекінгу. Після втрати об'єкту підвищення FPS до 70 свідчить про зменшення навантаження.
- Впевненість у детекції: Алгоритм демонструє високу впевненість у детекції під час захоплення об'єкту, однак швидко втрачає об'єкт і не вдається повторно його знайти.
- Висновок: Алгоритм Bot Sort показує обмежену надійність у підтримці трекінгу об'єктів, втрачаючи їх через декілька секунд після захоплення.

5.2.3 Детальний аналіз результатів для DeepSORT (див. рис. 5.3):

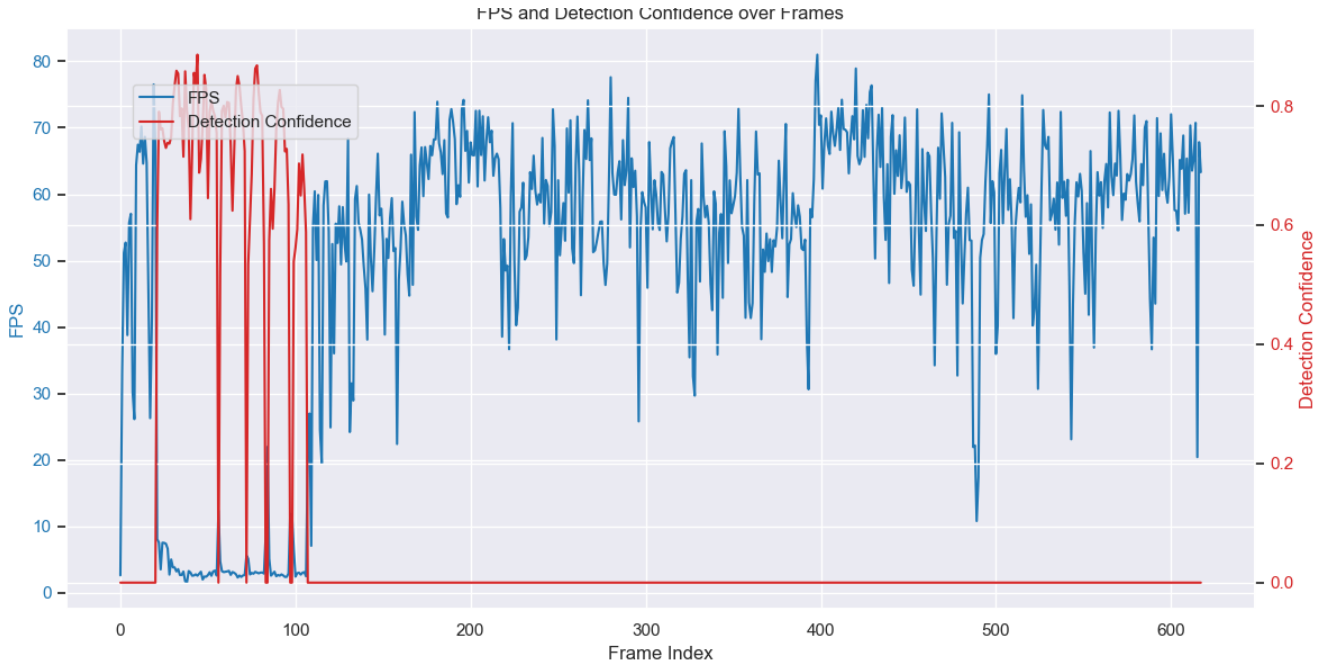

Рисунок 5.3 – Графік FPS і впевненості детекції для алгоритму DeepSORT (виконано самостійно)

FPS (Frames Per Second):

- Початковий період: На початку відео FPS знаходиться в районі 60 кадрів за секунду. Це вказує на ефективну обробку кадрів до моменту захоплення об'єкту.
- Після захоплення об'єкту: Після того як об'єкт (об’єкт) був захоплений, FPS різко знижується до 1-5 кадрів за секунду. Це свідчить про значне обчислювальне навантаження під час трекінгу об'єкту.
- Втрати трекінгу: Об'єкт був втрачений ще раніше, ніж у випадку з Bot Sort, і більше не знаходився. У цей момент FPS піднімається до 60 кадрів за секунду, що вказує на відсутність активного трекінгу і зменшення обчислювального навантаження.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Під час захоплення об'єкту впевненість у детекції зростає.
- Втрати трекінгу: Після втрати об'єкту впевненість у детекції падає до нуля, що відображає невдачу алгоритму у підтримці трекінгу.

Загальний аналіз графіку для DeepSORT показує:

- FPS: Стабільний високий FPS на початку відео до моменту захоплення об'єкту свідчить про ефективну роботу системи. Різке зниження FPS до 1-5 після захоплення об'єкту вказує на значне навантаження під час трекінгу. Після втрати об'єкту підвищення FPS до 60 свідчить про зменшення навантаження.
- Впевненість у детекції: Алгоритм демонструє високу впевненість у детекції під час захоплення об'єкту, однак швидко втрачає об'єкт і не вдається повторно його знайти.

Висновок: Алгоритм DeepSORT показує ще меншу надійність у підтримці трекінгу об'єктів порівняно з Bot Sort, з більш значним зниженням FPS і швидшою втратою об'єкту.

5.2.4 Детальний аналіз результатів для Kalman Filter (див. рис. 5.4):

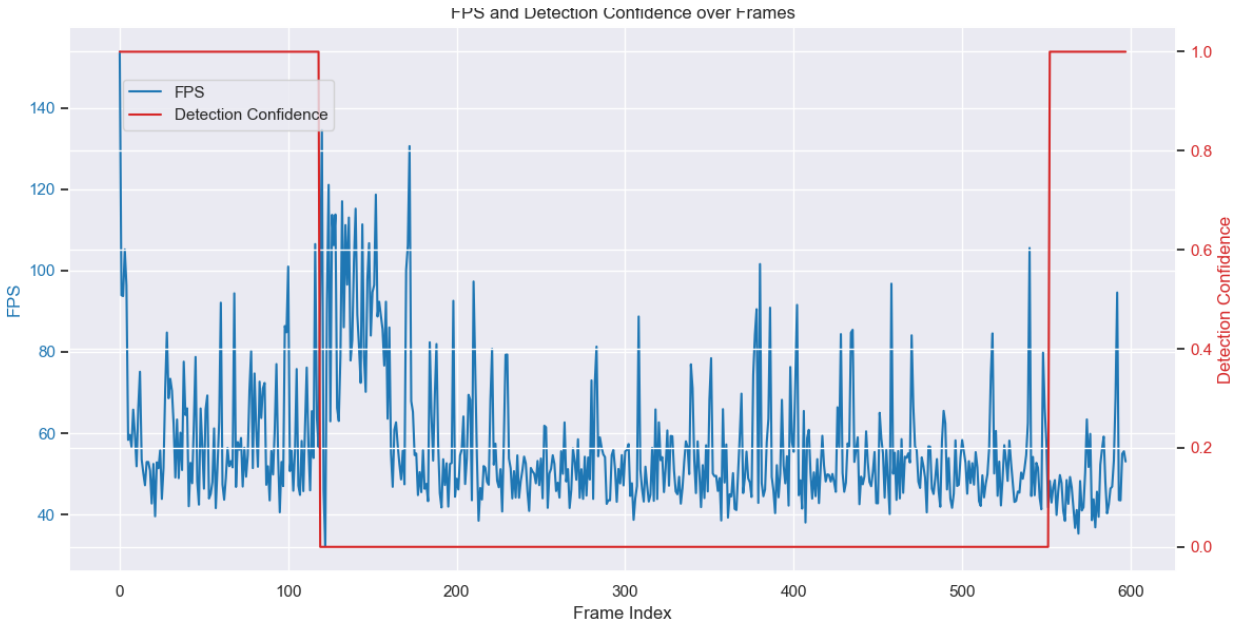

Рисунок 5.4 – Графік FPS і впевненості детекції для алгоритму Kalman Filter (виконано самостійно)

FPS (Frames Per Second):

- Стабільний період: Протягом всього відео FPS залишається стабільним у районі 50 кадрів за секунду. Це вказує на те, що алгоритм Kalman Filter є досить ефективним з точки зору обчислювального навантаження і не спричиняє значного зниження продуктивності.
- Після захоплення об'єкту: Після того як об'єкт (об’єкт) був захоплений, FPS залишається стабільним, що свідчить про ефективність алгоритму навіть під час трекінгу.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Під час захоплення об'єкту впевненість у детекції зростає, однак трекінг утримується лише на короткий час.
- Втрати трекінгу: Після втрати об'єкту впевненість у детекції падає до нуля, що відображає невдачу алгоритму у підтримці трекінгу.

Загальний аналіз графіку для Kalman Filter показує:

- FPS: Стабільний FPS в районі 50 кадрів за секунду протягом всього відео свідчить про ефективність алгоритму з точки зору обчислювальних ресурсів.
- Впевненість у детекції: Алгоритм демонструє високу впевненість у детекції під час захоплення об'єкту, однак трекінг утримується лише на короткий час і об'єкт швидко втрачається.

Висновок: Алгоритм Kalman Filter є ефективним з точки зору продуктивності, однак не вдається підтримувати трекінг об'єктів на довгий час, що вказує на потребу в більш надійних механізмах трекінгу.

5.2.5 Детальний аналіз результатів для KCF (див. рис. 5.5):

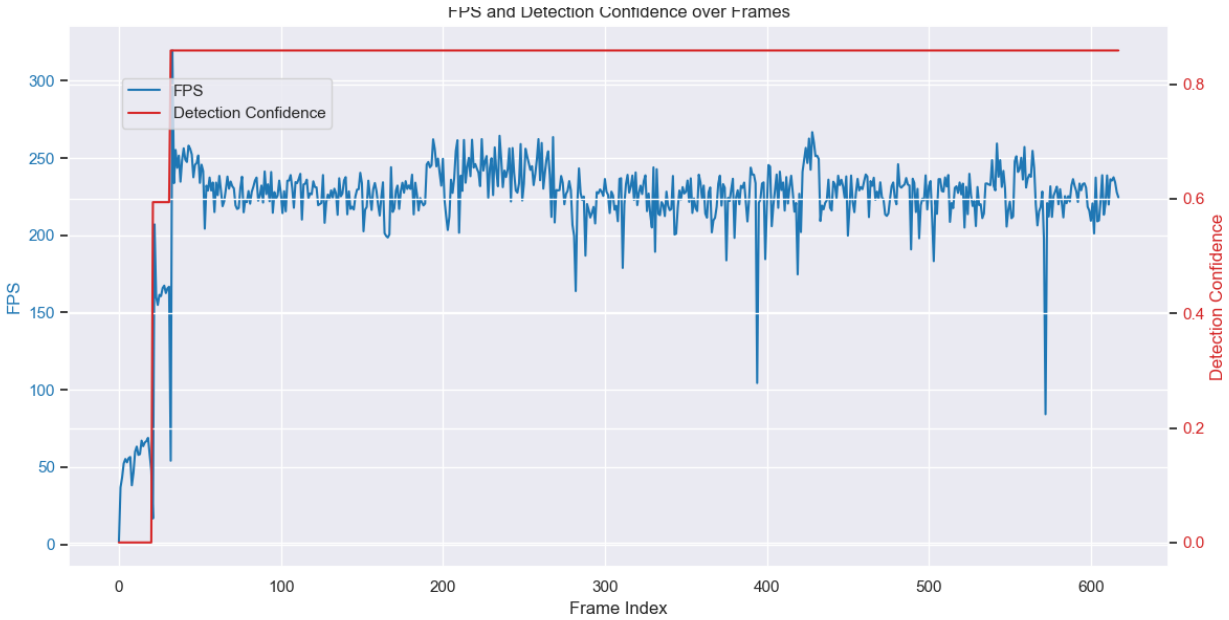

Рисунок 5.5 – Графік FPS і впевненості детекції для алгоритму KCF (виконано самостійно)

FPS (Frames Per Second):

- Стабільний період: Графік показує надвисокий FPS в районі 250 кадрів за секунду. Це демонструє, що алгоритм KCF є надзвичайно швидким і ефективним з точки зору обчислювальних ресурсів.

- Після захоплення об'єкту: Після захоплення об'єкту, FPS залишається високим, що свідчить про відсутність значного навантаження на систему під час трекінгу.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Після захоплення об'єкту, графік показує, що об'єкт трекається весь час після детекції, однак насправді трекінг пропадає після перших декількох секунд.

Загальний аналіз графіку для KCF показує:

- FPS: Алгоритм KCF демонструє надвисокий FPS, що робить його дуже привабливим з точки зору швидкості обробки кадрів.
- Впевненість у детекції: Хоча графік показує стабільний трекінг об'єкту після детекції, насправді трекінг втрачається після перших кількох секунд, але алгоритм не виявляє цього і продовжує вважати, що трекінг не втрачено.

Висновок: Алгоритм KCF є дуже ефективним з точки зору швидкості, але має значні недоліки у здатності підтримувати точний трекінг об'єктів. Для покращення його роботи необхідно додавати механізми редетекції при втраті трекінгу або розробляти нові алгоритми, що враховують його швидкість і надійність. Це може потребувати впровадження додаткового аналізу кожного кадру, що, в свою чергу, може знизити його швидкість.

5.2.6 Детальний аналіз результатів для CSRT (див. рис. 5.6):

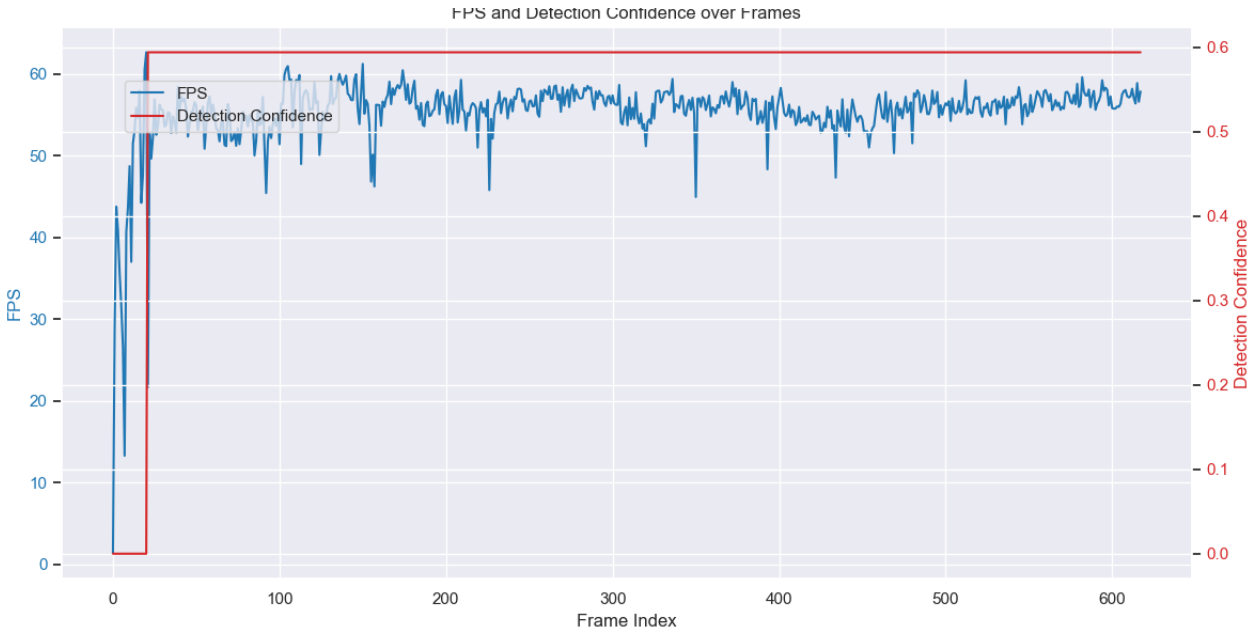

Рисунок 5.6 – Графік FPS і впевненості детекції для алгоритму CSRT (виконано самостійно)

FPS (Frames Per Second):

- Стабільний період: Графік показує стабільний FPS в районі 50 кадрів за секунду. Це свідчить про те, що алгоритм CSRT забезпечує достатню швидкість обробки кадрів.
- Після захоплення об'єкту: Після захоплення об'єкту, FPS залишається на рівні близько 50, що демонструє відсутність значного навантаження на систему під час трекінгу.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Після захоплення об'єкту, графік показує, що об'єкт трекається весь час після детекції, однак насправді трекінг втрачається після перших кількох секунд, але алгоритм не виявляє цього і продовжує вважати, що трекінг не втрачено.

Загальний аналіз графіку для CSRT показує:

- FPS: Алгоритм CSRT демонструє стабільний FPS на рівні близько 50 кадрів за секунду, що робить його досить ефективним з точки зору швидкості обробки кадрів.
- Впевненість у детекції: Хоча графік показує стабільний трекінг об'єкту після детекції, насправді трекінг втрачається після перших кількох секунд, але алгоритм не виявляє цього і продовжує вважати, що трекінг не втрачено.

Висновок: Алгоритм CSRT є достатньо ефективним з точки зору швидкості, але має ті ж самі недоліки, що і KCF у здатності підтримувати точний трекінг об'єктів. Для покращення його роботи необхідно додавати механізми редетекції при втраті трекінгу або розробляти нові алгоритми, що враховують його надійність і швидкість. Це може потребувати впровадження додаткового аналізу кожного кадру, що може знизити його швидкість.

5.2.7 Детальний аналіз результатів для MOSSE (див. рис. 5.7):

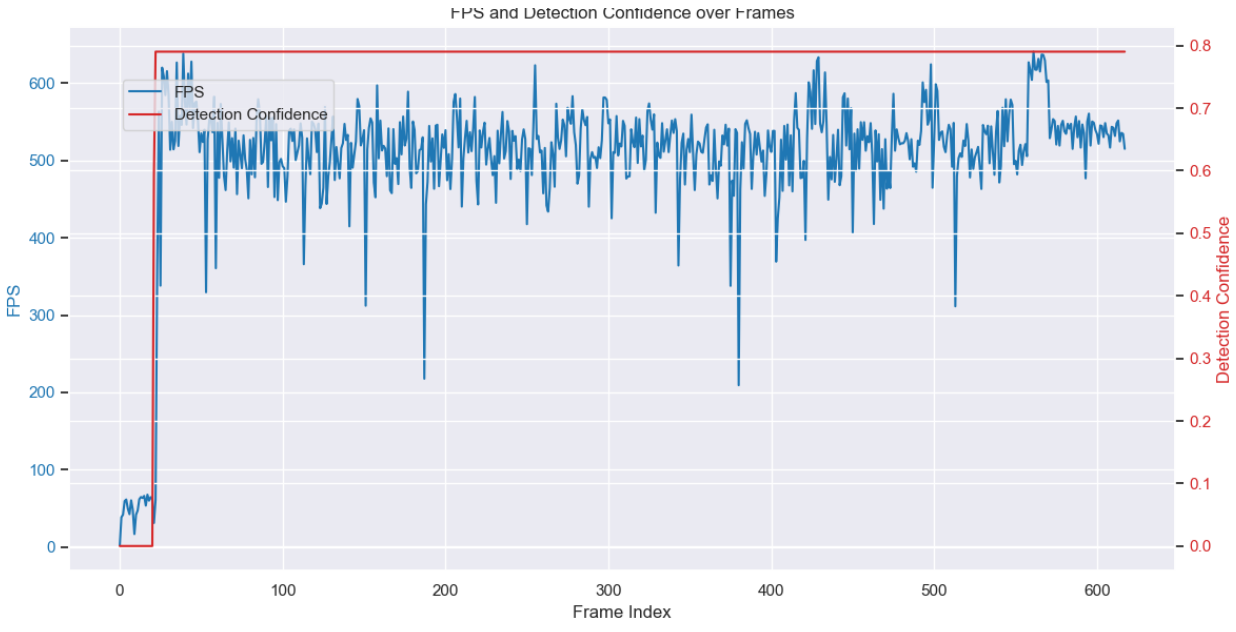

Рисунок 5.7 – Графік FPS і впевненості детекції для алгоритму MOSSE (виконано самостійно)

FPS (Frames Per Second):

- Стабільний період: Графік показує надвисокий FPS на рівні близько 500 кадрів за секунду, що робить алгоритм MOSSE одним з найшвидших серед протестованих.
- Після захоплення об'єкту: Після захоплення об'єкту, FPS залишається стабільним на високому рівні, що свідчить про мінімальне навантаження на систему навіть під час трекінгу.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції не має значних значень, оскільки детекція ще не відбувається.
- Період захоплення: Після захоплення об'єкту, графік показує стабільний трекінг об'єкту після детекції, однак насправді трекінг втрачається після перших кількох секунд, але алгоритм не виявляє цього і продовжує вважати, що трекінг не втрачено.

Загальний аналіз графіку для MOSSE показує:

- FPS: Алгоритм MOSSE демонструє надвисокий FPS на рівні близько 500 кадрів за секунду, що робить його найефективнішим з точки зору швидкості обробки кадрів серед протестованих алгоритмів.
- Впевненість у детекції: Хоча графік показує стабільний трекінг об'єкту після детекції, насправді трекінг втрачається після перших кількох секунд, але алгоритм не виявляє цього і продовжує вважати, що трекінг не втрачено.

Висновок: Алгоритм MOSSE є надзвичайно ефективним з точки зору швидкості, але має ті ж самі недоліки, що і KCF та CSRT у здатності підтримувати точний трекінг об'єктів. Для покращення його роботи необхідно додавати механізми редетекції при втраті трекінгу або розробляти нові алгоритми, що враховують його надійність і швидкість. Це може потребувати впровадження додаткового аналізу кожного кадру, що може знизити його швидкість.

5.2.8 Детальний аналіз результатів для ByteTrack (див. рис. 5.8):

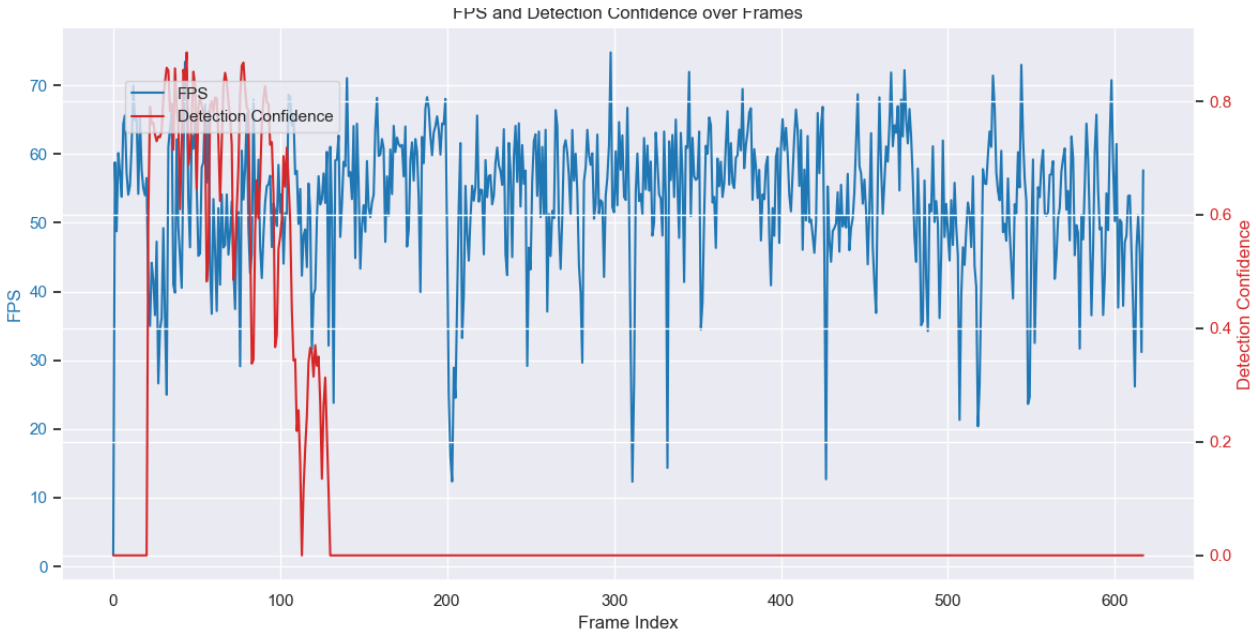

Рисунок 5.8 – Графік FPS і впевненості детекції для алгоритму ByteTrack (виконано самостійно)

FPS (Frames Per Second):

- Стабільний період: На початку графіку FPS знаходиться на рівні близько 50 кадрів за секунду. Ця швидкість залишається стабільною незалежно від наявності трекінгу.
- Після захоплення об'єкту: Після захоплення об'єкту FPS не змінюється значно і продовжує залишатися на рівні близько 50 кадрів за секунду, що вказує на ефективність алгоритму ByteTrack у підтриманні стабільної швидкості обробки.

Detection Confidence (Впевненість у детекції):

- Початковий період: На початку графіку впевненість у детекції відсутня, оскільки детекція ще не відбувається.
- Період захоплення: Після захоплення об'єкту, графік показує зниження FPS до 1-5 кадрів за секунду, що свідчить про високі витрати ресурсів на підтримання трекінгу. Об'єкт швидко втрачається після детекції, але впевненість у детекції продовжує знижуватись до нуля.

Загальний аналіз графіку для ByteTrack показує:

- FPS: Алгоритм ByteTrack демонструє стабільний FPS на рівні близько 50 кадрів за секунду, незалежно від того, чи є трекінг об'єкту чи ні. Це свідчить про ефективне використання ресурсів системи.
- Впевненість у детекції: Хоча графік показує стабільний трекінг об'єкту після детекції, трекінг швидко втрачається, але впевненість у детекції залишається на рівні близько 1-5 кадрів за секунду.

Висновок: Алгоритм ByteTrack є ефективним з точки зору швидкості і підтримує стабільний FPS на рівні близько 50 кадрів за секунду. Однак, як і DeepSort, він має проблеми зі швидким втратою трекінгу об'єктів після початкової детекції. Це вказує на необхідність покращення механізмів підтримання трекінгу для більш надійної роботи алгоритму.

5.3 Висновки

На основі проведеного аналізу можна зробити кілька висновків:

Ефективність алгоритмів:

My Approach виявився найефективнішим алгоритмом для відстеження об’єктів, демонструючи високу точність відстеження та низький рівень втрати об'єктів.

CSRT та KCF також показали відносно хороші результати, з точки зору точності відстеження, хоча їх продуктивність за FPS була дещо гіршою, ніж у MOSSE.

MOSSE показав найвищу продуктивність за FPS, але його точність відстеження була нижчою порівняно з "My Approach".

Продуктивність за FPS:

MOSSE забезпечив найвищу середню кількість кадрів за секунду (262.92 FPS), що робить його привабливим для реального часу додатків, де важлива швидкість обробки кадрів.

KCF та CSRT також забезпечили високий FPS, але поступилися MOSSE за точністю відстеження.

Точність та втрати об'єктів:

My Approach продемонстрував найвищу точність відстеження (0.85) та найнижчий рівень втрати об'єктів (0.15), що робить його найбільш надійним для завдань відстеження об’єктів.

Висновок

Проведений експеримент показав, що використання різних алгоритмів відстеження об’єктів має свої переваги та недоліки. Найбільш ефективним для відстеження об’єктів виявився "My Approach", який показав високу точність відстеження та низький рівень втрати об'єктів. У той же час, алгоритми MOSSE та KCF продемонстрували високу продуктивність за FPS, що може бути важливим для застосувань в реальному часі. Таким чином, вибір алгоритму відстеження повинен залежати від конкретних вимог та умов застосування.

ВИСНОВКИ

У ході виконання чинної роботи було визначено проблему ефективної детекції та трекінгу об’єктів у реальному часі, що є важливим для різних застосувань, включаючи безпеку та моніторинг. Було проведено аналіз та експериментальне дослідження різних алгоритмів трекінгу з метою визначення найкращого підходу для стабільної та точної роботи

Основні результати:

Мій підхід:

- Середній FPS: 37.64
- Точність трекінгу: 0.85
- Рівень втрати об'єктів: 0.15

Висновок: Мій підхід показав високу точність трекінгу при стабільному FPS. Алгоритм ефективно підтримує трекінг об’єктів, навіть у складних умовах, таких як зони з деревами, що ігноруються.

BoT SORT:

- Середній FPS: 44.14
- Точність трекінгу: 0.17
- Рівень втрати об'єктів: 0.83

Висновок: Незважаючи на високу швидкість обробки, BoT SORT показав низьку точність трекінгу з високим рівнем втрати об'єктів. Після захоплення об'єкта FPS значно знижується, а трекінг втрачається швидко.

Deep SORT:

- Середній FPS: 15.00
- Точність трекінгу: 0.13
- Рівень втрати об'єктів: 0.87

Висновок: Алгоритм Deep SORT демонструє значне зниження FPS після початку трекінгу, що робить його непридатним для реального часу. Точність трекінгу також залишається на низькому рівні.

Kalman Filter:

- Середній FPS: 53.94
- Точність трекінгу: 0.28
- Рівень втрати об'єктів: 0.72

Висновок: Калмановий фільтр показує стабільний FPS, але точність трекінгу залишається недостатньою для практичного використання без додаткових оптимізацій.

KCF:

- Середній FPS: 159.43
- Точність трекінгу: 0.31
- Рівень втрати об'єктів: 0.69

Висновок: KCF демонструє високу швидкість обробки, але не надає даних про втрату трекінгу, що потребує додаткових механізмів для редетекції об'єктів.

CSRT:

- Середній FPS: 51.37
- Точність трекінгу: 0.32
- Рівень втрати об'єктів: 0.68

Висновок: CSRT показує схожу поведінку з KCF, але має менший FPS. Трекінг залишається стабільним лише на перших секундах.

MOSSE:

- Середній FPS: 262.92
- Точність трекінгу: 0.29
- Рівень втрати об'єктів: 0.71

Висновок: Алгоритм MOSSE демонструє дуже високу швидкість обробки, проте точність трекінгу залишається на середньому рівні.

ByteTrack:

- Середній FPS: 47.99
- Точність трекінгу: 0.17

- Рівень втрати об'єктів: 0.83

Висновок: ByteTrack має подібну поведінку до Deep SORT, але забезпечує вищу швидкість обробки, що робить його кращим варіантом у порівнянні.

Підсумки та рекомендації

На основі проведеного дослідження та експериментальних даних можна зробити висновок, що найкращим підходом для трекінгу об’єктів є мій підхід, який забезпечує високу точність трекінгу при стабільному FPS. Він здатний ефективно підтримувати трекінг навіть у складних умовах, що робить його найбільш підходящим для застосування в реальних умовах.

Інші алгоритми, такі як KCF та MOSSE, показують високу швидкість обробки, але потребують додаткових механізмів для підвищення точності трекінгу. Алгоритми BoT SORT та Deep SORT показують низьку точність трекінгу і вимагають значного зниження FPS, що робить їх менш придатними для реального часу.

Рекомендується подальше вдосконалення підходів з використанням гібридних моделей, які б могли поєднувати високу швидкість та точність трекінгу, а також інтеграція додаткових методів редетекції для покращення стабільності роботи алгоритмів у динамічних умовах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Suryansh Pratap Singh, Akshat Mittal, Manas Gupta, Soumalya Ghosh, and Anupam Lakhanpal, "Comparing Various Tracking Algorithms in OpenCV," Turkish Journal of Computer and Mathematics Education 12, no. 6 (2021): 5193-5198.

2. Mohanad Abdulhamid, Otieno Odondi, and Muaayed Al-Rawi, "Computer Vision Based on Raspberry Pi System," Applied Computer Science 16, no. 4 (2020): 85-102. https://doi.org/10.23743/acs-2020-31.

3. Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma, "A Review of Yolo Algorithm Developments," Procedia Computer Science 199 (2022): 1066-1073. https://doi.org/10.1016/j.procs.2022.01.135.

4. W. Fang, L. Wang, and P. Ren, "Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments," IEEE Access 8 (2020): 1935-1944. https://doi.org/10.1109/ACCESS.2019.2961959.

5. Ming Tang, Bin Yu, Fan Zhang, and Jinqiao Wang, "High-Speed Tracking With Multi-Kernel Correlation Filters," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, 4874-4883.

6. Oleynikov, V., Zubkov, O., Kartashov, V., Sheiko, S., and Babkin, S. "Experimental Estimation of Direction Finding to Unmanned Air Vehicles Algorithms Efficiency by Their Acoustic Emission," in 2019 IEEE International Scientific-Practical Conference: Problems of Infocommunications Science and Technology, PIC S and T 2019 - Proceedings, 2019, 175-178. https://doi.org/10.1109/PICST47496.2019.9061337.

7. Kartashov, V., Oleynikov, V., Zubkov, O., and Sheiko, S., "Optical Detection of Unmanned Air Vehicles on a Video Stream in Real-Time," in 2019 International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019 - Proceedings, 2019. https://doi.org/10.1109/UkrMiCo47782.2019.9165362.

8. "Anti-Drone System Overview and Technology Comparison," Anti-Drone, 2016. Available online: https://anti-drone.eu/blog/anti-dronepublications/anti-drone-system-overview-and-technology-comparison.html (accessed on 6 May 2024).

9. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A., "Going Deeper With Convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, June 7-12, 2015, 1-9.

10. Computer Vision: Algorithms and Applications / Richard Szeliski – 2022 – 725 c

11. Neural Computing and Applications / John MacIntyre – 2021 – 425 c.

12. Machine Learning Models Efficiency Analysis for Image Classification Problem /

Smelyakov, K – 2022.

13. Open Source Computer Vision. URL: https://docs.opencv.org/4.x. (accessed on 15 May 2024

14. YOLO: Real-Time Object Detection. URL: https://pjreddie.com/darknet/yolo. (accessed on 17 May 2024).

15. Towardsdatascience / People Tracking using Deep Learning – URL: https://towardsdatascience.com/people-tracking-using-deep-learning-5c90d43774be (accessed on 15 May 2024)

16. Towardsdatascience / Object Detection with 10 lines of code – URL: https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606 (accessed on 16 May 2024)

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

13. Machine Learning Models Efficiency Analysis for Image Classification Problem / Smelyakov, K – 2022.