Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра системного програмного забезпечення

Тімков Юрій Юрійович,
студент групи АС-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Вдосконалений метод трекінгу об'єктів для систем відеоспостереження

Спеціальність:
121 – Інженерія програмного забезпечення
Освітня програма:
Інженерія програмного забезпечення

Керівник:
Рувінська Вікторія Михайлівна,
канд. техн. наук, професор

Одеса – 2021

ЗМІСТ

Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Любченко В.В.
«__» __ _____ 2021 р.

ЗАВДАННЯ НА
КВАЛІФІКАЦІЙНУ РОБОТУ

Тімкова Юрія Юрійовича, група АС-161

1. Тема роботи: Вдосконалений метод трекінгу об'єктів для систем відеоспостереження

Керівник роботи: Рувінська Вікторія Михайлівна, канд. техн. наук, професор

затверджені наказом ректора № 374-в від 25 жовтня 2021 р.

2. Зміст роботи:

Аналіз проблеми, формування технічного завдання на розробку, огляд та вибір методів вирішення задачі, проектування, програмна реалізація та тестування програмної системи, випробування програмної системи.

3. Перелік ілюстративного матеріалу:

У відповідності до слайдів електронної презентації.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання: 30 серпня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1.	Аналіз проблеми	10.09.21	вик.
2.	Формування технічного завдання на розробку	16.09.21	вик.
3.	Огляд та вибір методів вирішення задачі	01.10.21	вик.
4.	Проектування програмної системи	12.10.21	вик.
5.	Програмна реалізація системи	20.10.21	вик.
6.	Тестування програмної системи	01.11.21	вик.
7.	Оформлення записки	25.11.21	вик.

Здобувач вищої освіти _____ Ю.Ю. Тімков

Керівник роботи _____ В.М. Рувінська

РЕФЕРАТ

Робота присвячена розробці програмної системи для аналізу даних відеоспостереження у реальному часі.

Метою дослідження є підвищення точності розпізнавання небезпечних ситуацій, пов’язаних з поведінкою людей, у даних відеоспостереження у реальному часі за рахунок удосконаленого методу трекінгу об’єктів.

Розробка базуються на використанні технології виявлення об’єктів YOLO, методу відстеження об’єктів SORT, фреймворку Qt та мови програмування Python.

Як результат роботи створено удосконалений метод трекінгу об’єктів та реалізовано програмну систему для відеоаналітики, що його використовує.

Ключові слова: відеоаналітика, комп’ютерний зір, виявлення об’єктів, трекінг.

ABSTRACT

The work is devoted to the development of a software system for real-time video surveillance data analysis.

The aim of the study is to improve the accuracy of recognizing dangerous situations related to human behavior in real-time video surveillance data through an improved method of tracking objects.

The development is based on the use of YOLO object detection technology, SORT object tracking method, Qt framework and Python programming language.

As a result, an improved method of object tracking was created and a software system for video analytics that uses it was implemented.

ВСТУП

Із розвитком інформаційних технологій зростає і їх вплив на різні сфери життя людини. Одна з таких сфер, що зазнає значних змін під впливом постійного виникнення нових комп'ютерних технологій та засобів – безпека. У галузі, яка завжди була пріоритетною для людей та бізнесу, але потребувала значних людських ресурсів, швидко знайшли використання методи автоматизації – від примітивних апаратних механізмів контролю доступу до складних інформаційних систем з використанням штучного інтелекту [1].

Невід'ємна частина сучасних систем безпеки об'єктів власності – відеоспостереження. Завдяки підвищенню доступності апаратних компонентів навіть побутові користувачі можуть дозволити собі встановлення відеокамер у власних домівках та автомобілях. Це створює значний попит на програмні компоненти для відеоспостереження – для поєднання камер у мережі, управління ними, зберігання відеоданих та іншого [2]. Ці потреби сповна задовольняються ринком програмного забезпечення. Однак розвиток технологій штучного інтелекту, зокрема, комп'ютерного зору, дозволив зробити наступний крок в цифровізації безпеки власності – використання автоматичної відеоаналітики. Відеоаналітика дозволяє мінімізувати потребу у втручанні людини в роботу системи відеоспостереження, вимагаючи дій лише у разі виникнення небезпечних ситуацій, які розпізнаються автоматично [3].

Великі компанії та державні установи вже досить успішно користуються такими засобами. Але виробники рішень з відеоаналітики на ринку інформаційних технологій наразі не фокусуються на користувачах меншого масштабу зі спрощеними системами спостереження з декількох камер. Це зумовлено високими вимогами систем відеоаналітики до обчислювальних ресурсів [4], що не дозволяє їм ефективно працювати на комп'ютерах середньої та низької продуктивності та значно підвищує вартість роботи на хмарних ресурсах. Серед рішень же, що пропонуються як відкрите

програмне забезпечення, наявні оптимізовані під побутові комп’ютери системи, проте вони страждають від високого проценту помилок при розпізнаванні, що у сфері безпеки є неприйнятним.

У сфері безпеки однією з ключових функцій відеоаналітики є розпізнавання подій. Типові приклади подій, що може розпізнати відеоаналітика – наявність об’єктів у певній зоні, їх час знаходження у цій зоні, наявність руху та його швидкість, зміна кількості об’єктів, перетинання обмежувальних ліній. Такі події у застосуванні до реальних ситуацій дозволяють виявити наявність людей або інших об’єктів на забороненій території, їх кількість та час затримки у певній зоні, підозрілу активність, залишені підозрілі предмети тощо. Для детектування подій відеоаналітика використовує методи комп’ютерного зору, зокрема виявлення, розпізнавання та трекінг. Методи виявлення дозволяють визначати наявність об’єктів та їх місцезнаходження, методи розпізнавання – ідентифікувати об’єкти, методи трекінгу – підтримувати інформацію про об’єкти у часі. Саме трекінг є ключовим при детектуванні подій, оскільки без пов’язування інформації про об’єкти між кадрами аналіз їх поведінки, а з ним і розпізнавання більшості подій, були би неможливими.

У сучасних методах відеоаналітики трекінг об’єктів базується на комбінації методів їх виявлення та відстеження, зокрема для виявлення використовують глибинні нейронні мережі, а для відстеження – математичні та алгоритмічні підходи, включаючи фільтр Калмана [5] та угорський алгоритм [6], а також нейронні мережі, що визначають глибинні ознаки для запам’ятовування зовнішнього вигляду об’єктів.

У роботі створюється удосконалений метод трекінгу об’єктів на основі комбінації вищезазначених методів, на базі якого розробляється програмна система для відеоаналітики. Удосконалення методу трекінгу дозволяє вирішити наведену проблему завдяки підвищенню точності розпізнавання при достатніх для використання на комп’ютерах середньої продуктивності вимогах до ресурсів. Тобто, метою роботи є підвищення точності розпізнавання небезпечних ситуацій, пов’язаних

з поведінкою людей, у даних відеоспостереження в реальному часі за рахунок удосконаленого методу трекінгу об’єктів.

Для досягнення мети виконано наступний перелік задач:

- проведено аналіз рішень у сфері відеоаналітики, що існують на ринку або пропонуються як ПЗ з відкритим вихідним кодом;
- сформулювало технічне завдання на розробку програмної системи для розпізнавання небезпечних ситуацій, пов’язаних з поведінкою людей, у даних відеоспостереження;
- проведено аналіз наявних моделей та методів виявлення та трекінгу об’єктів;
- створено покращену версію методу трекінгу об’єктів у реальному часі із підвищеною відносно початкового методу точністю;
- створено алгоритм розпізнавання небезпечних ситуацій на основі даних трекінгу;
- реалізовано програмну систему згідно з технічним завданням;
- проведено випробування та оцінено результати роботи розробленої системи.

У роботі отримано нові наукові результати: удосконалено метод трекінгу об’єктів, який, на відміну від існуючих, базується на комбінації нейронної мережі для виявлення об’єктів, нейронної мережі, що визначає глибинні ознаки для запам’ятовування зовнішнього вигляду об’єктів, фільтру Калмана та угорського алгоритму; для нейронної мережі, що виявляє об’єкти, виконується донавчання на нових даних на кінцевому пристрої, що підвищує точність трекінгу та разом зі швидкодією оригінального методу дозволяє використовувати його для відеоаналітики на комп’ютерах середньої продуктивності.

У першому розділі даної роботи наведено аналіз наявних продуктів у сферах відеоспостереження та відеоаналітики, огляд їх функціоналу та продуктивності.

Другий розділ містить технічне завдання на розробку.

У третьому розділі проводиться аналіз наявних методів виявлення та відстеження об’єктів та розглядається розробка покращеного методу трекінгу об’єктів в реальному часі.

Четвертий розділ містить інформацію щодо вимог до програмної системи, її архітектурне проектування та деталі розробки.

У п’ятому розділі наведено опис використаних у процесі розробки технічних засобів, інструкції для користувача, а також результати тестування системи.

У шостому розділі описано випробування програмної системи та надано оцінку отриманим результатам.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Аналіз існуючих рішень дозволяє оцінити наявну ситуацію на ринку та науковий прогрес у предметній області, що розглядається. У роботі розглянуто існуючі програмні системи управління відеоспостереженням та реалізації відеоаналітики у них, а також окремі модулі відеоаналітики.

1.1 Огляд існуючих продуктів у сфері відеоспостереження

На ринку систем відеоспостереження присутня велика кількість рішень різного масштабу, як універсальних, так і вузькоспеціалізованих. Розглянемо найпопулярніші системи та їх характеристики.

1.1.1 Основні поняття

Відеоспостереження – використання оптико-електронних пристроїв для передачі відеосигналу в певне місце [7].

Система відеоспостереження – це програмно-апаратний комплекс (відеокамери, об’єктиви, монітори, реєстратори та інше устаткування), призначений для організації відеоконтролю як на локальних, так і на територіально розподілених об'єктах [8].

IP-камера (або мережева камера) знімає та передає відео безпосередньо через IP-мережу. Кожна камера має вбудований веб-сервер і свою IP-адресу. IP-камери дозволяють віддалено переглядати, записувати й керувати з будь-якого місця за допомогою веб-сервера або програмного забезпечення для керування відео.

Поле зору – зона покриття, яку забезпечує камера при перегляді на повному кадрі. Поле зору можна визначити за типом камери, об’єктивом і роздільною здатністю зображення.

Відеоспостереження зазвичай використовується для [8]:

- віддаленого моніторингу: для захисту від крадіжок, розкрадання майна;
- охорони об'єктів: для захисту периметра будівель;
- моніторингу операцій: для спостереження за повсякденними операціями і як інструмент для оптимізації операцій;
- попередження вандалізму: видимі камери можуть бути стримуючим фактором для вандалів через можливість того, що їх можна ідентифікувати на відео;
- безпеки працівників: для дотримання правил безпеки, а також для захисту роботодавця в цивільному процесі;
- охорони паркувань: для моніторингу крадіжок або пошкоджень транспортних засобів, а також аварій;
- спостереження за подіями: для контролю натовпу, а також для запобігання злочинам;
- громадської безпеки: для міських вулиць, парків і мікрорайонів для запобігання злочинності та підвищення громадської безпеки;
- моніторингу дорожнього руху: для покращення потоку дорожнього руху, виявлення порушень правил, моніторингу аварій.

1.1.2 Cisco Meraki MV

Система відеоспостереження для підприємств Cisco Meraki MV є одним з наймасштабніших та найпопулярніших продуктів, що покриває більшість потреб великих організацій у відеоспостереженні. Ключовими особливостями системи компанія Cisco позиціонує роботу «з коробки», централізоване керування системою через хмарний інтерфейс та підвищену безпеку передачі відео завдяки шифруванню. Meraki MV використовує нову технологію контролю пропускної здатності мережі, що розміщує відео у сховищі на самій камері, а не в хмарі, і завдяки цьому перерозподіляє потоки даних, гарантуючи, що мережа отримує необхідну пропускну здатність у критичні моменти. Такий підхід також дозволяє камерам працювати навіть за

відсутності з’єднання з мережею [9]. Перевагою системи Meraki є можливості інтеграції з багатьма рішеннями для створення комплексної системи безпеки [10].

Щодо методів аналітики - система Meraki MV може виявляти об’єкти – людей та транспортні засоби, а також створювати теплові карти руху. Cisco приділяє значну увагу конфіденційності, тому функції стеження у системі обмежені.

1.1.3 Rhombus

Як і Meraki, Rhombus - комплексне рішення у галузі безпеки з використанням камер та сенсорів. Програмно-апаратний комплекс Rhombus з’єднує камери одна з одною, а також зберігає та обробляє дані, через хмару. Система поставляється з єдиною приладовою панеллю, з якої можна контролювати в режимі реального часу те, що відбувається у просторі огляду камер. Rhombus, як і Meraki, має перевагу інтеграції з багатьма рішеннями, для створення більш повної інфраструктури безпеки [10]. Що стосується аналітики, Rhombus пропонує сповіщення з відфільтрованими кадрами про події, що виявляються за рухом, змінами освітлення та багатьма іншими факторами. Система підтримує розпізнавання облич та номерних знаків, а також підрахунок кількості осіб у кадрі [11].

Основним недоліком залишається орієнтація на промислових споживачів, що робить затрати на використання комплексу у менших масштабах неприйнятними.

1.1.4 EyeLine Video Management Software

Програмне забезпечення для керування відео EyeLine - варіант програмного забезпечення, яке підтримує багато камер. EyeLine може одночасно контролювати та записувати відео з понад 100 джерел. Програмне забезпечення дозволяє переглядати кадри в режимі реального часу на місці або онлайн з віддаленим входом. Функціонал виявлення руху економить місце у сховищі, зберігаючи кадри лише тоді, коли у зоні видимості відбуваються якісь події. Налаштування дозволяють ввімкнути сповіщення

про виявлення руху на електронну пошту або повідомлення на мобільний пристрій. У програмі були помічені проблеми з підтримкою певних камер, тому для її коректної роботи рекомендується використовувати веб-камери, підключені через USB [12].

1.1.5 ContaCam

Програмне забезпечення ContaCam позиціонує себе як швидке, легке та універсальне, але при тому з достатнім функціоналом, щоб задовольнити будь-які потреби безпеки. Налаштування програми мінімалістичні – користувачі повинні вибирати між виявленням руху чи безперервним записом, ввести ім’я камери та налаштувати тривалість зберігання файлів. Програмне забезпечення пропонує можливість щоденного створення підсумкових відео на базі виявлення руху та містить інтегровану технологію розпізнавання номерних знаків. Відомо, що ContaCam має стабільну продуктивність, але іноді у програмі виникають труднощі під час з’єднання з IP-камерами.

ContaCam – популярне рішення серед побутових споживачів через невимогливість до апаратного забезпечення та простоти функціоналу, але з цих причин воно не містить функціоналу просунутої відеоаналітики. Тим не менш, програма відкрита для розширення користувачами та пропонує інструкції для додаткових інтеграцій існуючих технологій аналітики [13].

1.1.6 iSpy

iSpy — це програма безпеки з відкритим вихідним кодом, яка підключається до IP-камер або веб-камер. Програмне забезпечення підтримує необмежену кількість камер і надає користувачам доступ через веб-інтерфейс на сайті iSpy або на окремих пристроях. Однак у програми існують деякі обмеження сумісності з комп’ютером, які можуть вимагати від користувача завантаження старішої версії програмного забезпечення. Функції аналітики цілком базуються на русі і включають сповіщення

про його виявлення або систематичні відеозвіти та знімки з моментами зміни його інтенсивності. Система також містить декілька функцій, пов’язаних зі звуком – включаючи аналітику даних з мікрофонів [14].

Програмне забезпечення безкоштовне для завантаження та використання, але для доступу до камер через веб-інтерфейс потрібна платна підписка. Через таку політику iSpy стала популярним продуктом, і має понад 2 мільйони користувачів у всьому світі.

1.2 Огляд наявних на ринку підсистем відеоаналітики

Системи відеоаналітики – вузькоспеціалізовані продукти, що зазвичай містять лише програмну складову та, на відміну від систем відеоспостереження, мають набір функцій, сфокусований безпосередньо на автоматизованому аналізі відеоданих.

1.2.1 Основні поняття

Відеоаналітика – технологія, яка обробляє цифровий відеосигнал за допомогою спеціального алгоритму для виконання функції, пов’язаної з безпекою [15]. Відеоаналітіка являє собою програмне забезпечення для роботи з відеоконтентом. В основі програмного забезпечення лежить комплекс алгоритмів машинного зору, що дозволяють вести моніторинг і проводити аналіз даних без прямої участі людини.

Основні функції відеоаналітики:

- виявлення об'єктів;
- спостереження за об'єктами;
- класифікація об'єктів;
- ідентифікація об'єктів;
- розпізнавання ситуацій;
- прогнозування поведінки об'єкта чи виникнення ситуації;
- інтелектуальне стиснення відеоконтенту з урахуванням інтересу споживача.

Існують наступні типи відеоаналітики [16]:

- Периметральна відеоаналітика. Застосовується для охорони ділянок та периметрів, виявляє вторгнення або перетин сигнальної лінії.
- Ситуаційна відеоаналітика. Застосовується для розпізнавання ситуацій, пов'язаних із поведінкою людей або з рухом транспортних засобів. Ситуаційна відеоаналітика може працювати на основі правил, заданих користувачем, або на основі накопиченої інформації.
- Бізнес-аналітика застосовується для оцінки продуктивності персоналу, оптимізації бізнес-процесів та досліджень поведінки клієнтів.
- Біометрична відеоаналітика застосовується для ідентифікації осіб за біометричними ознаками.
- Номерна відеоаналітика застосовується для розпізнавання реєстраційних знаків автомобілів, а також для аналізу їх руху за даними камер.
- Багатокамерна відеоаналітика застосовується для стеження за об'єктами за допомогою декількох камер. Результатом роботи багатокамерної відеоаналітики є траєкторія руху об'єкта на території спостереження.
- Технологічна відеоаналітика застосовується для моніторингу процесів, забезпечення якості виробництва, підвищення продуктивності.
- Тамперинг-сигналізація реалізує безперервний моніторинг працездатності обладнання з метою виявлення технічних несправностей, а також фактів несанкціонованого втручання у систему відеоспостереження.

1.2.2 Verizon Intelligent Video

Verizon Intelligent Video [17] – комплексне рішення для віддаленого моніторингу, яке забезпечує аналітику для оптимізації перегляду відео та забезпечення ситуаційної обізнаності на основі сповіщень. Продукт має хмарну систему керування, обробки та зберігання, та поставляється як із власним апаратним забезпеченням, так і окремим програмним сервісом. Verizon надає розширену аналітику відео, яка виконується

«майже в реальному часі». Заявлені функції підсистеми відеоаналітики надають можливість контролювати робочий персонал і місця, зони з інтенсивним рухом, офіси, парки, медичні центри, комунальні підприємства, будівельні майданчики. Система аналітики дозволяє переглядати години відео за хвилини шляхом формування звітів, що може значно підвищити продуктивність обробки даних. Основна спрямованість функцій аналітики продукту – збільшення швидкості розслідувань та автоматичне архівування доказів у випадку виникнення небажаної ситуації.

1.2.3 BriefCam

Набір продуктів, що розроблює компанія BriefCam, спрямований на швидкий перегляд та аналіз матеріалів з камер спостереження, що дозволяє швидко аналізувати записи шляхом створення стислого звіту про події, які алгоритми визначили як значущі. Але на відміну від аналітичних продуктів, що реалізують автоматизоване відеоспостереження у реальному часі, BriefCam фокусується на видаленні людського фактору при перегляді відео у пошуках значущих ситуацій.

Підсистема відеоаналітики надає функціонал швидкого пошуку та фільтрування об’єктів та подій, може визначати чоловіків, жінок, дітей, транспортні засоби та зміни освітлення, використовуючи 28 класів та атрибутів. Додатково підтримується розпізнавання обличчя, номерних знаків, схожості зовнішнього вигляду, кольору, розміру, вимірювання швидкості, визначення шляху, напрямку і часу перебування, що забезпечує потужний набір пошукових комбінацій [18].

1.2.4 IBM Intelligent Video Analytics

IBM Intelligent Video Analytics [19] надає функції створення відеоаналітики, щоб допомогти організаціям побудувати власні моделі візуального розпізнавання для своєї галузі. Цей продукт включає ширший набір стандартних моделей об’єктів і

класифікації, які можна використовувати для обробки відео в реальному часі або на записах.

Набір функцій відеоаналітики сфокусовано на аналізі подій, пов’язаних з наявністю та поведінкою людей – розпізнавання та відстеження облич, підрахунок числа людей, керування чергами та потоками людей, аналіз щільності натовпу та інше. У підсистемі відеоаналітики також доступні широкі можливості конфігурування сповіщень.

IBM не надає можливості обробляти дані у хмарі, а для налаштування системи та управління нею необхідний кваліфікований аналітик, тому рішення орієнтоване в основному на великих комерційних клієнтів.

1.2.5 Bosch Intelligent Video Analytics

Intelligent Video Analytics від Bosch [20] — це система допомоги охороні, яка позиціонується як рішення для критично важливих ситуацій - розпізнавання вторгнень на великих відстанях та в екстремальних погодних умовах, моніторинг дорожнього руху на перехрестях, автомагістралях або тунелях, високопродуктивний підрахунок людей або інша високопродуктивна відеоаналітика.

Підсистема відеоаналітики обробляє такі складні завдання, як розпізнавання перетину кількох ліній, стеження за маршрутом, виявлення незадіяних і видалених об’єктів, виявлення руху проти потоку, оцінка щільності та підрахунок людей у натовпі. У підсистемі відеоаналітики також доступні налаштування сповіщень та збирання статистичних даних, але, на відміну від продукту IBM, пропонуються різні рівні налаштувань – від спрощеної конфігурації через інтуїтивний графічний інтерфейс до повного контролю за допомогою редактору сценаріїв.

Система також виконує автоматичне калібрування камер, але таке спрощення спричиняє обмеження – список камер, що підтримуються, містить лише кілька найменувань.

1.2.6 Calipsa

Calipsa [21] – набір програмного забезпечення для відеоаналітики, що використовує просунуту технологію зменшення кількості хибних спрацювань тривоги. Відеоаналітика працює повністю на хмарних ресурсах і включає незалежні підсистеми виявлення людей та транспортних засобів, сповіщення про небезпечні ситуації у реальному часі, перевірки стану камер та криміналістичного аналізу відео на записах. Згідно з інформацією від розробників, технологія зменшення кількості хибних спрацювань видаляє 85% хибних сповіщень та доводить точність розпізнавання реальних небезпечних ситуацій до 90%. Функціонал аналізу у реальному часі обмежено підрахунком об’єктів в зонах інтересу.

1.3 Порівняння розглянутих систем відеоспостереження та підсистем відеоаналітики

Наведені системи відеоспостереження та підсистеми відеоаналітики відрізняються за функціональною спрямованістю та містять різні можливості для користувачів, тому доцільно порівняти їх за кількома загальними характеристиками, що є важливими для виявлення основних проблем.

Головними характеристиками для порівняння є особливості апаратної складової систем та підсистем (наявність спеціалізованих апаратних компонентів та використання хмарних обчислювальних ресурсів), робота аналітики у реальному часі та функціональні можливості аналітики, зокрема розпізнавання складних ситуацій (сукупчення людей, тривала затримка у полі зору, перетин обмежувальних ліній, підозріла активність тощо). У порівняння також включено характеристики, що є другорядними, проте впливають на досвід використання та можливості користувачів – такі як наявність спрощеної системи налаштувань аналітики та функціоналу автоматичних сповіщень.

Порівняльна характеристика функціоналу та особливостей розглянутих рішень, що надають фунціонал відеоспостереження та містять відеоаналітику, наведена у

таблиці 1.1. Порівняння окремих модулів та підсистем відеоаналітики надано у таблиці 1.2.

Таблиця 1.1 – Порівняльна характеристика розглянутих систем відеоспостереження

Характеристика	**Cisco Meraki MV**	**Rhombus**	**EyeLine**	**ContaCam**	**iSpy**	**Розроблена програма**
Наявність єдиної програмно-апаратної системи	Так	Так	Ні	Ні	Ні	Ні
Хмарна обробка даних	Так	Так	Частково	Ні	Частково	Ні
Автоматична аналітика у реальному часі	Так	Так	Частково	Частково	Частково	Так
Сповіщення про небезпечні ситуації	Так	Так	Частково	Ні	Так	Так
Можливість запуску на локальних пристроях	Ні	Ні	Так	Так	Так	Так

Таблиця 1.2 – Порівняльна характеристика підсистем відеоаналітики

Характеристика	**Verizon IV**	**BriefCam**	**IBM IVA**	**Bosch IVA**	**Calipsa**	**Розроблена програма**
Хмарна обробка даних	Так	Так	Ні	Ні	Так	Ні
Аналітика у реальному часі	Частково	Ні	Так	Так	Частково	Так
Спрощене налаштування	Частково	Ні	Ні	Так	Ні	Так

Продовження таблиці 1.2

Характеристика	**Verizon IV**	**BriefCam**	**IBM IVA**	**Bosch IVA**	**Calipsa**	**Розроблена програма**
Розпізнавання складних ситуацій	Ні	Частково	Так	Ні	Ні	Частково
Можливість запуску на неспеціалізованих пристроях	Ні	Так	Ні	Ні	Так	Так

Як видно з результатів порівняння, всі розглянуті рішення можна умовно поділити на дві категорії: корпоративні системи з хмарною обробкою даних та програми для широкого кола споживачів зі спрощеною відеоаналітикою. Жодна з цих категорій не задовольняє потреб користувачів у високому рівні автоматизації аналізу даних відеоспостереження за невелику вартість: перша вимагає значних затрат на апаратну складову та хмарні обчислювальні потужності, друга – не надає достатнього функціоналу для надійного розпізнавання можливих небезпек та сповіщення про них.

Для задовільнення таких потреб необхідна програмна система, що зможе розпізнавати складні ситуації у реальному часі на локальних пристроях користувачів. Існуючі системи не відповідають цим вимогам через високі затрати обчислювальних ресурсів на найбільш важливий етап обробки відеоданих – трекінг. Тобто системи, що забезпечують достатню точність трекінгу для розпізнавання складних ситуацій, не придатні для роботи у реальному часі на неспеціалізованих комп’ютерах користувачів.

Отже актуальним є створення програмної системи для аналізу даних відеоспостереження, що використовуватиме удосконалений метод трекінгу об’єктів для розпізнавання небезпечних ситуацій у реальному часі і сповіщення про них.

1.4 Висновки до розділу

У даному розділі було розглянуто декілька існуючих програмних та програмно-апаратних систем та модулів, що надають функціонал відеоспостереження та відеоаналітики. У результаті порівняння виявлено основні тенденції на ринку систем відеоспостереження та підсистем відеоаналітики та негативний вплив, який ці тенденції спричиняють на досвід користування певної категорії споживачів. Такий аналіз дозволив провести обґрунтування актуальності даної роботи.

2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ

2.1 Мета проведення роботи

Підставою розробки є завдання на дипломне проектування. Тема роботи – «Вдосконалений метод трекінгу об'єктів для систем відеоспостереження». Мета роботи – підвищення точності розпізнавання небезпечних ситуацій, пов’язаних з поведінкою людей, у даних відеоспостереження у реальному часі за рахунок удосконаленого методу трекінгу об’єктів. Пропонується розробка програмної системи для аналізу даних відеоспостереження, що розпізнаватиме небезпечні ситуації, пов’язані з поведінкою людей, та сповіщатиме про них у реальному часі.

2.2 Призначення розробки

Задачею розробки є створення програмної системи, що отримує потік даних з під’єднаних відеокамер, аналізує його та розпізнає небезпечні ситуації, пов’язані з поведінкою людей, сповіщаючи про них користувача.

Система поставлятиметься як програма для настільних та портативних комп’ютерів під керуванням операційної системи Windows. При запуску програма дає користувачу можливість обрати під’єднані до пристрою камери, налаштувати параметри сповіщення та переглядати результати аналізу відео як у реальному часі, так і через вбудований журнал подій. При виникненні небезпечних ситуацій система надсилатиме користувачу повідомлення згідно налаштуванням сповіщення.

Небезпечні ситуації визначаються з фіксованого набору, що може коригуватися користувачем. Параметри розпізнавання ситуацій включають числові значення, що обмежують безпечні показники поведінки об’єктів (їх кількість, швидкість руху, обмежувальні лінії та зони, час знаходження у зоні, кількість повторних появ), та перемикачі, що вимикають розпізнавання окремих ситуацій (наприклад, наявність залишених предметів).

Для автоматизації процесу розпізнавання небезпечних ситуацій необхідно збирати інформацію про об’єкти за допомогою трекінгу. Для його реалізації потрібно проаналізувати існуючі методи та технології для виявлення та відстеження об’єктів. На основі отриманих результатів необхідно розробити програму, яка виконуватиме функції згідно з цим технічним завданням.

2.3 Вимоги до програми

Розроблена програмна система повинна мати наступні функціональні можливості:

- розпізнавати небезпечні ситуації;
- повідомляти користувача про виникнення небезпечних ситуацій за допомогою сповіщень у програмі;
- надавати можливість отримувати сповіщення про виникнення небезпечних ситуацій на електрону пошту та у мессенджер;
- надавати можливість користувачу під’єднати до програми камери, під’єднані до пристрою;
- надавати можливість налаштовувати сповіщення;
- надавати можливість регулювати параметри виникнення небезпечних ситуацій;
- надавати параметри виникнення небезпечних ситуацій за замовчуванням;
- надавати користувачу доступ до журналу подій у системі;
- надавати можливість переглядати відео з візуальним відображенням обмежувальних рамок та ідентифікаторів відстежених об’єктів у реальному часі.

До програмної системи висуваються наступні нефункціональні вимоги:

- частота обробки кадрів не повинна бути нижчою за 15 кадрів на секунду, час обробки одного кадру – не вищим за 0.25 секунд;
- частота обробки кадрів при демонстрації результатів трекінгу у реальному часі не повинна бути нижчою за 20 кадрів на секунду.

2.4 Вимоги до програмної документації

В документації до програмної системи повинна міститься наступна інформація:

- специфікація функціональних вимог до системи;
- опис ситуацій, які може розпзінавати система;
- опис архітектури програми;
- інструкції з встановлення та використання.

2.5 Технічно-економічні показники

Розроблена програмна система може застосовуватись користувачами для автоматизації аналізу даних відеоспостереження та розпізнавання небезпечних ситуацій у зонах огляду відеокамер. Створений удосконалений метод трекінгу може бути повторно використаний у схожих програмах або в інших предметних областях, де є потреба у функціоналі трекінгу.

2.6 Технічні вимоги

До програмної системи висунуті наступні технічні вимоги:

- програма повинна працювати на ПК та ноутбуках з операційною системою Windows версії 10 та вище.
- виконання трекінгу для різних камер повинно використовувати можливості паралелізації для рівномірного використання ядер процесору пристрою.

2.7 Стадії та етапи розробки програмної системи

Стадії та етапи розробки програмної системи наведено у таблиці 2.1.

Таблиця 2.1 — Стадії та етапи розробки

Стадія	Етап	Зміст робіт	Строки
Огляд існуючих рішень	Аналіз існуючих систем з функціоналом відеоаналітики.	Вивчення функціональних можливостей та особливостей існуючих рішень.	05.09.21 – 10.09.21

Продовження таблиці 2.1

Стадія	Етап	Зміст робіт	Строки
Технічне завдання	Визначення ситуацій, розпізнавання яких підтримуватиме система	Опис шаблонів ситуацій, що підтримує система, та зумовлених ними технічних обмежень.	10.09.21 – 16.09.21
	Створення специфікації вимог до програмної системи.	Складання документації щодо вимог до програмної системи.	16.09.21 – 22.09.21
Технічний проект	Створення методики трекінгу об’єктів на даних з відеокамер	Розробка методу трекінгу об’єктів на відео	05.09.21 – 01.10.21
Технічний проект	Проектування модулю аналізу відео	Детальне проектування та опис модулю, що отримує та аналізує дані трекінгу	02.10.21 – 12.10.21
	Проектування десктопної програми	Створення діаграм архітектури програми в цілому	02.10.21 – 12.10.21
Робочий проект	Реалізація програмної систем	Кодування логіки програми, реалізація інтерфейсу, налагодження	12.10.21 – 20.10.21
Готовий проект	Випробування системи	Перевірка функціоналу системи, випробування характеристик, аналіз результатів	20.10.21 – 01.11.21

2.8. Висновки до розділу

У розділі сформоване технічне завдання на розробку програмної системи. Створено опис вимог до системи, а також розплановано стадії та етапи роботи.

3 РОЗРОБКА ВДОСКОНАЛЕНОГО МЕТОДУ ТРЕКІНГУ ОБ'ЄКТІВ ТА ЙОГО ВИКОРИСТАННЯ ДЛЯ СИСТЕМ ВІДЕОСПОСТЕРЕЖЕННЯ

Як зазначено у технічному завданні, основна задача програми – розпізнавати небезпечні ситуації та сповіщати про них. Небезпечні ситуації формально визначаються як наявність певних характеристик об'єктів у полі зору або перевищення ними певних показників, при чому значення показників та характеристики встановлюються користувачем або залишаються значеннями за замовчуванням (які підбираються як найбільш універсальні на етапі розробки та коректуються на базі отриманих даних під час роботи системи). Очевидно, вимірювання таких показників вимагає збирання інформації про об'єкти, для чого необхідний метод трекінгу, тобто виявлення об'єктів та їх відстеження у часі.

У даному розділі представлено розробку вдосконаленого методу трекінгу об'єктів, що буде використовуватись для розпізнавання подія, що відносяться до небезпечних ситуацій, у програмній системі для аналізу даних відеоспостереження.

Задачу трекінгу об'єкту можна поділити на два етапи – виявлення та відстеження. Існує значна чисельність рішень різної спрямованості для виконання цих етапів, тож проаналізуємо наявні засоби реалізації для кожного з них.

3.1 Огляд моделей та методів для виявлення об'єктів

Виявлення об'єктів – це комп'ютерна технологія, пов'язана з комп'ютерним баченням та обробкою зображень, яка займається виявленням екземплярів семантичних об'єктів певного класу (наприклад, людей, будівель чи автомобілів) у цифрових зображеннях та відео [22].

Підходи до виявлення об'єктів поділяються на методи на основі нейронних мереж та без їх використання.

Методи без використання нейронних мереж спочатку визначають ознаки за допомогою наведених нижче підходів, а потім використовують техніки для класифікації – наприклад, метод опорних векторів. До найбільш популярних методів можна віднести:

- Фреймворк Віоли-Джонса для виявлення об’єктів

Фреймворк виявлення об'єктів Віоли-Джонса [23] (англ. Viola–Jones object detection framework) — це метод виявлення об'єктів, який запропонували в 2001 році. Метод має дуже високий рівень виявлення і дуже низький показник хибнопозитивних результатів за будь-яких умов [24].

Алгоритм має чотири етапи:

1) вибір ознак Хаара;
2) створення цілісного образу;
3) навчання Adaboost;
4) використання каскадних класифікаторів.

Незважаючи на те, що алгоритм можна навчити виявляти різноманітні класи об’єктів, він позиціонується насамперед як рішення задачі виявлення обличчя. Швидкість роботи алгоритму дозволяє обмежене використання у реальному часі, тому що заявлена частота обробки складає 15 кадрів на секунду [23].

- Масштабонезалежне перетворення ознак

Масштабонезалежне перетворення ознак (англ. Scale-invariant feature transform, SIFT) - це алгоритм комп’ютерного зору для виявлення, опису та встановлення відповідності локальних особливостей зображень, винайдений в 1999 році [25]. Його застосування включають розпізнавання об’єктів, навігацію роботизованих систем, 3D-моделювання, розпізнавання жестів та відстеження на відео.

Алгоритм складається з кількох ключових стадій: масштабно-інваріантне визначення ознак, співставлення ознак і індексація, ідентифікація кластерів за допомогою перетворення Хофа, верифікація моделі за допомогою лінійного методу

найменших квадратів та виявлення статистичних викидів. Саме масштабно-інваріантне визначення ознак є основною особливістю алгоритму, що дозволяє досягти перевершення більшості сучасних методів для зображень зі зміною масштабу в діапазоні до 2,5 разів та поворотом до 50 градусів [26].

- Гістограма орієнтованих градієнтів

Гістограма орієнтованих градієнтів (HOG) [27] — дескриптор ознак, який застосовується в алгоритмах комп'ютерного зору з метою виявлення об'єктів. В оригинальному дослідженні автори реалізували виявлення пішоходів на знімках, згодом розвивши ідею до виявлення людей на відео та інших об'єктів на окремих кадрах.

В основі підходу лежить розрахунок напрямку градієнтів в точках зображення. Такий метод близький до SIFT дескриптора, але його особливістю є використання нормалізації контрасту.

Оскільки дескриптор HOG працює з локальними клітинами, він нечутливий до геометричних перетворень зображення. Щодо результативності – підхід дає майже ідеальні результати на оригінальній пішохідній базі даних MIT, однак зі зміною наборів даних точність значно варіюється [28].

Підходи на базі нейронних мереж здатні повністю виконувати етап виявлення об'єктів без окремого визначення ознак і зазвичай базуються на згорткових нейронних мережах. Розглянемо найбільш використовувані з таких підходів:

- Архітектура R-CNN

Запропонована у 2014 році архітектура R-CNN використовує ідею попереднього вибору регіонів-кандидатів, що отримала назву пропозиції регіонів (англ. region proposals). Така архітектура використовує селективний пошук для вибору 2000 регіонів, які подаються на вхід нейронної мережі. Глибинні шари мережі діють як екстрактор ознак, в той час як повнозв'язні шари використовують ці ознаки для класифікації об'єкту в обраних регіонах. На додаток до передбачення присутності

об’єкту алгоритм також передбачає значення зсуву регіону, що збільшує точність визначення рамки об’єкту [29].

Архітектура R-CNN має ряд недоліків. По-перше, її неможливо реалізувати в реальному часі, оскільки для обробки кожного зображення потрібно близько 47 секунд [30]. По-друге, алгоритм селективного пошуку є фіксованим, тобто на цьому етапі навчання не відбувається. Це може призвести до створення поганих регіонів-кандидатів.

Автор оригінальної архітектури вирішив деякі недоліки R-CNN, побудувавши швидший алгоритм виявлення об’єктів, який отримав назву Fast R-CNN. Замість розрахованих регіонів в нейронну мережу передається вхідне зображення, щоб створити карту ознак. Регіони-кандидати визначаються безпосередньо з карти ознак, і після приведення до фіксованого розміру подаються до повнозв’язних шарів [31].

Причина, через яку Fast R-CNN швидший за R-CNN, полягає в тому, що зникла необхідність щоразу передавати 2000 регіонів у згорткову нейронну мережу. Натомість операція згортки виконується лише один раз для зображення.

У 2016 році було створено архітектуру для виявлення об’єктів Faster R-CNN, яка усуває алгоритм вибіркового пошуку та дозволяє мережі навчатися виявленню регіонів. Подібно до Fast R-CNN, у новій архітектурі зображення подається на вхід до згорткової мережі, яка розраховує карту ознак. Замість використання алгоритму вибіркового пошуку на карті ознак для визначення регіонів, для прогнозування регіонів використовується окрема мережа. Прогнозовані регіони потім приводяться до фіксованого розміру за допомогою шару об’єднання, після чого відбувається класифікація зображення в запропонованому регіоні та прогнозування значень зсуву для обмежувальних рамок [32].

Як показують виміри, Faster R-CNN набагато швидше, ніж його попередники. Тому його можна використовувати навіть для виявлення об’єктів у реальному часі [29].

- Single Shot MultiBox Detector

Single Shot MultiBox Detector (SSD) – підхід до виявлення об'єктів на зображеннях за допомогою однієї глибокої нейронної мережі. Підхід, названий SSD, дискретизує простір обмежувальних прямокутників у набір блоків з різними співвідношеннями сторін і масштабами для розташування карти ознак. Під час прогнозування мережа генерує оцінки наявності об’єктів кожної категорії у кожному блоці і виробляє коригуючі значення зміщення, щоб блоки краще відповідали формі об’єкта. Крім того, мережа поєднує прогнози з кількох карт ознак з різною роздільною здатністю, щоб точніше обробляти об’єкти різного масштабу [33]. Модель SSD є простою порівняно з методами, які вимагають попередньої пропозиції регіонів, оскільки вона повністю виключає генерацію пропозицій і інкапсулює всі обчислення в одній мережі. Завдяки цьому SSD легко навчати та легко інтегрувати в системи, які потребують компонента виявлення.

Експериментальні результати на наборах даних PASCAL VOC, MS COCO і ILSVRC підтверджують, що SSD має порівнянну точність з методами, які використовують додатковий крок пропозиції регіонів, і є набагато швидшою, при цьому забезпечуючи уніфіковану структуру як для навчання, так і для безпосереднього використання. Для входу 300×300 SSD досягає 72,1% середньої точності у тесті VOC2007 при 58 кадрах в секунду на Nvidia Titan X, перевершуючи модель Faster R-CNN [34].

- You Only Look Once

You Only Look Once (YOLO) – новий підхід до виявлення об'єктів. Попередні підходи до виявлення об’єктів використовували класифікатори для виконання виявлення. Замість цього у YOLO одна нейронна мережа прогнозує обмежувальні рамки та ймовірності класів безпосередньо з повних зображень за один прохід [35].

Така уніфікована архітектура надзвичайно швидка. Базова модель YOLO обробляє зображення в режимі реального часу зі швидкістю 45 кадрів в секунду.

Менша версія мережі, Fast YOLO, дає продуктивність у 155 кадрів в секунду, в той же час досягаючи вдвічі кращої середньої точності, ніж інші детектори реального часу. У порівнянні з найсучаснішими системами виявлення, YOLO робить більше помилок локалізації, але має менше шансів передбачити помилкові виявлення (false positive) там, де нічого не існує. YOLO перевершує всі інші методи виявлення, включаючи SSD і R-CNN, з великим відривом при обробці як природних зображень, так і ілюстрацій [36].

- Single-Shot Refinement Neural Network for Object Detection (RefineDet)

Для виявлення об’єктів двоетапний підхід (наприклад, Faster R-CNN) досягає найвищої точності, тоді як одноетапний підхід (наприклад, SSD) має перевагу високої ефективності. Щоб успадкувати переваги обох і подолати їхні недоліки, у RefineDet було запропоновано новий детектор на основі одного етапу, який досягає кращої точності, ніж двоетапні методи, і підтримує ефективність одноетапних методів. RefineDet складається з двох взаємопов’язаних модулів, а саме: модуля уточнення прив’язки та модуля виявлення об’єктів. Перший модуль має на меті відфільтрувати порожні зони, щоб зменшити простір пошуку для класифікатора, і приблизно налаштувати розташування та розміри регіонів, щоб забезпечити кращу ініціалізацію для подальшої обробки. Другий модуль бере отримані регіони як вхідні дані від першого і виконує їх подальше уточнення та прогнозування міток класів.

Експерименти з PASCAL VOC 2007, PASCAL VOC 2012 і MS COCO демонструють, що RefineDet досягає мети використання переваг двох підходів для отримання подібної до них точності виявлення з високою ефективністю [37].

- Focal Loss for Dense Object Detection (Retina-Net)

Автори даного підходу, як і автори RefineDet, звертаються до проблеми поділу методів детекції об’єктів на одноетапні, орієнтовані на швидкість, та двоетапні, популяризовані архітектурою R-CNN, що мають високу точність. Однак замість спроб поєднати найкращі сторони обох методів, пропонується детальний аналіз ситуації та

пошук рішення. Було виявлено, що головною причиною проблеми є екстремальний дисбаланс класів переднього плану та фону, який виникає під час навчання повнозв'язних шарів детекторів. Усунути цей дисбаланс класів можна було, змінивши стандартну перехресну ентропію таким чином, щоб вона зменшила ваги значення loss, присвоєне добре класифікованим прикладам. Нова функція втрат «focal loss» зосереджує навчання на невеликому наборі складних прикладів і запобігає переповненню детектора великою кількістю легких негативних відповідей під час навчання.

Щоб оцінити ефективність рішення було розроблено та навчено новим методом простий детектор RetinaNet. Результати показують, що RetinaNet може зрівнятися зі швидкістю попередніх одноступеневих детекторів, перевершуючи при цьому точність усіх існуючих сучасних двоступеневих детекторів [38].

3.2 Огляд моделей та методів для відстеження об'єктів

Відстеження об'єктів - процес визначення місця розташування об'єктів у часі за допомогою камери.

Більшість алгоритмів відстеження використовують підхід, який називається відстеженням за виявленням. Він включає незалежний детектор, який застосовується до всіх кадрів зображення для виявлення об'єктів, та трекер, який використовує вихідні дані детектору. Дані виявлення на кожному кадрі використовуються для відстеження шляхом їх з'єднання та призначення ідентичних ідентифікаторів обмежуючим рамкам, що містять той самий об'єкт [39].

За способом використання кадрів відео відстеження може працювати двома методами:

- Пакетний метод: алгоритми використовують інформацію з майбутніх відеокадрів, коли визначають ідентичність об'єкта в певному кадрі. Ця методологія забезпечує кращу якість відстеження.

- Онлайновий метод: у той час як алгоритми пакетного відстеження отримують доступ до майбутніх кадрів, онлайн-алгоритми використовують лише поточну та минулу інформацію, щоб зробити висновки щодо певного кадру.

Онлайн-методи відстеження зазвичай працюють гірше, ніж пакетні, через те, що вони залишаються обмеженими поточним кадром. Однак ця методологія необхідна для роботи у реальному часі.

Згорткові нейронні мережі залишаються найбільш використовуваним методом для відстеження об’єктів [40]. Однак серед методів також наявні рекурентні нейронні мережі (RNN), автокодери (AE), генеративні змагальні мережі (GAN), сіамські нейронні мережі (SNN) та інші алгоритми, не засновані на нейронних мережах.

Розглянемо найпопулярніші реалізації методів відстеження об’єктів:

- Відстеження об'єктів OpenCV

Засоби відстеження об’єктів OpenCV включають трекери BOOSTING, MIL, KCF, CSRT, MedianFlow, TLD, MOSSE та GOTURN. Кожен з цих трекерів найкраще підходить для різних цілей.

Вибір алгоритму відстеження об’єктів OpenCV залежить від переваг і недоліків цього конкретного трекера. CSRT найкраще підходить, коли користувачеві потрібна більш висока точність відстеження об’єктів і його влаштовує повільна частота обробки кадрів. KCF – не такий точний у порівнянні з CSRT, але забезпечує порівняно вищу частоту обробки. Трекер MOSSE найшвидший в OpenCV, але його точність навіть нижча, ніж відстеження за допомогою KCF. GOTURN є єдиним алгоритмом для відстеження об’єктів в бібліотеці OpenCV, що заснований на глибинній нейронній мережі [41].

- Multi-domain convolutional neural network

Багатодоменна згорткова нейронна мережа (MDNet) – архітектура для відстеження, заснована на дискримінаційно навченій згортковій нейронній мережі. Метод включає попереднє тренування згорткової мережі на великому наборі відео,

щоб отримати загальні представлення про об’єкти. Мережа складається із спільних шарів і кількох гілок рівнів, і кожна гілка відповідає за бінарну класифікацію для визначення об’єкту в кожному домені (окремій навчальній послідовності). Навчання мережі виконується по відношенню до кожного домену ітераційно, щоб отримати загальні представлення в спільних шарах. При відстеженні об’єктів у новій послідовності створюється нова мережа, що об’єднує спільні шари в попередньо навченій CNN з новим шаром бінарної класифікації, який оновлюється онлайн. Відстеження в режимі онлайн здійснюється шляхом оцінки регіонів-кандидатів, випадково відібраних навколо попередньої позиції об’єкту [42].

Алгоритм показує чудову точність у порівнянні з сучасними методами в існуючих тестах. Запропоноване у [43] покращення методу прискорило роботу оригінальної мережі приблизно у 25 разів, що дозволило використовувати його для відстеження у реальному часі.

- Simple Online and Realtime Tracking

Simple Online and Realtime Tracking (SORT) - підхід до відстеження кількох об’єктів, сфокусований на ефективній роботі у реальному часі. Незважаючи на використання лише елементарної комбінації знайомих методів, таких як фільтр Калмана для компонентів відстеження, цей підхід досягає точності, порівнянної з іншими онлайн-трекерами. Крім того, завдяки простоті методу відстеження, трекер оновлюється зі швидкістю 260 Гц, що в 20 разів швидше, ніж інші сучасні трекери [44].

Наведений у [45] подальший розвиток підходу інтегрує інформацію про зовнішній вигляд об'єктів відстеження за допомогою додаткової нейронної мережі. Експериментальні оцінки показують, що внесені зміни зменшують кількість перемикань ідентифікаторів на 45%.

3.3 Детальний огляд обраних для подальшого дослідження існуючих моделей та методів

3.3.1 Виявлення об’єктів за допомогою нейронної мережі YOLO

Зважаючи на перевагу над іншими підходами у співвідношенні точність/швидкість, установлено, що для ефективного виявлення об’єктів у реальному часі треба обрати модель YOLO.

YOLO – однорівнева нейронна мережа для виявлення регіонів знаходження об’єктів за один прохід по зображенню. Історія розвитку та еволюції мереж YOLO включає п’ять основних та декілька побічних версій, з яких три перших створено авторами оригінальної мережі, інші були пізніше реалізовані спільнотою.

Основна ідея YOLO – поділ зображень на клітини (grid cells) розміром 13x13, до яких прикріплюються обмежувальні рамки, що визначаються при навчанні. Далі для кожної клітини визначаються 4 параметри передбаченої рамки (x, y – позиція та h, w – висота та ширина відповідно), мітка об’єкту та параметер впевненості у наявності об’єкту у рамці (confidence score). Таким чином знижується кількість обчислень, тому що виявлення та класифікація виконуються одночасно. Однак такий підхід призводить до появи великої кількості дубльованих передбачень, так як декілька клітин передбачують різні рамки для одного об’єкту. YOLO вирішує цю проблему за допомогою техніки NMS (non-max suppression) – множина отриманих обмежувальних рамок фільтрується для виявлення рамок з найбільшим значенням впевненості, які і стають вихідними рамками об’єктів. Оригінальна архітектура YOLO має 24 згорткових шари з 2 повнозв’язними шарами наприкінці [35].

YOLOv2 була запропонована для вирішення основних проблем YOLO — виявлення малих об’єктів у групах та точності локалізації. YOLOv2 збільшує середню точність мережі на 2 відсотки, вводячи пакетну нормалізацію. Набагато більш ефектним доповненням, запропонованим у YOLOv2, стало додавання блоків

прив'язки. YOLO прогнозує один об’єкт на клітину сітки. Це створює проблеми, коли одна клітинка містить більше одного об’єкта. YOLOv2 позбавляється від цього обмеження, дозволяючи передбачати 5 обмежуючих прямокутників для однієї клітинки. Число 5 емпірично отримано як таке, що має хороший компроміс між складністю моделі та продуктивністю прогнозування. Основою архітектури YOLOv2 стала мережа DarkNet-19, що містить 19 згорткових шарів і 5 шарів максимального об’єднання (Max-Pooling) [36].

YOLO9000 був запропонований як алгоритм для виявлення більшої кількості класів. Набір даних виявлення об’єктів, на якому навчалися перші дві моделі (COCO), має лише 80 класів. Для порівняння, мережа класифікації ImageNet може обробляти 22 000 класів. Щоб уможливити виявлення багатьох інших класів, YOLO9000 використовує мітки як із ImageNet, так і з COCO [36]. Таке збільшення потужності підходу завдало впливу на точність, тому YOLO9000 демонструє гірші показники, ніж інші мережі YOLO.

Хоча YOLOv2 — це надшвидка мережа, з часом почали з’являтися альтернативи, які пропонують кращу точність (наприклад детектори SSD). Хоча вони набагато повільніші, вони випереджають YOLOv2 і YOLO9000 з точки зору точності. Покращення YOLO за допомогою сучасних CNN було запропоновано у версії YOLOv3. У ній використовується мережа DarkNet-53, архітектурна новизна якої дозволяє робити прогнозування в трьох різних масштабах, що усуває недоліки мережі у виявленні малих об’єктів. YOLOv3 передбачає лише три обмежувальні рамки на клітину, але у трьох різних масштабах, тобто до 9 блоків [46].

YOLOv4 пропонує загальне оновлення архітектури та введення нових покращень у метод навчання мережі для подальшого покращення точності виявлення. Автори також пропонують версію YOLOv4 Tiny, яка забезпечує швидше виявлення об’єктів і вищий FPS, знижуючі точність передбачення [47].

YOLACT (You Only Look At Coefficients) є застосуванням принципу YOLO для сегментації у реальному часі [48].

YOLOv5 — це проект з відкритим вихідним кодом, який складається з сімейства моделей виявлення об’єктів і засобів виявлення на основі моделі YOLO, попередньо навченої на наборі даних COCO [49].

Прийнявши до уваги переваги нововведень описаних нейронних мереж, для виконання виявлення у проекті обрано YOLOv4. Наведемо детальний опис її структури та основних відмінностей від інших версій мережі.

Першим кроком у підвищенні точності в YOLOv4 було збільшення глибини мережі, щоб розширити поле зору і збільшити складність моделі. Основою моделі стала мережа Darknet53 [47]. На відміну від Darknet19, що використовувалась у більш ранніх версіях, вона містить з’єднання пропуску (residual connections), що дозволяють значенням градієнту при навчанні проходити через мережу, пропускаючи нелінійні функції активації для запобігання експоненціальному зростанню помилки при зворотному розповсюдженні [50]. Загальне число згорткових блоків також було збільшено. Архітектуру мережі Darknet53 наведено у таблиці 3.1.

Таблиця 3.1 – Архітектура Darknet53

Тип шару	Кількість повторень	Кількість фільтрів	Розмір ядра / зсув	Вихідний розмір
Convolutional	1	32	3x3	256x256
Convolutional	1	64	3x3 / 2	128x128
Convolutional	1	32	1x1	
Convolutional		63	3x3	
Residual				128x128
Convolutional	1	128	3x3 / 2	64x64
Convolutional	2	64	1x1	
Convolutional		128	3x3	
Residual				64x64

Продовження таблиці 3.1

Тип шару	Кількість повторень	Кількість фільтрів	Розмір ядра / зсув	Вихідний розмір
Convolutional	1	256	3x3 / 2	32x32
Convolutional	8	128	1x1	
Convolutional		256	3x3	
Residual				32x32
Convolutional	1	512	3x3 / 2	16x16
Convolutional	8	256	1x1	
Convolutional		512	3x3	
Residual				16x16
Convolutional	1	1024	3x3 / 2	8x8
Convolutional	4	512	1x1	
Convolutional		1024	3x3	
Residual				8x8
Avgpool	1		Global	
Connected	1		1000	
Softmax	1			

Для вирішення проблем з виявленням малих об’єктів у мережу було додано додатковий блок SPP (spatial pyramid pooling) [51]. Існуюча раніше мережа вимагала вхідного зображення фіксованого розміру. Ця вимога була «штучною» і знижувала точність розпізнавання зображень малого розміру. У блоках SPP просторова інформація у вигляді карт ознак витягується декількома шарами пулінгу (pooling) для різних масштабів, що потім конкатенується. Вихідна карта ознак містить інформацію з шарів пулінгу усіх масштабів у фіксованому розмірі незалежно від розмірів вхідного зображення.

Як метод агрегації параметрів з різних рівнів остовної мережі для різних рівнів детектора використовується PANet [52]. Ця архитектура замінила FPN з YOLOv3 з метою подальшого додавання функціоналу сегментації.

«Голова» мережі залишилася незмінною порівняно з YOLOv3 – вона використовує поділ зображення на клітини з обмежувальними рамками, з яких визначаються найбільш підходящі для кожного об’єкту.

Результуюча схема мережі наведена на рис. 3.1.

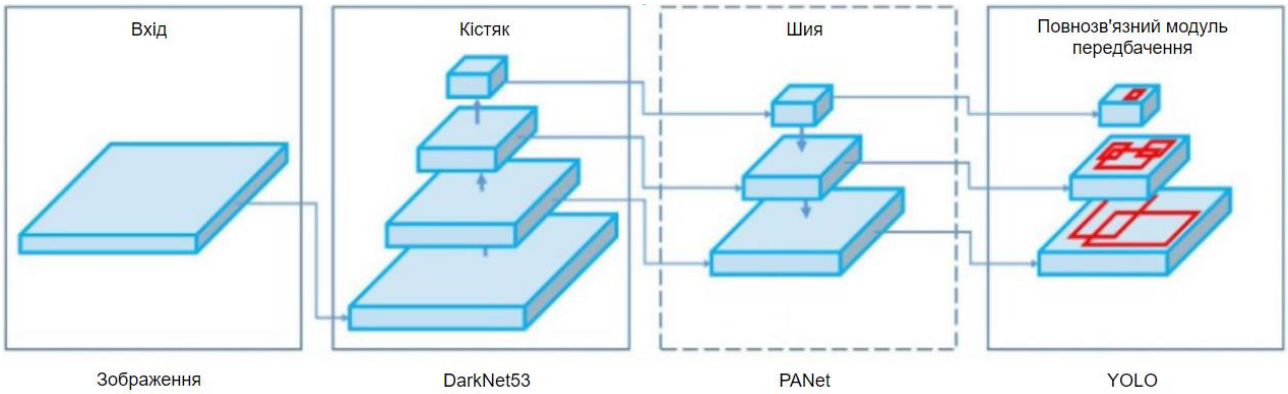

Рисунок 3.1 – Схема мережі YOLOv4 [модіфіковано з 47]

Зазнав покращень також метод навчання нової версії YOLO – було додано нові методи доповнення даних (CutMix [53] та Mosaic), метод блочного відкидання пікселів для запобігання перенавчанню (DropBlock regularization) [54] та згладжування міток (class label smoothing) – зменшення цільової впевненості (confidence) до 0.9 замість 1.0 при обчисленні помилки для запобігання запам’ятовуванню певних класів.

Далі у підрозділі 3.4.1 описані внесені в YOLOv4 зміни для покращення точності виявлення.

3.3.2 Відстеження об'єктів з використанням підходу SORT

Алгоритм виявлення надає покадрову інформацію про наявні у полі зору об’єкти, проте для виконання трекінгу недостатньо лише порівнювати послідовні кадри – це впливає на швидкодію та призводить до частих перемикань ідентифікаторів через помилки виявлення на деяких кадрах. Тому для ефективного трекінгу необхідний метод відстеження. У роботі основою такого методу обрано підхід SORT.

SORT – підхід до відстеження кількох об’єктів, що використовує комбінацію відомих простих та ефективних алгоритмів для досягнення значної продуктивності [44].

Фільтр Калмана. Задача, що вирішує фільтр Калмана [5] – маючи вимірювану величину з сенсора (у випадку відстеження об’єктів – з детектора), що містить похибку, максимально точно передбачити її реальне значення. Фільтр найкраще працює для лінійних систем із залученими гауссівськими процесами. У випадку відстеження об’єктів треки майже не виходять з лінійної області, а більшість шумів, що спричиняють помилки детектування, потрапляють в область Гаусса. Отже, задача підходить для використання фільтрів Калмана.

Ключова ідея оригінального підходу SORT – використання фільтру Калмана для пов’язування передбачень стану об’єкту на попередньому кадрі з виявленим станом на поточному кадрі, підтримуючи вірогідність помилок у процесі. Стан об’єкта на кожному кадрі – це вектор, що містить 8 змінних (u, v, a, h, u`, v`, a`, h`), де (u, v) – центр рамки об’єкту, a – співвідношення сторін рамки, h – висота рамки, інші змінні – відповідні швидкості. Для стану об’єкту у лінійному процесі рівняння 3.1 описують покази детектору та передбачене фільтром значення на одному кадрі.

$$x`_{k+1} = x_k + u_k + t_k;\ z_{k+1} = x_k + n_k\ , \tag{3.1}$$

де k – номер поточного кадру,

x – реальне значення стану,

x` – передбачене значення стану,

u – величина, що контролює зміну стану між кадрами для передбачення,

t – похибка передбачення,

z – покази детектору,

n – похибка детектору.

Ідея полягає в тому, що для отримання наближення до істинного стану необхідно вирахувати коефіцієнти для його розрахунку за показаннями неточного детектору і розрахованим передбаченням. Розрахунок виконується за формулою 3.2.

$$x_{k+1}^{best} = K * z_{k+1} + (1 - K) * (x_k^{best} + u_k), \quad (3.2)$$

де x^{best} – найкраще наближення до істиного стану,

K – коефіцієнт розрахунку передбачення.

Оскільки фільтр працює рекурсивно, на кожному кроці необхідно мінімізувати помилку за допомогою коефіцієнту К. Для мінімізації використовується середньоквадратичне значення помилки, яке визначається отриманою після підстановки значень та виконання перетворень формулою 3.3.

$$E(e_{k+1}^2) = E((x_{k+1} - x_{k+1}^{best})^2) = (1 - K)^2(E(e_k^2) + E(t_k^2)) + K^2E(n_{k+1}^2), \quad (3.3)$$

де E(var) означає середнє значення змінної var,

e – значення помилки.

Після прирівняння похідної від виразу 3.3 до нуля формулюється рівняння 3.4 для мінімізації похибки.

$$K_{k+1} = \frac{E(e_k^2) + E(t_k^2)}{E(e_k^2) + E(t_k^2) + K^2E(n_{k+1}^2)} \quad (3.4)$$

Формула 3.4 є так званим посиленням Калману для поставленої задачі та дозволяє ітераційно оптимізувати значення К для мінімізації середньоквадратичної помилки передбачення.

Отримані при розрахунку наближення до істинного стану положення об’єктів можна використовувати для подальшого призначення ідентифікаторів.

Відстань Махаланобіса. Наступний етап відстеження – розрахування відстаней між спрогнозованими фільтром та реально виявленими позиціями об’єктів. Автори SORT використали для цієї мети відстань Махаланобіса. Ця метрика більш точна, ніж евклідова відстань, оскільки вимірюється відстань між змінними, що корелюють. У загальному випадку відстань Махаланобіса обчислюється за формулою 3.5 [55].

$$D_M(x) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)}\,, \tag{3.5}$$

де $x = (x_1, x_2, \dots, x_N)^T$ – багатовимірний вектор,

$\mu = (\mu_1, \mu_2, \dots, \mu_N)^T$ – середні значення множини,

S – матриця коваріації.

Якщо відстань між передбаченим станом з фільтру Калмана та виявленим станом менше за встановлену межу, ідентифікатор об’єкта з минулого кадру може асоціюватися з новим станом (рамкою).

Угорський алгоритм. Подальше присвоювання ідентифікаторів відповідним об’єктам є відомою задачею оптимізації, і виконується за допомогою Угорського алгоритму [6] за поліноміальний час.

Сформулювати задачу можна у термінах теорії графів: для заданого дводольного графу *G=(S,T,E)*, де *S* – наявні об'єкти з ідентифікаторами з попередніх кадрів та *T* – нові передбачені на поточному кадрі об’єкти, *E* – вартості ребер, що дорівнюють відстаням Махаланобіса між станами об’єктів; необхідно знайти повне пароутворення.

Алгоритм використовує функцію потенціалу, що визначається як y: $(S \cup T) \to R$, якщо $y(i) + y(j) \leq c(i,j)$ для всіх $i \in S$, $j \in T$, де $c(i,j)$ – вартість ребра між i та j. Тоді потенціал для всього графу визначається як $y = \sum_{v \in (S \cup T)} y(v)$, а вартість будь-якого повного пароутворення є меншою, ніж це значення.

Використовуючи наведені дані, алгоритм знаходить повне пароутворення та потенціал з однаковою вартістю, що доказує оптимальність обох значень. Складність виконання цієї версії алгоритму дорівнює $O(n^4)$.

У результаті виконання угорського алгоритму формується набір присвоєних ідентифікаторів, що зі зміною положень об’єктів з часом відповідають одним й тим самим об’єктам. Такі ідентифікатори являються фактичним результатом трекінгу об’єктів.

Підхід DeepSORT. Наведена комбінація методів досить ефективно працює для виконання задачі відстеження, проте не може обробляти втрати ідентифікаторів у зв’язку з перекриваннями об’єктів. Для вирішення цієї проблеми було запропоновано метод DeepSORT [45], ідея якого – підтримувати інформацію про зовнішній вигляд об’єктів у вигляді глибинних ознак за допомогою окремої нейронної мережі. Створення такої мережі досягнуто шляхом тренування простого класифікатора, у якого потім видаляється шар класифікації. Тоді нейронна мережа, що залишається, має вихід у форматі вектору ознак розмірністю 128x1. Архітектура мережі наведена у таблиці 3.2.

Таблиця 3.2 – Архітектура нейронної мережі у методі DeepSORT

Тип шару	Розмір ядра / зсув	Вихідний розмір
Convolutional	3x3 / 1	32x128x64
Convolutional	3x3 / 1	32x128x64
Max pooling	3x3 / 2	32x64x32
Residual	3x3 / 1	32x64x32
Residual	3x3 / 1	32x64x32
Residual	3x3 / 2	64x32x16
Residual	3x3 / 1	64x32x16
Residual	3x3 / 2	128x16x8
Residual	3x3 / 1	128x16x8
Dense		128
Batch and normalization		128

Як вихідна метрика відстані тепер використовується косинусна відстань [56] між векторами ознак (формула 3.6), з якої безпосередньо визначаються асоціації ідентифікаторів з об’єктами.

$$D_{cos} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \ , \tag{3.6}$$

де A, B – багатовимірні вектори, між якими проводиться пошук відстані,

n – розмірність векторів.

Метод DeepSORT має обмеження – глибинна нейронна мережа спроектована та навчена тільки для відстеження людей і вимагає значних змін для відстеження декількох класів об’єктів.

3.4 Удосконалений метод трекінгу об'єктів для відеоспостереження

3.4.1 Зміни для покращення точності виявлення об’єктів з метою збільшення точності подальшого трекінгу

YOLOv4 демонструє значну точність та швидкість роботи, але приймаючи до уваги той факт, що виконання виявлення буде відбуватись на статичній камері, тобто фонове зображення буде залишатися незмінним, є доречним навчання нейронної мережі на кінцевому пристрої.

Ідея такого навчання викликає декілька проблем. По-перше, виконання навчання на пристрої користувача може викликати небажані зміни у вагах мережі, що будуть позначатися на досвіді використання. По-друге, продуктивність системи під час навчання буде значно знижено. По-третє, для навчання необхідні розмічені навчальні дані.

Вирішення першої та частково другої проблеми можливо завдяки використанню донавчання на нових даних. Ідея такого підходу – внесення змін у ваги вже навченої на великому наборі даних мережі за допомогою ще одного етапу навчання на малому

наборі даних, специфічних для коректного випадку. Позитивними моментами підходу є можливість використання малої кількості елементів в навчальній вибірці та значне прискорення адаптації до конкретного завдання [57, 58]. У контексті проекту є сенс навчати тільки повнозв'язний модуль передбачення мережі, заморозивши всі інші ваги. Так можна запобігти небажаним змінам у екстракторі ознак та зменшити затрати обчислювальних ресурсів.

Розмітку даних для навчання зазвичай виконують вручну, але на пристрої користувача такий підхід неможливий. Альтернативою є використання автоматичних засобів виявлення, зокрема тих, що мають більшу, ніж цільова мережа, точність. Такий підхід, однак, створює додаткове навантаження на обчислювальні ресурси.

Вирішити проблеми і додаткового навантаження, і зниження продуктивності системи в цілому дозволить введення «режиму пристосування» у цільову систему. Протягом певного часу після встановлення система знаходитиметься у такому режимі, збираючи дані для навчання, щоб розмітити їх за допомогою більш точного алгоритму та виконати донавчання цільової мережі на нових даних. Штатні функції системи у цей час будуть виконуватися на базі цільової мережі, але матимуть менші точність та швидкість.

Як алгоритм розмітки даних було обрано мережу Fast R-CNN, як таку, що демонструє 70% середньої точності на наборі даних PASCAL VOC 2012 [31] (у порівнянні з 45.5% в YOLOv4 [47]), та має близький до YOLO формат даних, що дозволяє легше інтегрувати її у систему.

Перший етап навчання – збір даних. Для найкращого ефекту варто обрати зображення, на яких присутні об'єкти, та додати рівну кількість зображень, що не містять їх (або містять лише постійні статичні об'єкти). Тому для кожної камери обираються всі кадри з новими об'єктами (випадково серед усіх кадрів, на яких присутній цей об'єкт) та випадкові «порожні» зображення. Час збирання такого набору даних буде варіюватися залежно від інтенсивності поява об'єктів у зоні

видимості камер, тому варто встановити обмеження. Збір даних для всіх камер триває максимум 150 годин. Після збору 100 зображень набір для камери вважається завершеним, якщо ж по завершенні часу було зібрано менше – набір видаляється з остаточних даних для навчання.

Подальші етапи вимагають змін у налаштуванні нейронної мережі та мають велике навантаження на систему, тому функціонування системи на їх час припиняється.

Наступний етап після збору – розмічування даних за допомогою мережі Fast R-CNN. Зображення масштабуються до розмірів 608x608 та обробляються мережею Fast R-CNN, в результаті для кожного зображення отримуємо файл зі списком об’єктів та їх позицій і розмірів обмежувальних рамок.

Маючи розмічені дані, можна приступати до процесу навчання. Параметри навчання наведено у таблиці 3.3.

Таблиця 3.3 – Параметри навчання мережі

Параметр	Значення	Пояснення
channels	608x608x3	Оптимальний розмір зображень згідно документації YOLO [59] – число, що ділиться на 32, 3 – число каналів RGB зображення.
batch	10	Число зображень у одному пакеті навчання, підібрано емпірично з урахуванням рекомендацій у документації.
subdivisions	2	YOLO визначає розмір міні-пакету як *batch/ subdivisions.* Завдяки поділу на міні-пакети знижується навантаження на оперативну пам’ять.
max_batches	100	У YOLO *max_batches* фактично визначає кількість епох навчання як *dataset_size / max_batches * batch_size*
train;test;val sizes	0.8 * *dataset_size; 0.1 * dataset_size; 0.1 * dataset_size;*	Розміри наборів для навчання, валідації та тестування мережі.

Продовження таблиці 3.3

Параметр	Значення	Пояснення
learning_rate	0.00001	Параметр, що відповідає за швидкість навчання. Використовуємо мале значення, оскільки донавчання не повинно значно змінювати ваги мережі [60].
Steps	*dataset_size*0.8; dataset_size*0.9*	Параметри, що визначають моменти зміни параметру *learning_rate*, яких вимагає архітектура YOLO.

Оскільки основна мета навчання мережі – підвищення точності виявлення об’єктів, для оцінки результатів навчання використаємо метрику IoU [61] (intersection over union) – відношення площі перетину отриманої та очікуваної областей сегментації до площі їх об’єднання (формула 3.7).

$$IoU = \frac{S(P \cap GT)}{S(P \cup GT)}, \tag{3.7}$$

де S – площа,

P – передбачена обмежувальна рамка,

GT – реальна обмежувальна рамка.

Наступний етап – тестування результатів. Варто зазначити, що незважаючи на заходи з запобігання можливого негативного впливу навчання на точність мережі, вірогідність такого впливу все ще існує. Для виявлення такого впливу можна оцінити точність виявлення налаштованої мережі у порівнянні з оригінальною. Для цього будемо перевіряти мережу на тестовій частині набору даних до та після навчання, і у випадку зменшення метрики IoU відміняти його результати. У разі якщо негативного впливу не сталося, та результати навчання успішні, мережа перекомпілюється з отриманими новим файлом ваг з’єднань та замінює оригінальну у алгоритмі виявлення. Система звільняє використані ресурси та після перезавантаження починає працювати у штатному режимі.

Робота з мережею в режимі навчання вимагає наявності на комп’ютері користувача середовища Python 3.0, що за відсутності буде встановлюватися разом з системою. За допомогою засобів керування пакетами буде встановлено додаткові бібліотеки для навчання мережі у вигляді модулів Python. Також встановлюватиметься нескомпільована версія YOLOv4, яка буде видалена по завершенні донавчання.

3.4.2 Модифікований метод трекінгу

Оскільки ключовим параметром удосконалення методу трекінгу в рамках даної роботи є точність, доцільно використати переваги як оригінального підходу SORT, так і нової версії DeepSORT. Незважаючи на те, що в експериментах метод DeepSORT демонструє значення точності відстеження декількох об’єктів (MOTP, Multi-object tracking precision) 79.1 при значенні 72.1 в оригінальному методі, автори дослідження визнають, що в деяких випадках метод SORT має кращі результати [45]. До того ж оригінальний метод SORT дозволяє відстежувати об’єкти будь-якого з класів, які підтримує метод виявлення – на відміну від методу DeepSORT, глибинна нейронна мережа у якому навчена для виявлення глибинних ознак лише для об’єктів класу «людина».

Для об’єднання підходів необхідно використати дані з фільтру Калмана та вектори ознак з нейронної мережі в одному методі зіставлення станів об’єктів з ідентифікаторами. Для цього введемо спільну метрику відстані, наведену у формулі 3.8.

$$D_t = hD_{cos} + (1 - h)D_M \,, \tag{3.8}$$

де h – коефіцієнт вкладу кожної частини методу, що визначається емпірично.

Отримане нове значення відстані можна використовувати як вартість ребер дводольного графу в Угорському алгоритмі, наведеному в оригінальному підході SORT.

Етапи методу трекінгу об’єктів до та після внесення змін наведено на рис. 3.2 та 3.3.

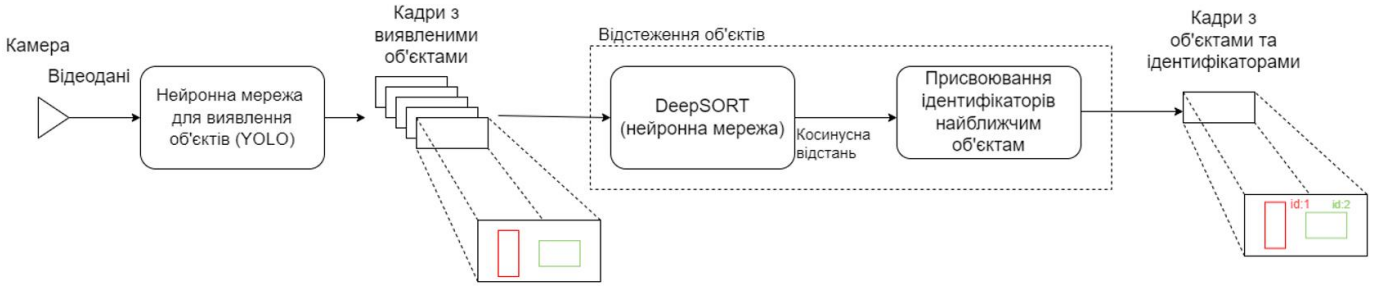

Рисунок 3.2 – Етапи оригінального методу трекінгу об'єктів (до внесення змін)

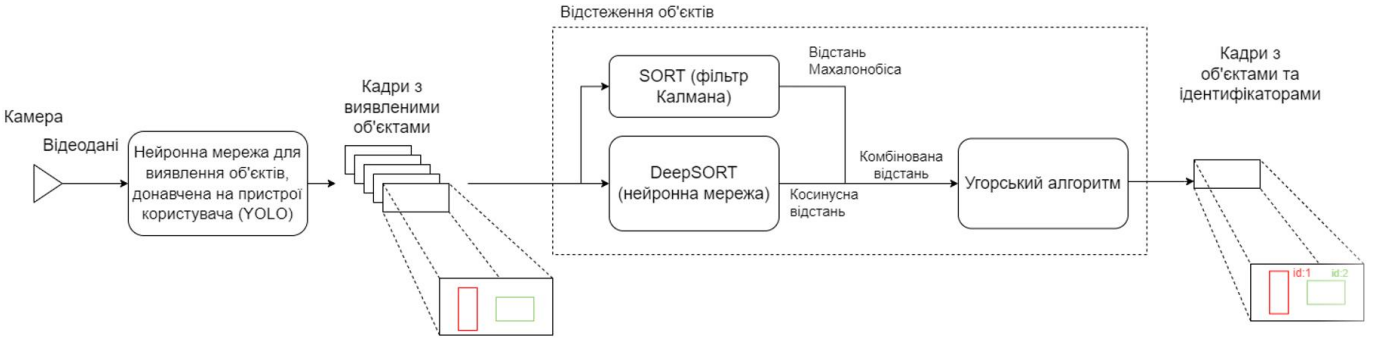

Рисунок 3.3 – Етапи удосконаленого методу трекінгу об’єктів

Наведені зміни незначно впливають на швидкість роботи систем відеоспостереження за рахунок використання відносно простих для обчислення математичних та алгоритмічних методів.

3.5 Алгоритм розпізнавання небезпечних ситуацій на основі виявлення та відстеження об’єктів

Програмна система має виконувати задачу розпізнавання небезпечних ситуацій у реальному часі та сповіщення про них. Основна та найскладніша з точки зору реалізації частина системи – трекінг об’єктів, вихідні дані якого використовуються для аналізу ситуацій. Оскільки детальний опис методу трекінгу було наведено у підрозділах 3.3 та 3.4, у цьому підрозділі увагу сфокусовано на іншій частині системи, що вимагає детального розгляду – розпізнавання небезпечних ситуацій на основі даних трекінгу.

Для кращого розуміння функції розпізнавання небезпечних ситуацій перш за все визначимо формат вхідних та вихідних даних алгоритму. На вході алгоритм отримує дані трекінгу об’єктів у форматі масиву, кожен елемент якого – кортеж, що містить ідентифікатор об’єкту, його клас, позицію у двовимірному просторі та ширину й висоту обмежувальної рамки. Алгоритм також отримує на вхід значення параметрів розпізнавання ситуацій, що були налаштовані користувачем – набір числових та булевих значень, з якими будуть порівнюватися отримані при аналітиці значення (координати обмежувальних зон, швидкість об’єктів, час знаходження у зоні видимості, коди певних небажаних класів об’єктів тощо). Вихідні дані алгоритму – звіти про ситуації у форматі коду типу ситуації, часу виникнення та додаткових даних про об’єкти.

Оскільки більшість ситуацій вимагають аналізу поведінки відстежених об’єктів, доцільно зберігати інформацію про зміни їх характеристик у часі. Метод трекінгу містить таку інформацію, але у вигляді глибинних ознак у нейронній мережі та параметрів у фільтрі Калмана, надаючи на виході лише ідентифікатори, що вдалося назначити. Тому необхідно підтримувати список відстежених об’єктів разом з інформацією, яка дозволить аналізувати їх поведінку.

Приймаючи до уваги вищезазначене, поділимо алгоритм на дві частини: збір інформації про об’єкти (назвемо її стан об’єктів) та її аналіз.

Розглянемо першу частину алгоритму. Для скорочення називатимемо положенням об’єкту вектор (x, y, h, w), де x та y – координати об'єкту у двовимірному просторі, а h та w – висота та ширина обмежувальної рамки відповідно. Швидкість об’єкту визначається як швидкість зміни кожної координати, а також розмірів обмежувальної рамки, що відповідають руху за віссю глибини.

На етапі збору інформації підтримуватимемо наступну інформацію про кожен об’єкт:

- ідентифікатор;

- клас;
- поточне положення;
- час існування ідентифікатора (тобто час від першої появи об’єкту);
- поточна швидкість;
- середня швидкість;
- кількість поява у полі зору (тобто виходу з зони огляду камери та подальшого повернення в неї);
- час останнього знаходження у полі зору;
- середній час знаходження у полі зору;
- середнє прискорення (абсолютне значення).

Для калібрування значень за замовчуванням для кожного класу об’єктів підтримуватимемо середнє значення всіх параметрів стану, окрім поточного положення та швидкості, а також поточну та середню кількість об’єктів кожного класу.

На етапі аналізу інформації необхідно порівнювати параметри стану об’єктів з такими, що вважаються критичними для виникнення небезпечної ситуації. Наведемо список небезпечних ситуацій, що оброблятимуться, та методи їх розпізнавання:

- Поява об’єкту. Наявність об’єкту визначеного налаштуваннями класу у полі зору вважається небезпечною ситуацією.
- Скупчення об’єктів. Визначається як перевищення критичної кількості об’єктів у зоні видимості. Використовується тільки для об’єктів класу «людина».
- Перетин ліній/знаходження у зоні. Користувач може встановити до 3 обмежувальних зон на кожну камеру. При виявленні положення об’єкту в обмежувальних зонах ситуація вважається небезпечною.
- Затримка у зоні огляду. Включає перевірку на перевищення критичних значень часу останнього знаходження у полі зору, середнього часу знаходження у полі зору та часу існування ідентифікатору.

- Зникнення об’єкту. Якщо час існування об’єкту визначеного налаштуваннями класу перевищує встановлений, об’єкт вважається постійним. Зникнення такого об’єкту з поля зору розцінюється як небезпечна ситуація.
- Перевищення швидкості. Якщо поточна або середня швидкість об’єкту перевищують критичні значення, як у абсолютному вигляді, так і за напрямом одної з осей, це вважається небезпечною ситуацією.
- Раптове зникнення об’єкту. Раптовим вважається зникнення не біля границі зони огляду та без перекривання поточного положення іншим об’єктом. Факт такого зникнення може бути індикатором саботажу камери, тому випадок вважається небезпечною ситуацією.
- Підозріла поведінка. Ситуація розпізнається тільки для об’єктів класу «людина» та є найбільш абстрактною з наведених. Аналіз на підозрілу поведінку включає, перш за все, перевищення критичного значення абсолютного середнього прискорення, що є індикатором підозрілих переміщень. Аналізується також кількість появ у полі зору, середній час знаходження у ньому та середня швидкість. Критичні значення параметрів не можуть бути налаштовані користувачем, але можуть бути виключені ним з аналізу. Оскільки аналіз вимагає наявності середніх значень для всіх об’єктів, ситуація не обробляється у «режимі пристосування», поки виконується збір такої інформації.

Після виконання етапу аналізу отримані дані виглядають як коди розпізнаних ситуацій та об’єкти, що їх спричинили. У випадку виникнення кількох ситуацій з одним об’єктом або однієї ситуації з кількома об’єктами отримані дані агрегуються. Отриманий звіт включає коди ситуацій, час їх виникнення, інформацію про стан об’єктів, що задіяні, окремі значення перевищених критичних параметрів та знімки з камери у момент виникнення.

Ітерація алгоритму виконується після кожного оновлення даних, тобто після обробки кожного кадру методом трекінгу. Для покращення швидкодії алгоритм

працює паралельно з роботою модулю трекінгу над наступним кадром відео, але ітерація може відбуватися довше за його обробку, тому кадри трекінгу складаються у чергу та обробляються з неї послідовно. У випадку надмірного збільшення черги для нейтралізації впливу на час роботи над кадром всієї системи кадри можуть пропускатися. Такий підхід дозволяє швидко повернути продуктивність системи до нормального рівня під час навантаження на алгоритм розпізнавання ситуацій без значних втрат даних.

Підсумовуючи, наведемо покроковий опис алгоритму розпізнавання небезпечних ситуацій:

1. Отримання на вхід результатів трекінгу об’єктів на поточному кадрі та параметрів ситуацій.
2. Збір та оновлення інформації про кожен відстежений об’єкт.
3. Оновлення агрегованої інформації для всіх класів об’єктів.
4. Перевірка інформації про всі відстежені об’єкти та їх класи на наявність небезпечних ситуацій шляхом порівняння з критичними параметрами для кожного типу таких ситуацій.
5. Формування звітів про розпізнані ситуації.
6. Агрегування звітів про пов’язані ситуації.
7. Передача на вихід отриманих звітів про ситуації.

3.6 Висновки до розділу

У даному розділі створено покращений метод трекінгу об’єктів при відеоспостереженні. Проведено аналіз наявних методів виявлення та відстеження об’єктів. На основі проведеного аналізу для реалізації методу трекінгу обрано нейронну мережу для виявлення YOLOv4 та підхід до відстеження SORT. В обрані методи та моделі внесені модифікації з метою покращення точності відстеження. Внесені зміни дозволили адаптувати метод для роботи трекінгу та відеоаналітики на

його основі з достатньою точністю на пристрої користувача, а також підтримувати роботу з об’єктами з більшого числа класів.

На базі удосконаленого методу трекінгу створено алгоритм розпізнавання небезпечних ситуацій у відеоданих у реальному часі. У розділі наведено опис цього алгоритму, а також деталей реалізації донавчання нейронної мережі для виявлення об’єктів на нових даних.

4 ПРОЕКТУВАННЯ СИСТЕМИ

Проектування дозволяє визначити внутрішню структуру системи та деталізувати її зовнішні якості на базі отриманої на етапі аналізу вимог інформації. У ході проектування створюється детальний опис компонентів системи, який використовується під час розробки. Приймаючи до уваги особливості вимог до розроблюваної системи, а саме високу продуктивність та простоту використання, можна одразу виявити базову форму проекту: необхідно створити систему у вигляді десктопної програми та реалізувати ефективні за точністю алгоритми обробки відео.

Специфікація вимог програмного забезпечення відповідно ISO/IEC/IEEE 29148:2011 [62] – структурований набір вимог до програмного забезпечення та його зовнішніх інтерфейсів. Включає функціональні вимоги, що описують функції, які необхідно реалізувати у системі, та як користувачі будуть взаємодіяти з ними. Опис поведінки системи наводиться у вигляді сценаріїв використання, що описують варіанти взаємодії між користувачем та програмою. Окрім цього специфікація також містить нефункціональні вимоги – ті, що описують властивості та характеристики системи.

На етапі проектування описується також архітектура системи – як високорівнева, так і детальна для кожної з її складових частин.

4.1 Опис функціональних вимог

Розглянемо докладно сценарії варіантів використання програмної системи. Діаграма варіантів використання для системи наведена на рис. 4.1.

Окрім наведених на діаграмі варіантів використання сценаріїв система має виконувати дії, що не вимагають безпосередньої участі користувача у ролі ініціатора. Перелік таких дій включає:

- розпізнавання небезпечних ситуацій у даних з під’єднаних відеокамер;

- надання візуальних повідомлень про розпізнані небезпечні ситуації через користувацький інтерфейс;
- надання повідомлень про розпізнані небезпечні ситуації через інтерфейс програмної системи, електрону пошту та мессенджер Telegram;
- зберігання у журналі подій інформації про розпізнані ситуації, зміни налаштувань, помилки роботи та інші події у системі.

Для системи можна виділити одного актора, що взаємодіє з нею – це безпосередньо Користувач. Основні варіанти використання включають налаштування системи, дії з журналом, отримання повідомлень про ситуації та перегляд вихідного обробленого відеопотоку з камер. Налаштування включають конфігурацію підключених камер, сповіщень та параметрів розпізнавання.

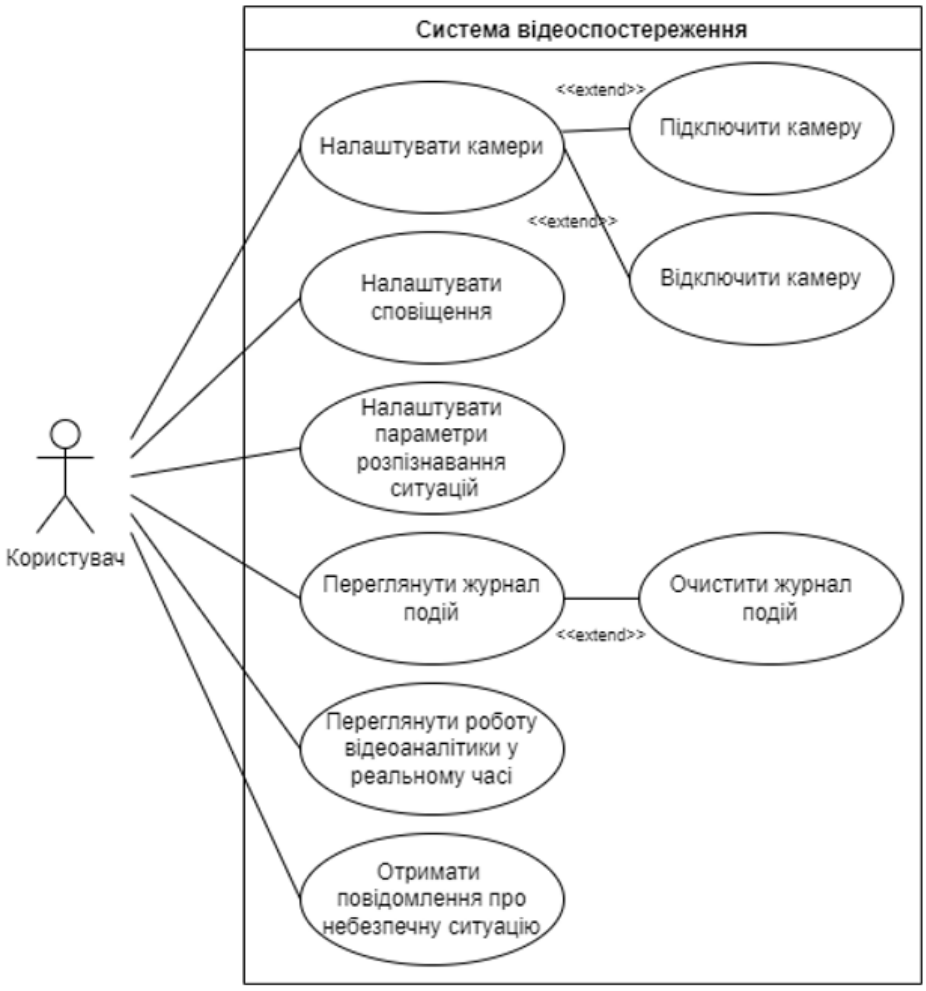

Рисунок 4.1 – Діаграма варіантів використання системи відеоспостереження

Наведемо докладний опис сценаріїв варіантів використання.

Сценарій для варіанту використання «Переглянути роботу відеоаналітики у реальному часі»:

Опис: користувач запускає програму, система починає транслювати на екран відео з камери з візуальним відображенням обмежувальних рамок та ідентифікаторів відстежених об’єктів.

Тригер: користувач запускає програму.

Основний сценарій:

1. Система відображає елементи інтерфейсу та починає транслювати на екран відео з візуальним відображенням обмежувальних рамок та ідентифікаторів відстежених об’єктів з першої у списку або останньої обраної користувачем камери.

Альтернативні сценарії:

1а. Користувач не додав жодної камери.

1а1. Система відображає повідомлення про необхідність додавання камер, користувач погоджується, перехід до сценарію «Налаштувати камеру».

1а2. Користувач відмовляється, система починає очікування додавання камер з розгорнутим повідомленням.

Сценарій для варіанту використання «Налаштувати камери»:

Опис: користувач переходить до списку камер, система відображає список.

Тригер: користувач взаємодіє з елементом інтерфейсу для управління камерами.

Основний сценарій:

1. Користувач натискає кнопку управління камерами, система відображає список камер з операційної системи пристрою.

Альтернативні сценарії:

1а. Не додано жодної камери.

1а1. Система відображає порожній список камер та переходить до циклічного опитування операційної системи на їх наявність.

Сценарій для варіанту використання «Підключити камеру»:

Опис: користувач обирає камеру, система додає камеру до списку оброблюваних.

Тригер: користувач взаємодіє з елементом інтерфейсу для додавання камер.

Передумова: виконано сценарій «Налаштувати камери».

Основний сценарій:

1. Користувач натискає кнопку додавання камери, система відображає список активних камер з операційної системи пристрою.

2. Користувач обирає камеру, система додає камеру до списку підключених.

Альтернативні сценарії:

2а. Користувач відмовляється від додавання камер.

2а1. Система ховає список камер з операційної системи пристрою.

Сценарій для варіанту використання «Відключити камеру»:

Опис: користувач обирає камеру, система видаляє камеру зі списку оброблюваних.

Тригер: користувач взаємодіє з елементом інтерфейсу для видалення камер.

Передумова: виконано сценарій «Налаштувати камери».

Основний сценарій:

1. Користувач натискає кнопку видалення камер, система відображає список підключених до системи камер.

2. Користувач обирає камеру, система видаляє камеру зі списку підключених.

Альтернативні сценарії:

2а. Користувач відмовляється від додавання камер.

2а1. Система ховає список камер з операційної системи пристрою.

2б. Користувач обирає активну у режимі перегляду камеру.

2б1. Система видаляє камеру зі списку підключених і змінює камеру у режимі перегляду на наступну в списку, або відображає запит на додавання камер за її відсутності.

Сценарій для варіанту використання «Переглянути журнал подій»:

Опис: користувач відкриває журнал, система відображає інформацію про критичні ситуації, зміни налаштувань, помилки у роботі та інші події в системі у вигляді списку записів формату «час виникнення події – назва типу події – короткий опис основних характеристик події – елемент інтерфейсу для перегляду повної інформації про подію».

Тригер: користувач запускає програму та переходить до журналу.

Основний сценарій:

1. Користувач натискає кнопку «Перегляд журналу подій», система відображає записи журналу.
2. Користувач обирає один з записів, система відображає докладну інформацію про запис.

Сценарій для варіанту використання «Очистити журнал подій»:

Опис: користувач відкриває журнал, система відображає записи про критичні ситуації, зміни налаштувань та інші події в системі.

Актори: користувач.

Передумова: виконано основний сценарій «Переглянути журнал подій».

Тригер: користувач натискає кнопку очищення журналу.

Основний сценарій:

1. Система запитує підтвердження дії, користувач підтверджує.
2. Система видаляє записи журналу.

Альтернативні сценарії:

1а. Користувач відмовляється від дії.

1а1. Система повертає інтерфейс у стан перегляду журналу.

Сценарій для варіанту використання «Налаштувати параметри розпізнавання критичних ситуацій»:

Опис: користувач відкриває налаштування (кількість об’єктів, їх місцезнаходження, час знаходження у зоні спостереження) та змінює їх, система зберігає.

Актори: користувач.

Тригер: користувач взаємодіє з елементом інтерфейсу для налаштувань розпізнавання ситуацій.

Основний сценарій:

1. Система відображає список налаштувань, користувач змінює обрані параметри.

2. Користувач натискає кнопку зберігання, система запитує підтвердження змін.

3. Користувач підтверджує, система зберігає зміни та повертає інтерфейс до режиму перегляду в реальному часі.

Альтернативні сценарії:

3а. Користувач відмовляється від дії.

3а1. Система видаляє зміни та повертає інтерфейс до режиму перегляду в реальному часі.

Сценарій для варіанту використання «Налаштувати сповіщення»:

Опис: користувач відкриває налаштування сповіщень (ідентифікатор аккаунту в мессенджері, адресу електронної пошти) та змінює їх, система зберігає.

Тригер: користувач взаємодіє з елементом інтерфейсу для налаштувань сповіщень.

Основний сценарій:

1. Система відображає список полів для вводу даних, користувач змінює обрані дані.

2. Користувач натискає кнопку повернення, система запитує підтвердження змін.

3. Користувач підтверджує, система зберігає зміни та повертає інтерфейс до режиму перегляду в реальному часі.

Альтернативні сценарії:

3а. Користувач відмовляється від дії.

3а1. Система видаляє зміни та повертає інтерфейс до режиму перегляду в реальному часі.

Сценарій для варіанту використання «Отримати повідомлення про небезпечну ситуацію»:

Опис: система розпізнає ситуацію та надає користувачу повідомлення про її виникнення зі звітом у форматі «час та дата – тип ситуації – класи задіяних об’єктів – ідентифікатори задіяних об’єктів – додаткова інформація для ситуації» та знімком з камери з виділеними кольором обмежувальними рамками та ідентифікаторами об’єктів; користувач отримує повідомлення, переглядає звіт та закриває його.

Тригер: виникнення небезпечної з точки зору системи ситуації у зоні огляду підключеної відеокамери.

Передумова: додано хоча б одну камеру, отримання повідомлень не вимкнено, користувач додав налаштування для параметрів розпізнавання ситуації, що виникла.

Основний сценарій:

1. У зоні огляду камери відбувається небезпечна ситуація певного типу, система розпізнає ситуацію та надсилає користувачу повідомлення зі звітом та знімком з камери у момент виникнення з накладеними обмежувальними рамками та ідентифікаторами відстежених об’єктів. Система також надсилає звіт зі знімком на електрону пошту користувача та у мессенджер.
2. Користувач отримує повідомлення, переглядає звіт та закриває його, система згортає повідомлення.

Альтернативні сценарії:

1а. Користувач не додав налаштувань повідомлень через електрону пошту та/або мессенджер, або запит на відправлення повідомлення повертає помилку мережі.

1а1. Система не надсилає повідомлення та додає запис про помилку у журнал подій.

2а. Користувач не реагує на повідомлення протягом 3 хвилин, при цьому вікно програми неактивно.

2а1. Система надсилає нотифікацію до ОС пристрою, перехід до п. 2.

4.2 Опис нефункціональних вимог

4.2.1 Вимоги до продуктивності

Вимоги до продуктивності:

1. Максимальний час відгуку програми на взаємодію з елементом інтерфейсу – не більше 0.5 секунди;

2. Максимальний час запуску програми – не більше 5 секунд.

3. Час обробки відео обумовлюється апаратним забезпеченням, але не має бути нижчим за 0.2 секунди на один кадр.

4. Час від моменту виникнення небезпечної ситуації до моменту відправлення сповіщення не повинен перевищувати 2.5 секунди.

4.2.2 Вимоги до надійності

Вимоги до стійкості програми до збоїв у роботі описуються такими правилами:

1. Програма повинна залишити всі ресурси операційної системи пристрою у коректному стані.

2. При наступному запуску програма повинна коректно запуститися і не містити жодних змін у результаті збою.

4.2.3 Вимоги до умов експлуатації

Програма не потребує спеціального навчання використанню та має убезпечити користувача від невалідних дій за допомогою інтерфейсу. Налаштування повинні бути зрозумілими та не вимагати від користувача спеціальних знань. За відсутності змін у налаштуваннях від користувача програма має використовувати валідні налаштування за замовчуванням. Запис відео має проводитись у будь-яких умовах, утім коректна робота трекінгу за особливих умов (відсутності достатнього освітлення тощо) не гарантується.

4.2.4 Вимоги до середовища функціонування

Вимоги до програмного забезпечення:

Для функціонування застосування на пристрої повинна бути встановлена операційна система Windows версії 10 або вище.

Вимоги до програмного забезпечення для коректної роботи пристроїв запису відео (драйверів) ідентичні до вимог операційної системи.

Вимоги до апаратного забезпечення:

Для функціонування застосування пристрій, на якому виконується запуск, повинен задовольняти наступним апаратним вимогам:

- розмір вільної вбудованої пам’яті – не менше 800 Мб;
- розмір вільної оперативної пам’яті – не менше 512 Мб;
- центральний процесор із характеристиками Intel Celeron J4005 або вище та тактовою частотою не менше 2.0 ГГц;
- відеокарта з підтримкою Nvidia CUDA та характеристиками GeForce GT 430 або вище, обсяг відеопам’яті не менше 512 Мб;

Система підтримує зовнішні камери з інтерфейсом USB 2.0 та вище, вбудовані камери ноутбуків та камери інших видів, що підтримуються операційною системою Windows 10.

4.3 Архітектура системи

Програмна система складається з таких основних компонентів: компонент роботи з камерами, компонент виявлення, компонент донавчання виявлення, компонент відстеження, компонент аналізу ситуацій, компонент сповіщення, компонент налаштувань. Компоненти виявлення, відстеження та донавчання виявлення складають підсистему трекінгу. Структура системи та зв’язки між компонентами наведені у вигляді діаграми компонентів [63] на рисунку 4.2.

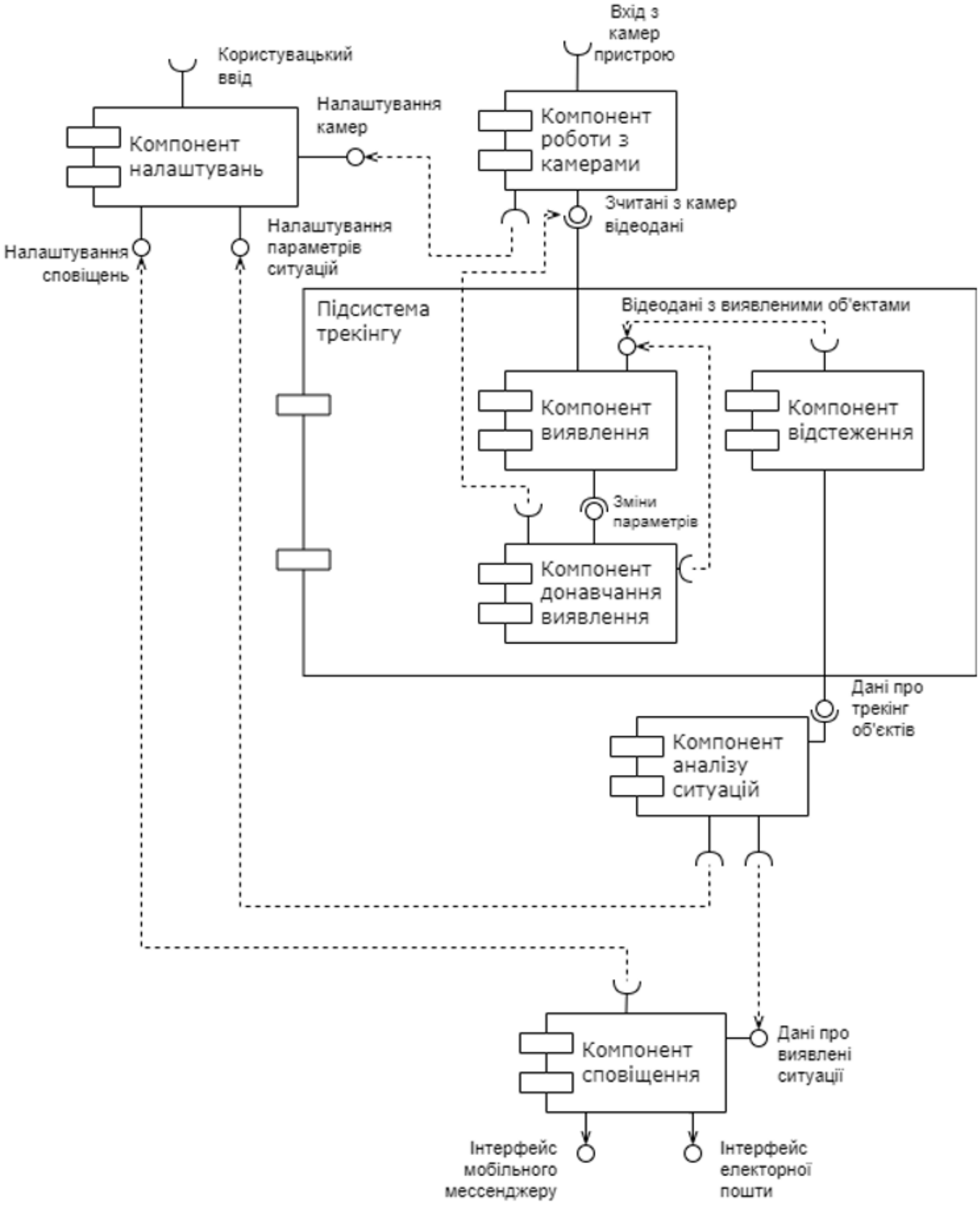

Рисунок 4.2 – Діаграма компонентів програмної системи

Наведемо опис кожного з високорівневих компонентів та підсистем:

- компонент роботи з камерами – відповідає за взаємодію з камерами через операційну систему, зчитує, обробляє та передає відеодані у підсистему трекінгу;
- підсистема трекінгу – включає компоненти виявлення, донавчання виявлення та відстеження, повністю інкапсулює функціонал трекінгу, отримує необроблені відеодані, передає результати трекінгу;
- компонент виявлення – відповідає за виявлення об’єктів, отримує необроблені відеодані, може отримувати зміни параметрів виявлення, передає оброблені відеодані з результатами виявлення об’єктів;
- компонент донавчання виявлення – відповідає за навчання моделі виявлення об’єктів на кінцевому пристрої користувача, отримує необроблені відеодані та відеодані з результатами виявлення об’єктів, передає зміни параметрів виявлення;
- компонент відстеження – виконує відстеження об’єктів, отримує відеодані з результатами виявлення об’єктів, передає результати відстеження об’єктів, що є остаточними результатами трекінгу;
- компонент аналізу ситуацій – визначає наявність небезпечних ситуацій з отриманих даних трекінгу та налаштувань і передає дані про розпізнані ситуації;
- компонент сповіщення отримує дані про розпізнані ситуації та налаштування сповіщень і на базі них відправляє повідомлення у сервіс електроної пошти та мобільний мессенджер;
- компонент налаштувань контролює налаштування системи, які отримує з користувацького вводу, валідує, зберігає та передає відповідним компонентам.

4.4 Детальне проектування логічного представлення системи

Наведемо деталізоване представлення кожного з компонентів високорівневої архітектури системи, наведеної раніше у підрозділі 4.3. Для демонстрації класів та зв’язків між ними використаємо діаграми програмних класів.

Діаграму програмних класів для компоненту роботи з камерами наведено на рисунку 4.3.

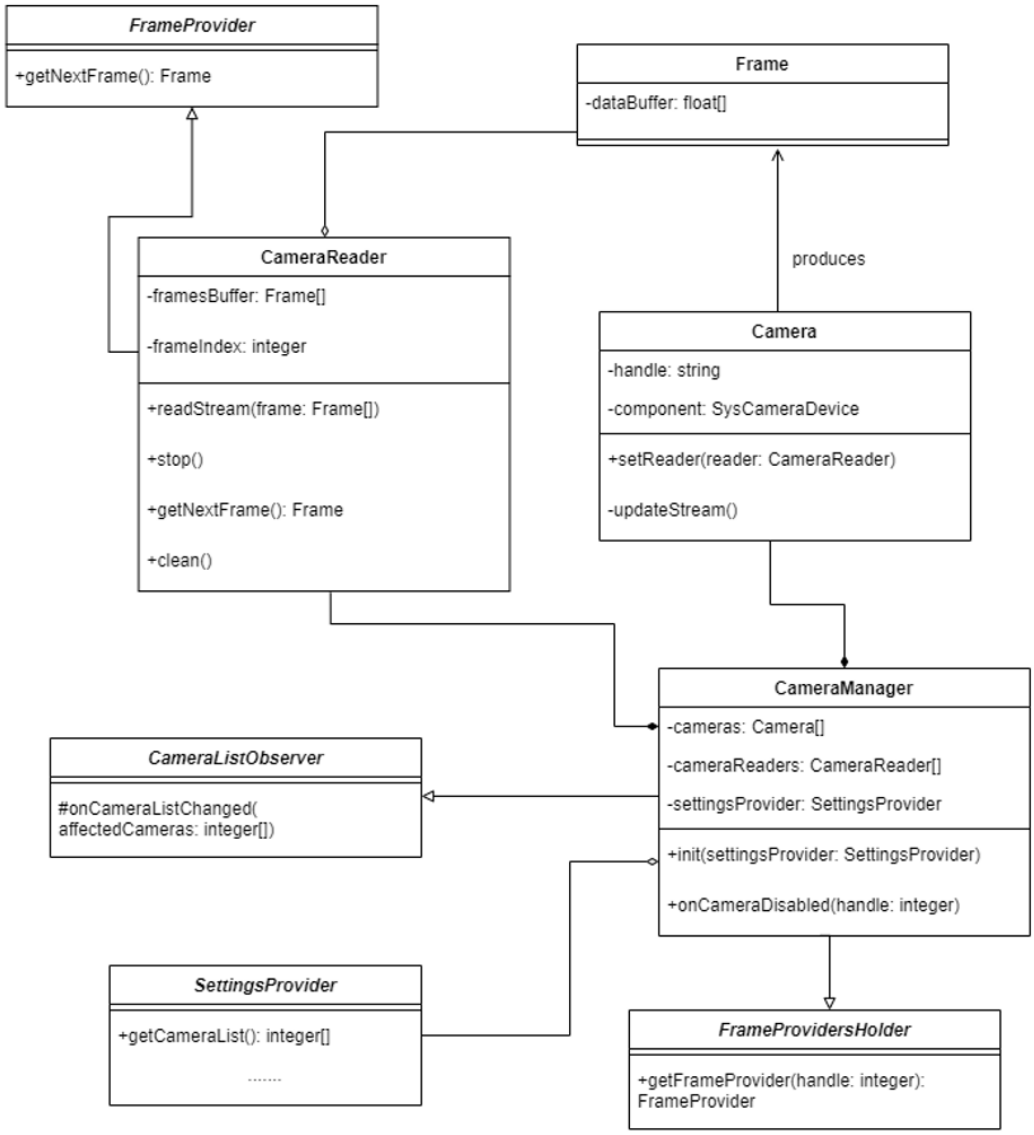

Рисунок 4.3 – Діаграма програмних класів для компоненту роботи з камерами

Функцію керування усім компонентом виконує клас CameraManager. Він містить масив елементів класу Camera, що є програмною обгорткою для вбудованого класу камери, та масив елементів класу CameraReader, що виконує під’єднання до вихідного потоку даних камери та перетворює необроблені дані потоку в об’єкти класу Frame. CameraReader також кешує кадри у кільцевому буфері та повертає наступний кадр за запитом через метод getNextFrame. Для відповідності принципу розділення

інтерфейсів SOLID [64] залежні від кадрів у CameraReader класи користуються інтерфейсом FrameProvider, якій він реалізує.

CameraManager отримує актуальний масив дескрипторів камер при ініціалізації з об’єкту SettingsProvider, та наслідує інтерфейс CameraListObserver (у відповідності до паттерну проектування GoF «Спостерігач» [65]), що дозволяє маніпулювати активними камерами у відповідності до змін налаштувань у відповідь на зворотні виклики з компоненту налаштувань. CameraManager також наслідує клас FrameProvidersHolder, що дозволяє виділяти необхідні іншим компонентам об’єкти CameraReader, не порушуючи принцип розділення інтерфейсів.

Розглянемо спільну діаграму програмних класів для компонентів виявлення та відстеження (рис. 4.4).

Спочатку розглянемо частину, що відноситься до компоненту виявлення. Клас управління виявленням DetectionManager наслідує інтерфейс CameraListObserver для реагування на зміни у списку камер у налаштуваннях (аналогічно CameraManager). Він також отримує при ініціалізації об’єкт класу FrameProvidersHolder, від якого отримує доступ до об’єктів FrameProvider для кожної з камер. Сам масив дескрипторів камер також ініціалізується за допомогою об’єкту SettingsProvider та оновлюється через інтерфейс CameraListObserver.

DetectionManager створює масив об’єктів ObjectDetector (відповідно принципам «Створювач» та «Інформаційний експерт» GRASP [66]), що відповідають за процес виявлення для кожної камери. ObjectDetector отримує при ініціалізації відповідний екземпляр FrameProvider для отримання кадрів та зчитує кадри по готовності до обробки, використовуючи можливості буферізації, реалізовані у компоненті роботи з камерами. У випадку недоступності кадру ObjectDetector переходить у режим очікування з опитуванням FrameProvider. Обробка кадру включає подання його на вхід нейронній мережі YOLO (дескриптор для роботи з якою міститься у кожному екземплярі ObjectDetector, а синхронізація використання реалізується за допомогою

вбудованого механізму семафорів) та перетворення результатів у формат ObjectTrackingData, що містить всю необхідну інформацію про виявлені об’єкти.

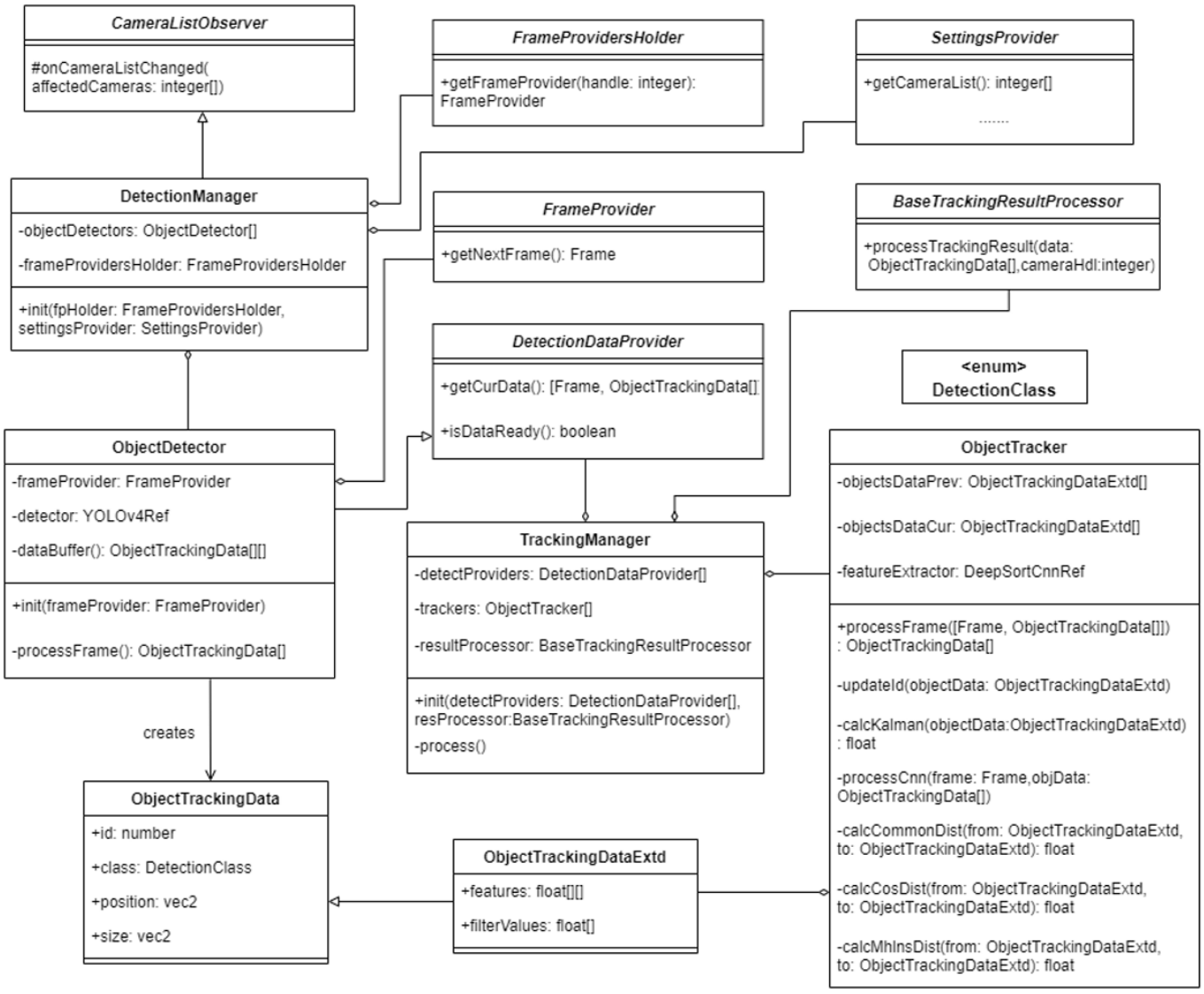

Рисунок 4.4 – Діаграма програмних класів компонентів виявлення та відстеження

ObjectDetector також наслідує клас DetectionDataProvider, що надає інтерфейс для отримання даних трекінгу для поточного кадру та перевірки готовності цих даних.

Далі за ходом виконання алгоритму переходимо до компоненту відстеження, де у дію вступає клас TrackingManager, що керує відстеженням об’єктів. Він, по аналогії з DetectionManager та ObjectDetector, створює масив класів ObjectTracker – по

екземпляру для кожної камери. При ініціалізації TrackingManager отримує масив об’єктів DetectionDataProvider (що є класом, від якого наслідує ObjectDetector) та об’єктів BaseTrackingResultProcessor, який використовує для делегування подальшої обробки результатів трекінгу.

Клас ObjectTracker містить логіку усього алгоритму відстеження (що відповідає принципам високої зв’язності та низького зацеплення GRASP [66]), включаючи обробку нейронною мережею для витягнення глибинних ознак зовнішнього вигляду об’єктів, фільтр Калмана, розрахунок комбінованої відстані між об’єктами та присвоювання ідентифікаторів. Порядок звертання до нейронної мережі синхронізовано так само, як і до мережі виявлення. Отримані дані трекінгу зберігаються для поточного та попереднього кадрів у масиві об’єктів ObjectTrackingDataExtd – розширенні класу ObjectTrackingData, що включає глибинні ознаки для об’єкту та параметри фільтру Калмана.

Порядок обробки кадрів при відстеженні відрізняється від порядку при виявленні – керуючий клас TrackingManager опитує кожен з об’єктів DetectionDataProvider про готовність кадру та за готовності викликає метод processFrame відповідного об’єкту ObjectTracker, блокуючи виконання потоку для даних поточної камери до отримання значення, що повертається. Повернене значення (масив об’єктів ObjectTrackingData) передається до інтерфейсу обробника результатів трекінгу BaseTrackingResultProcessor.

Компонент донавчання реалізує маніпулювання файлами датасетів та параметрів мереж, логіку навчання нейронної мережі, а також механизми збору та підрахунку даних для навчання та таймер завершення цього збору. Більшу частину компоненту реалізовано у вигляді сценаріїв на мові програмування Python у процедурному стилі, тому до діаграм програмних класів його не включено.

Далі розглянемо компонент аналізу ситуацій, діаграма програмних класів для якого наведена на рисунку 4.5.

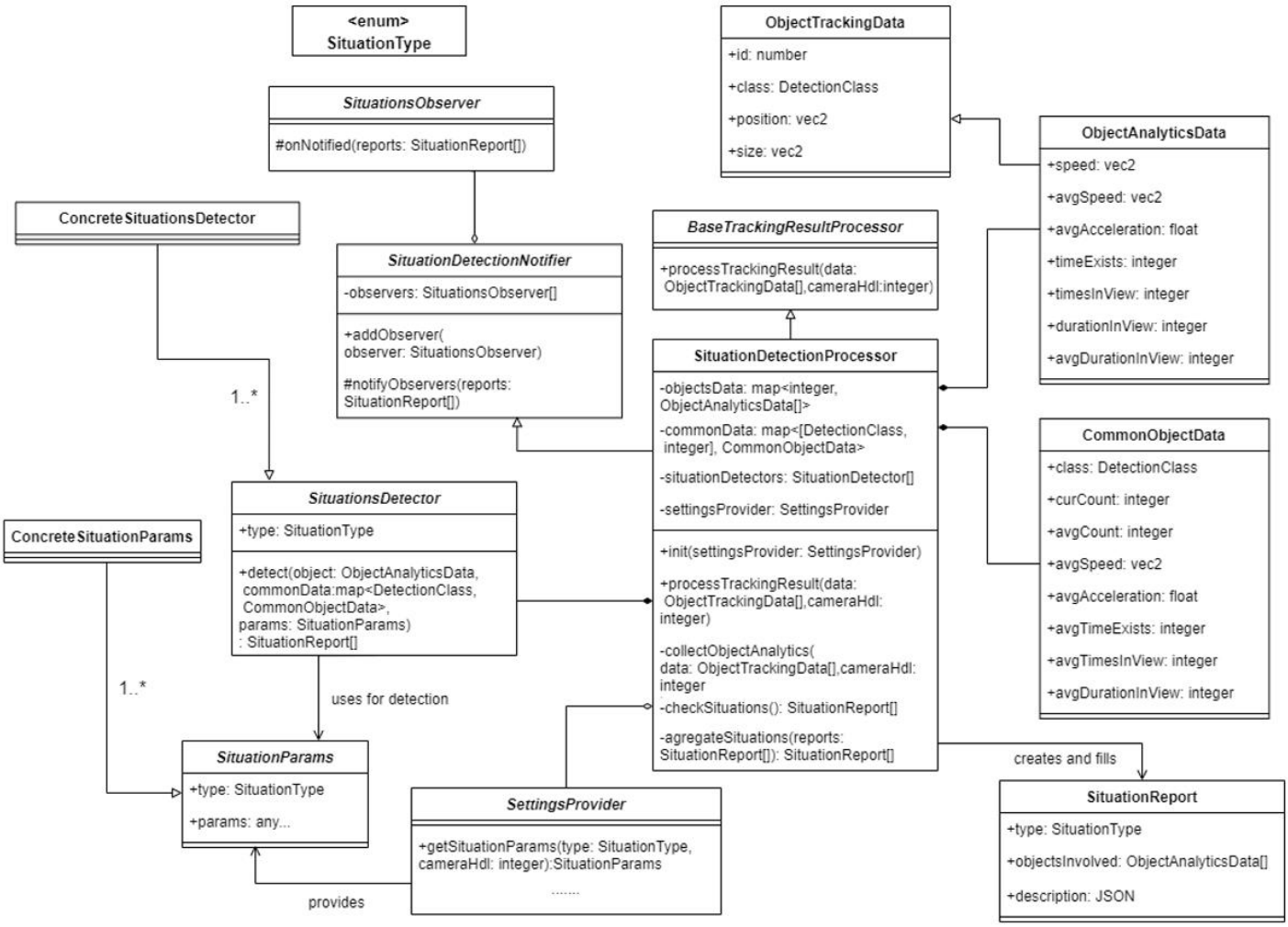

Рисунок 4.5 – Діаграма програмних класів компоненту аналізу ситуацій

Як вказано на діаграмі, центральну логіку компоненту містить клас SituationDetectionProcessor, що наслідує клас BaseTrackingResultProcessor. Він приймає при ініціалізації об’єкт SettingsProvider, від якого отримує налаштування розпізнанавання ситуацій. При виклику методу обробки результатів трегінгу він виконує аналіз інформації та збирає результати у форматі об’єктів класів ObjectAnalyticsData (клас-нащадок ObjectTrackingData) та CommonAnalyticsData, що містять дані про об’єкти трекінгу, наведені у підрозділі 4.4.1. Після збирання інформації SituationDetectionProcessor делегує розпізнавання ситуацій класам, які позначені на діаграмі як ConcreteSituationsDetector. Вони наслідують абстрактний клас SituationsDetector та відповідають за розпізнавання кожний своєї ситуації.

Кожний екземпляр ConcreteSituationsDetector отримує також об’єкт класу ConcreteSituationParams, похідного від класу налаштувань SituationParams, відповідний налаштуванням розпізнавання конкретної ситуації. Після розпізнавання ситуацій SituationDetectionProcessor створює звіт у вигляді класу SituationReport, агрегує дані про пов’язані ситуації та сповіщає про них інші класи через механізм шаблону «Спостерігач», реалізований у класах SituationObserver та SituationDetectionNotifier.

На рисунку 4.6 наведено діаграму класів для компоненту сповіщення.

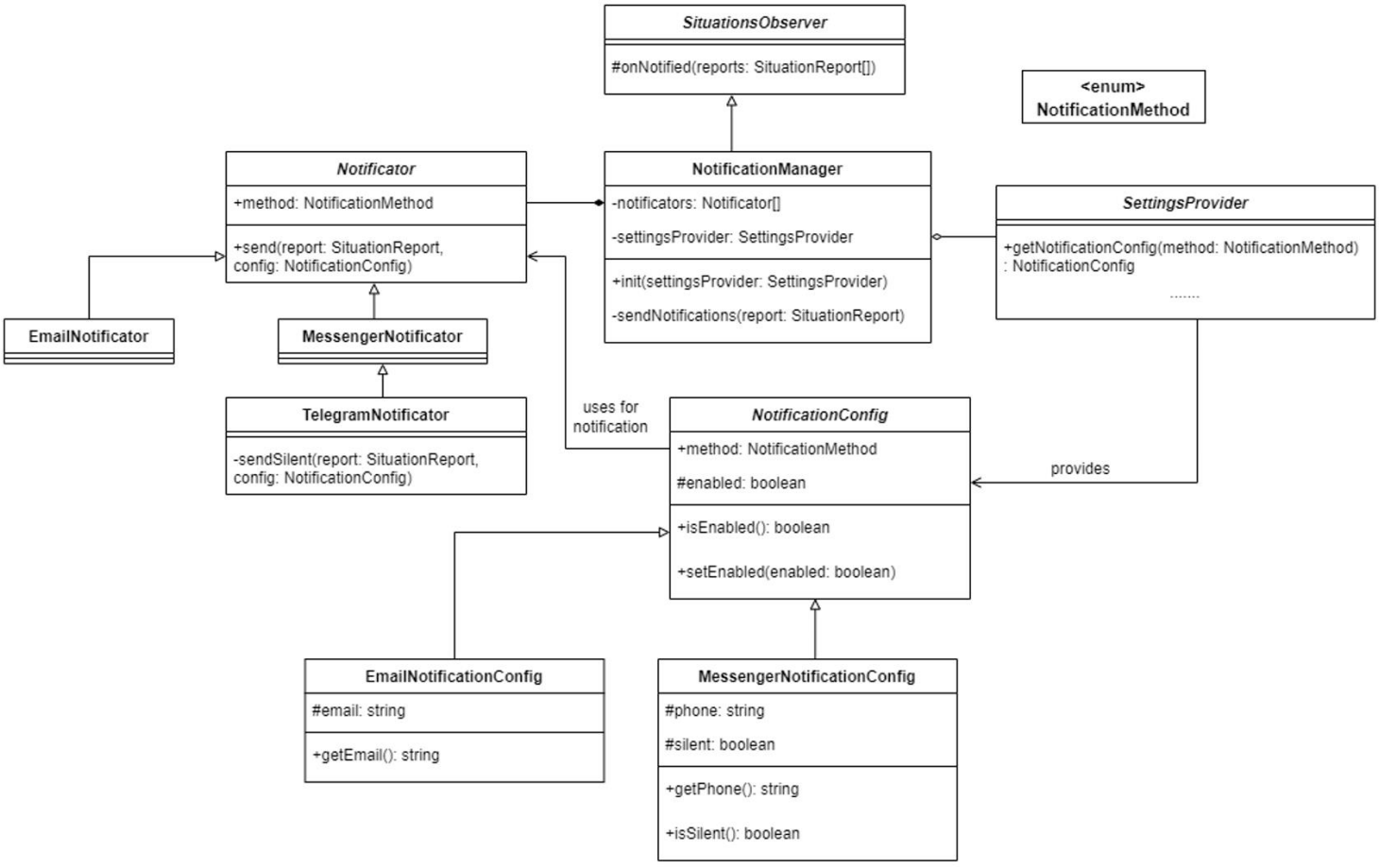

Рисунок 4.6 – Діаграма програмних класів компоненту сповіщення

Як бачимо на діаграмі, компонентом сповіщення керує клас NotificationManager, який наслідує клас спостерігача за ситуаціями SituationObserver (що також є частиною

реалізації шаблону проектування «Спостерігач»), через інтерфейс якого отримує від класу SituationDetectionNotifier повідомлення про виникнення ситуацій. При отриманні повідомлення про ситуацію NotificationManager викликає методи відправки сповіщень у реалізаціях абстрактного класу Notificator – EmailNotificator для електроної пошти та TelegramNotificator для мессенджеру Telegram, яким передає отримані як параметр через метод onNotified класу SituationObserver звіти з інформацією про ситуації. При відправлені повідомлень екземпляри використовують відповідні конфігурації, які отримують через SettingsProvider у вигляді відповідних кожному методу відправлення реалізацій класу NotificationConfig.

Діаграму класів для компоненту налаштувань наведено на рисунку 4.7.

Налаштуваннями керує клас SettingsMenu, що наслідує клас SettingsProvider, який було використано для доступу до налаштувань з більшості інших компонентів. Клас SettingsMenu містить масив екземплярів SettingsHolder – абстрактного класу, нащадки якого обробляють певні налаштування – камер (CameraSettingsHolder), ситуацій (SituationSettingsHolder) або повідомлень (NotificationSettingsHolder). Реалізації класу SettingsHolder містять логіку зберігання, оновлення з інтерфейсу користувача, завантаження та скидання до початкових значень тих налаштувань, які вони контролюють. Для кожного типу налаштувань використовується також власна реалізація абстрактного класу Config, у якій безпосередньо зберігаються відповідні значення налаштувань (конкретні реалізації – NotificationConfig, SituationParams, CamerasConfig). Налаштування кожного типу містять власну структуру – параметри ситуацій поділяються на класи для кожної конкретної ситуації, конфігурації повідомлень – на окремі класи для повідомлень на пошту та через мессенджер. Окрім того CameraSettingsHolder наслідує клас CameraListChangedNotifier, який є частиною ще однієї реалізації шаблону «Спостерігач» для подій змін списку камер, які відстежують раніше описані класи за допомогою наслідування від CameraListObserver.

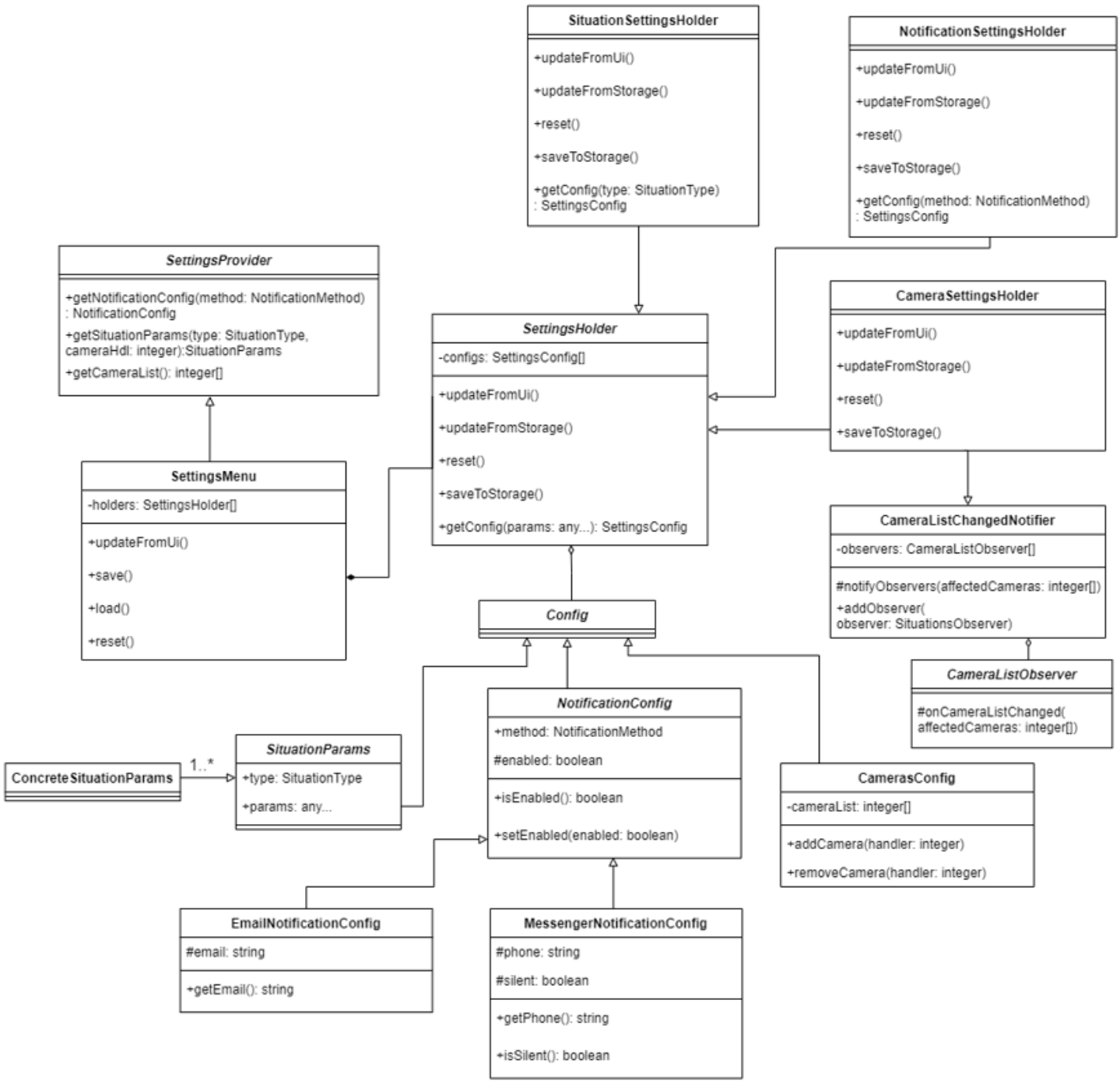

Рисунок 4.7 – Діаграма програмних класів компоненту налаштувань

Наведені компоненти містять незалежну від реалізації один одного структуру та спілкуються за допомогою виділених інтерфейсів. Ініціалізація керуючих класів компонентів відбувається у функції запуску програми.

4.5 Висновки до розділу

У розділі виконано опис проектування системи на всіх етапах – від формулювання вимог до створення архітектури. Створено опис функціональних вимог системи у вигляді сценаріїв варіантів використання, описано нефункціональні вимоги, спроектовано високорівневу архітектури системи у вигляді компонентів та описано деталі проектування кожного з них.

5 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

5.1 Опис програмних технологій

Програмну систему створено на Python [67] – інтерпретованій об'єктно-орієнтованій мові програмування. Для розробки використано версію Python 3.9.9.

Для роботи з елементами машинного навчання у системі застосовано PyTorch [68] – бібліотеку машинного навчання з відкритим вихідним кодом, засновану на бібліотеці Torch, що застосовується у таких сферах, як комп'ютерний зір та обробка природної мови. Це безкоштовне програмне забезпечення з відкритим вихідним кодом, випущене під модифікованою ліцензією BSD.

Для створення інтерфейсу користувача використано PyQt5 [69] – плагін для зв'язування кросплатформеного набору інструментів GUI Qt з мовою програмування Python. PyQt - безкоштовне програмне забезпечення, доступне для некомерційного використання за умовами ліцензії GNU General Public License (GPLv3).

NumPy - бібліотека для мови програмування Python, яка додає підтримку великих багатовимірних масивів і матриць, а також містить велику колекцією математичних функцій високого рівня для роботи з ними. У програмній системі NumPy використано для роботи з багатовимірними масивами даних при навчанні нейронної мережі. Numpy може бути вільно використана у будь-яких проектах згідно з модифікованою ліцензією BSD [70].

Для маніпулювання відеоданими та візуалізації результатів трекінгу об'єктів використано OpenCV (Open Source Computer Vision Library) [71] – бібліотеку програмних функцій для роботи з комп'ютерним зором у реальному часі. Бібліотека є кросплатформеною і безкоштовною для використання за ліцензією Apache 2.

Ще одна бібліотека, застосована для візуалізації результатів трекінгу об'єктів – Matplotlib [72]. Це набір засобів для створення графіків для мови програмування Python та її розширення NumPy. Має власну ліцензію Matplotlib license, що відповідає

правилам використання PSF (Python Software Foundation) та містить лише сумісний з ліцензією BSD код.

Інші застосовані технології – бібліотека telepot (ліцензія MIT) для роботи з API мессенджера Telegram, Pillow (ліцензія HPND) для відкривання, управління та збереження зображень різних форматів, SciPy (ліцензія BSD) – для використання деяких математичних функцій, PyInstaller (ліцензія GPL) – для створення вихідного exe-файлу.

5.2 Інструкція з встановлення

Програмна система розповсюджується у вигляді архіву, що містить файли даних програми та файл setup.bat, що запускає процес встановлення (рис 5.1).

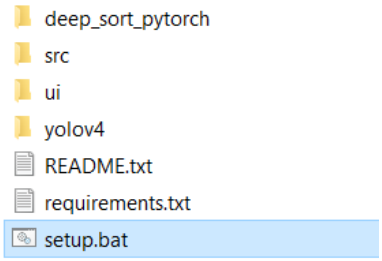

Рисунок 5.1 – Вміст архіву з програмною системою

Після розпакування архіву та запуску файлу setup.bat розпочнеться процес встановлення, протягом якого буде завантажено необхідні модулі та створено файл запуску програми. Під час встановлення буде відкрито вікно командної строки, у якому можна спостерігати за прогресом процесу (рис. 5.2) та вихідними повідомленнями про встановлені модулі і залежності. Встановлення системи вимагає наявності на комп’ютері середовища мови програмування Python. За його відсутності користувачу буде запропоновано його завантаження з офіційного магазину застосувань Microsoft.

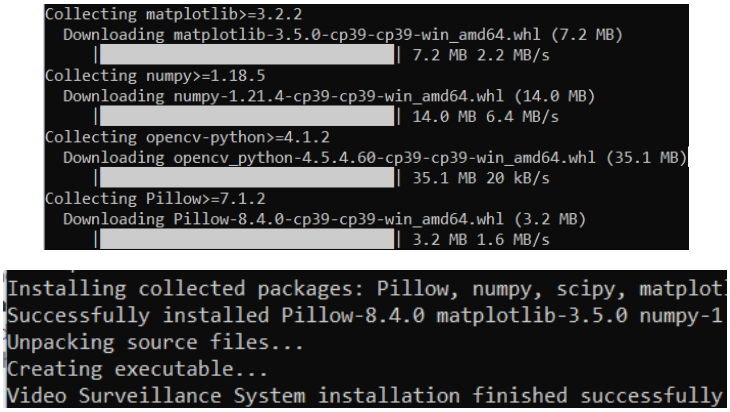

Рисунок 5.2 – Прогрес встановлення у вікні командної строки

По завершенню встановлення буде відображено вікно з повідомленням про його успішність (рис 5.3), а у робочій директорії з'явиться файл vss.exe (рис.5.4).

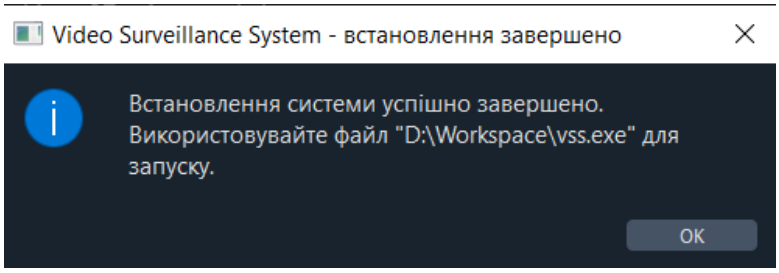

Рисунок 5.3 – Повідомлення про успішне встановлення системи

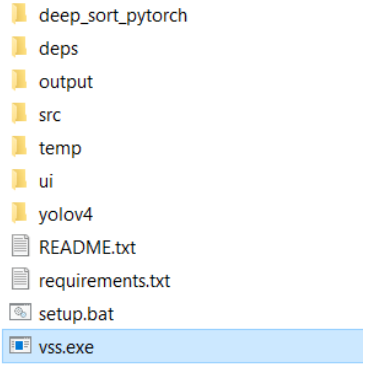

Рисунок 5.4 – Вміст директорії програми після встановлення

Подальший запуск роботи програми виконується за допомогою виконання стартового файлу vss.exe. Архів програми також включає файл Readme, який містить текстову частину цієї інструкції.

5.3 Інструкція з використання

Після запуску стартового файлу програмної системи vss.exe користувач потрапляє на головний екран (рис. 5.5), на якому наявна зона перегляду результатів роботи аналітики зі списком камер (із запитом на додавання камер при першому запуску), та кнопки переходу до різних типів налаштувань та журналу подій.

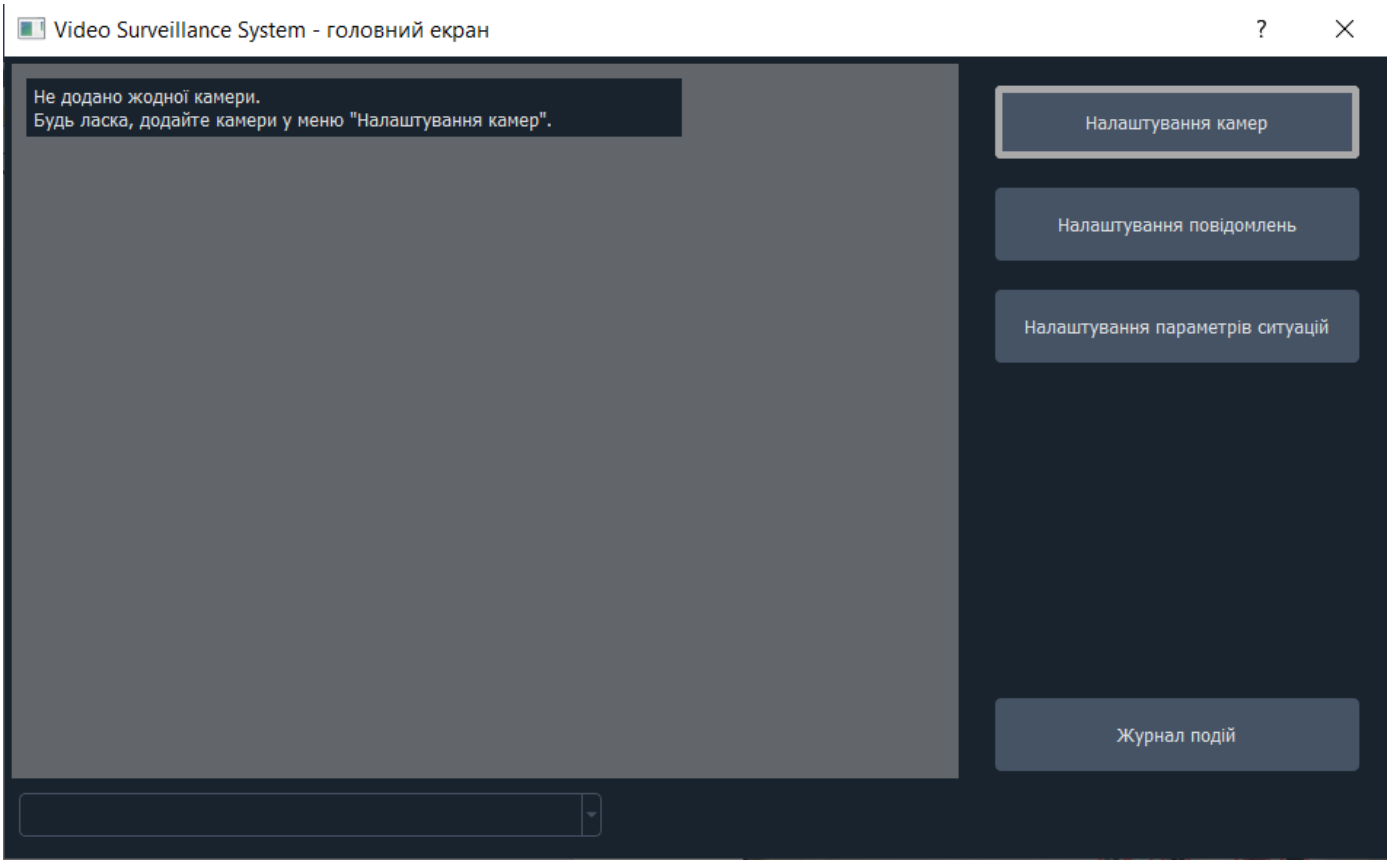

Рисунок 5.5 – Головний екран системи під час першого запуску

Додавання камер відбувається у відповідному меню, у яке можна перейти по натисканню кнопки «Налаштування камер» на головному екрані, після чого виконати вибір активних камер зі списку операційної системи (рис 5.6).

Після додавання камери стає можливим перегляд результатів роботи аналітики у реальному часі на головному екрані (рис 5.7).

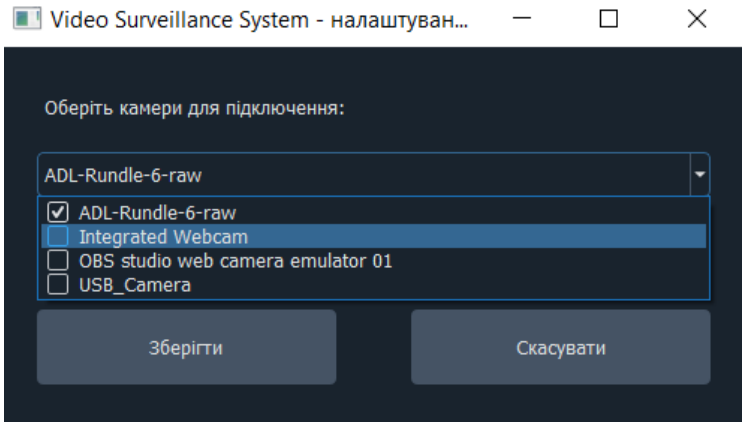

Рисунок 5.6 – Меню налаштування камер

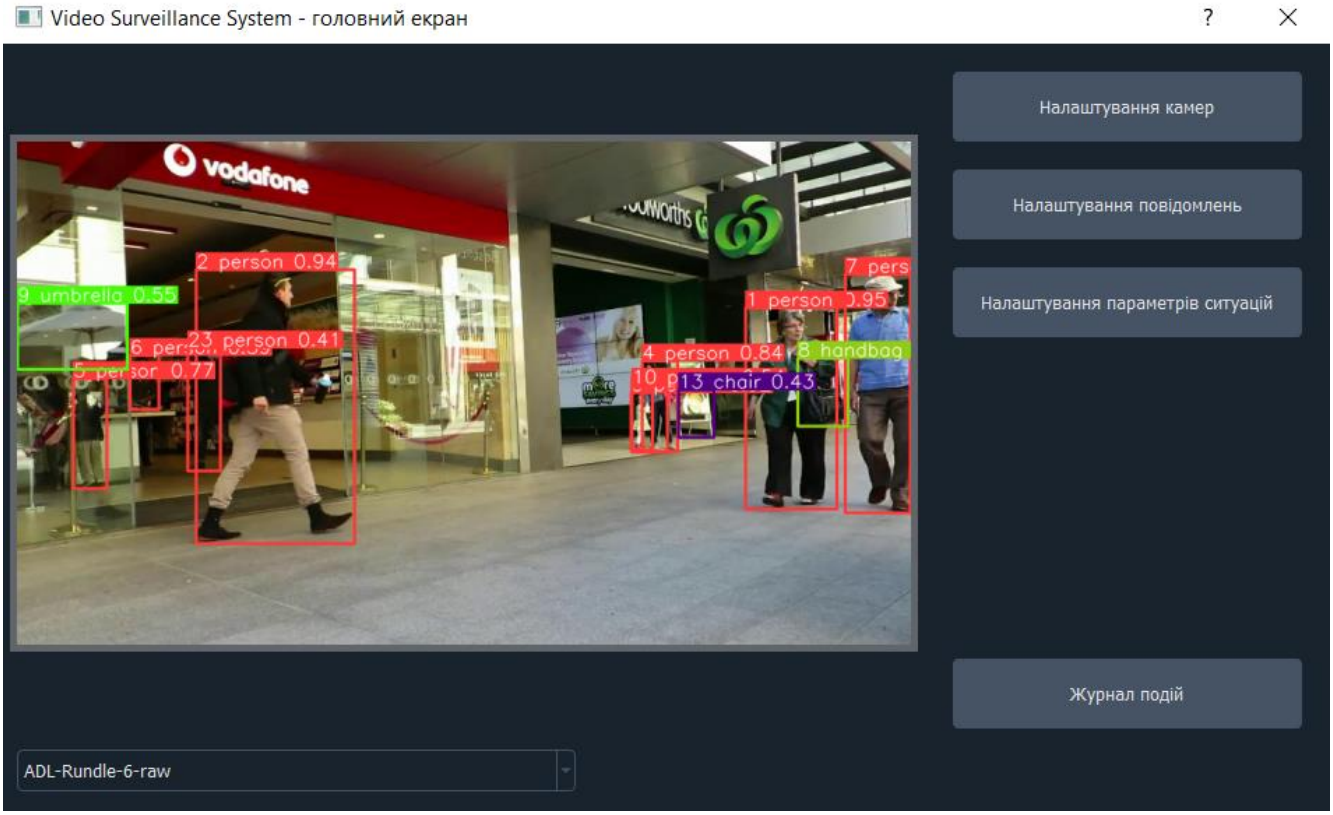

Рисунок 5.7 – Головний екран системи після додавання камери

Налаштування ситуацій відбувається у меню, що доступне по натисканню кнопки «Налаштування параметрів ситуацій» на головному екрані. Меню налаштувань містить списки, перемикачі та поля вводу параметрів для різних типів ситуацій та класів об’єктів (рис 5.8). Координати обмежувальних зон для ситуацій

типу «Знаходження у зоні» задаються шляхом виділення за допомогою маніпулятора (миші/тачпада) та візуально відображаються на зоні перегляду певної камери. Для зберігання виділеної зони необхідно натиснути кнопку «Додати заборонену зону».

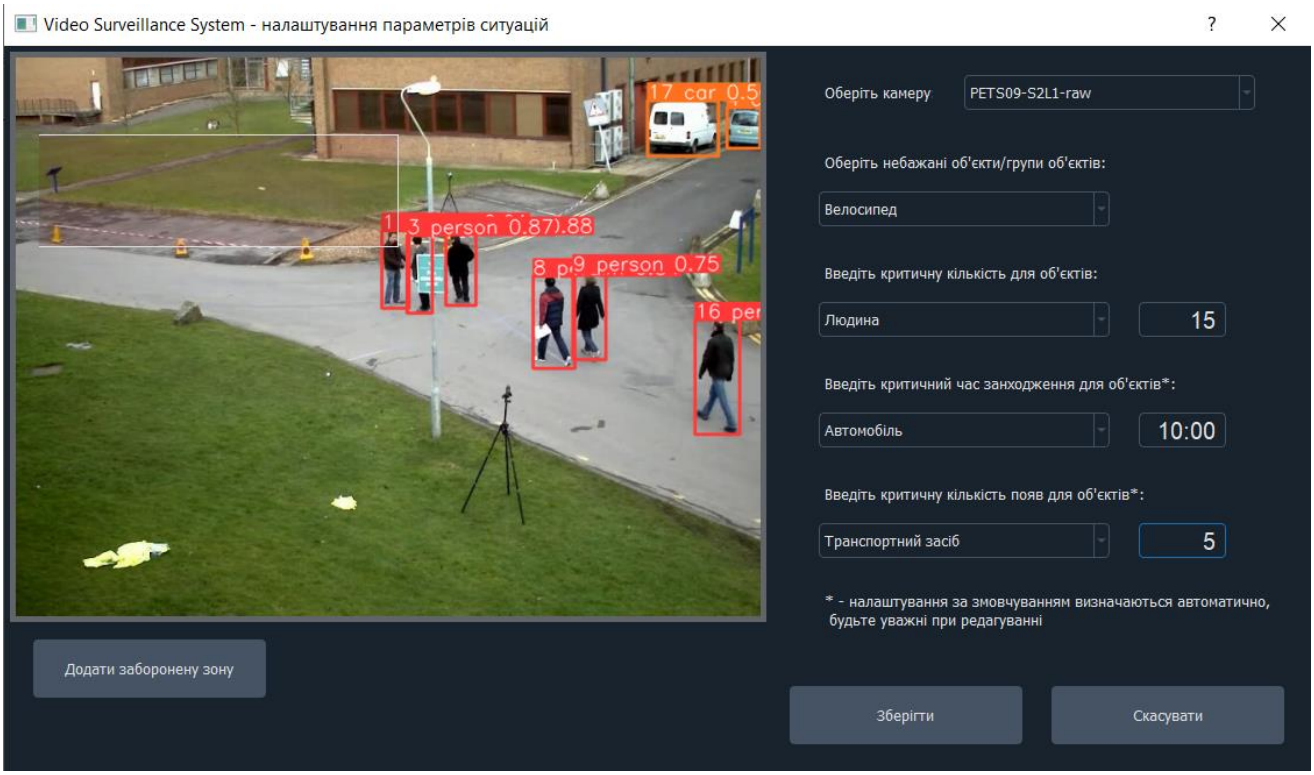

Рисунок 5.8 – Меню налаштування параметрів ситуацій

Налаштування сповіщень також відбувається шляхом переходу до відповідного меню за допомогою натискання кнопки на головному екрані. Меню містить поля вводу адреси електронної пошти та ідентифікатору акаунту Telegram, а також перемикачі беззвучного режиму у Telegram та у всій системі (рис. 5.9).

Журнал подій відображає записи про події (виникнення ситуацій, зміни налаштувань, помилки у роботі) в системі у форматі списку з елементами «час виникнення - тип – короткий опис» (рис. 5.10), що за натисненням відображають детальний опис відповідної події (рис. 5.11). Перехід до журналу виконується за натисненням кнопки «Журнал подій» у головному меню, кнопка «Очистити» у вікні журналу видаляє усі записи у ньому.

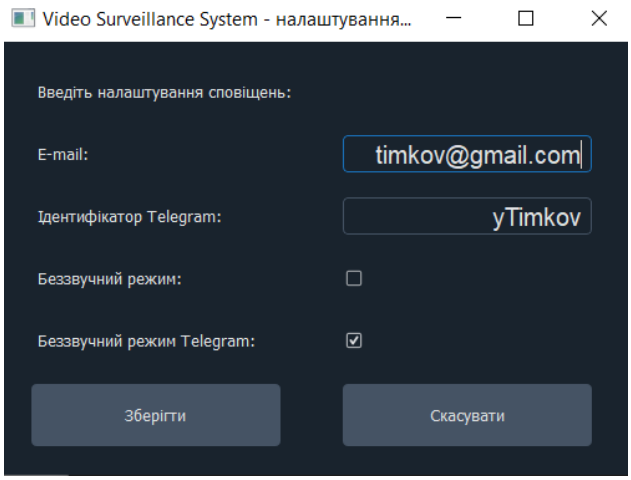

Рисунок 5.9 – Меню налаштування сповіщень

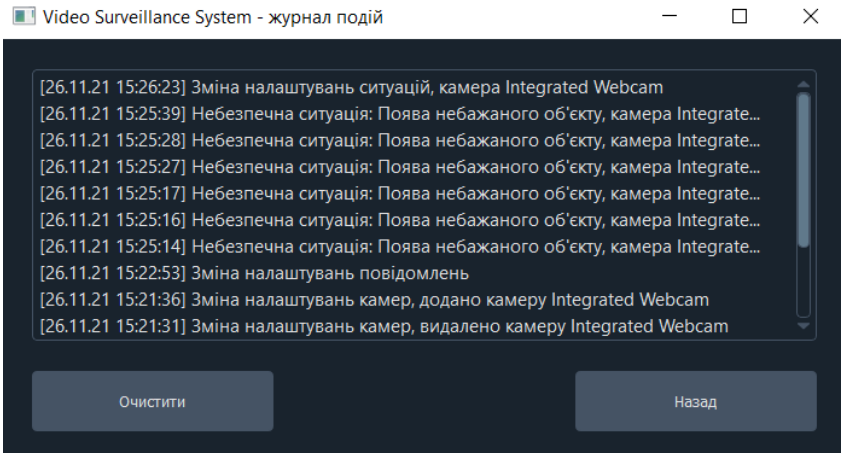

Рисунок 5.10 – Вікно журналу

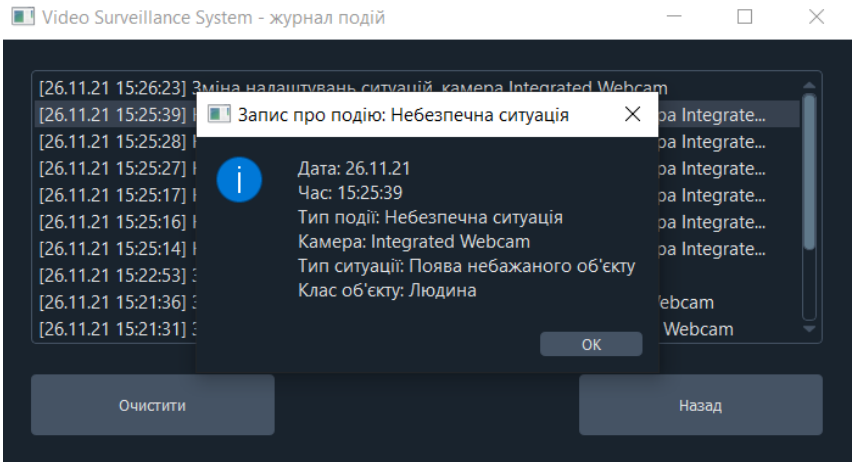

Рисунок 5.11 – Вікно журналу з детальним описом обраної події

У разі виникнення небезпечної ситуації система відправляє сповіщення на електрону пошту (приклад на рис. 5.12) та/або у мессенджер Telegram (приклад на рис. 5.13) відповідно налаштуванням. Система також відображає повідомлення через інтерфейс програми (рис. 5.14), якщо така функція не була відімкнена у налаштуваннях.

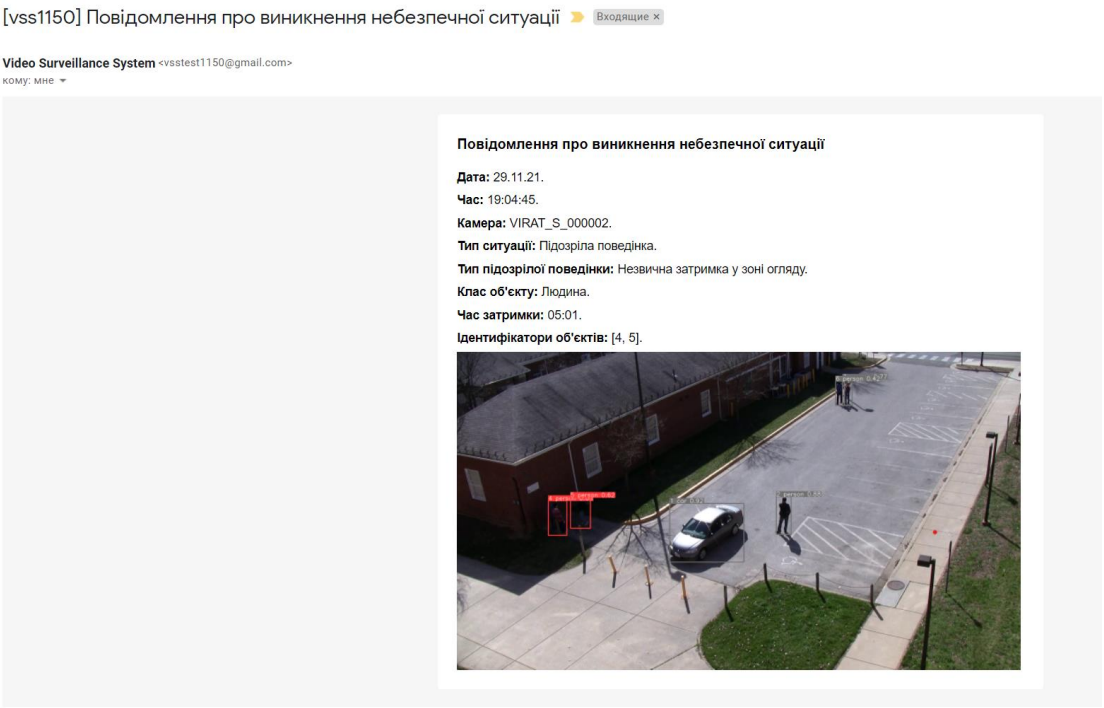

Рисунок 5.12 – Електронний лист з повідомленням про небезпечну ситуацію

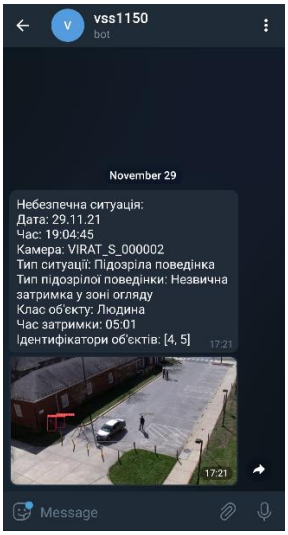

Рисунок 5.13 – Повідомлення про небезпечну ситуацію у мессенджері Telegram

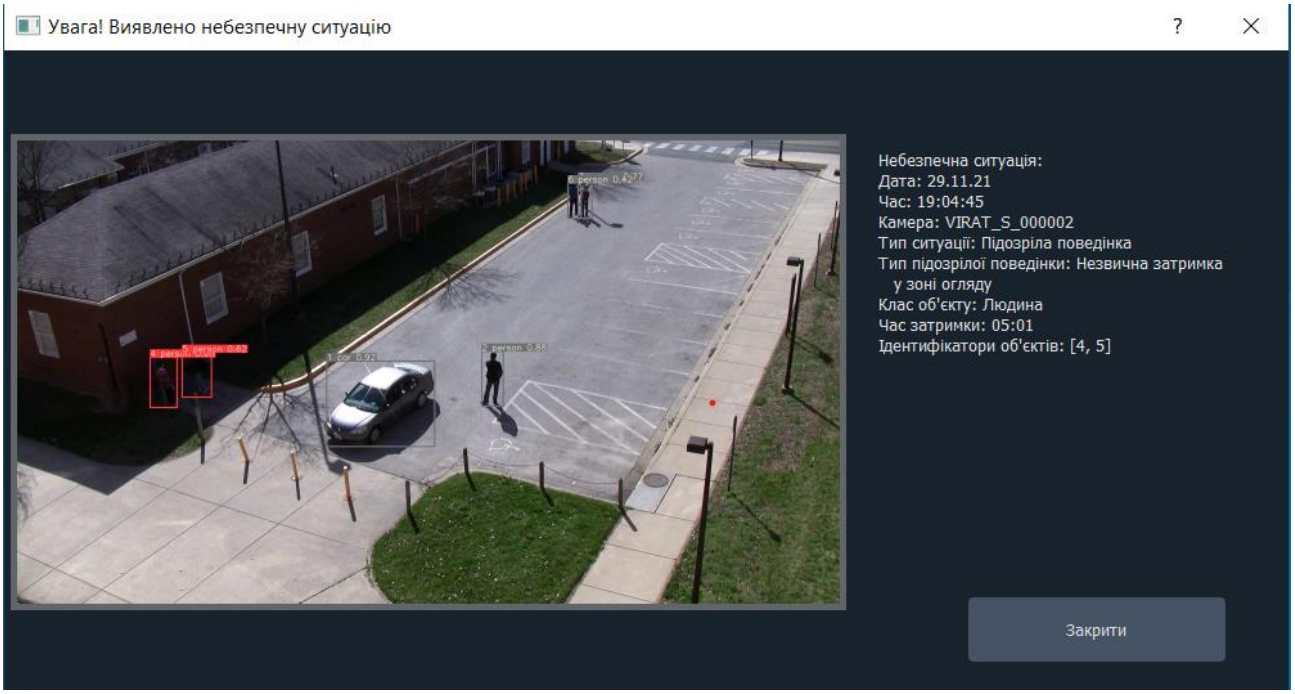

Рисунок 5.14 – Повідомлення про небезпечну ситуацію через інтерфейс програми

Система відправляє повідомлення на електрону пошту та у мессенджер Telegram у разі наявності даних у налаштуваннях сповіщень. Інформація у повідомленнях включає дату, час, тип ситуації, подробиці про об’єкти, що задіяні, та знімок з камери у момент виникнення, що разом з виділенням кольором рамок об’єктів та інформацією про їх ідентифікатори дає можливість візуально оцінити ситуацію самостійно.

Наведена у прикладах (рисунки 5.12 – 5.14) ситуація відноситься до типу підозрілої поведінки. Додаткова інформація у звіті про неї включає тип підозрілої поведінки (незвична затримка в зоні огляду), клас задіяних об’єктів (людина), список ідентифікаторів об’єктів (4, 5), та час затримки (05:02 секунди). Ситуацію було розпізнано за допомогою алгоритму, описаного у підрозділі 3.5.1. При обробці кожного кадру на першому етапі алгоритму оновлювалась та зберігалась інформація про об’єкти, що фігурують у ситуації, включаючи час їх знаходження у зоні огляду. На другому етапі алгоритму інформація перевірялася на перевищення критичних значень, виявлених автоматично або назначених користувачем. При обробці кадру, на якому відбулося перевищення критичного значення часу знаходження у зоні огляду

для об’єкту класу «людина» (встановленого користувачем у налаштуваннях як 5 хвилин) система виявила небезпечну ситуацію. При формуванні звітів система також виявила, що час виникнення перевищення критичного параметру майже не відрізняється для двох об’єктів, тому агрегувала ці випадки в один звіт.

5.4 Функціональне тестування

Для перевірки основного функціоналу системи наведено тестові сценарії, створені на базі описаних у підрозділі 4.1 варіантах використання системи та інших функціональних вимогах (табл. 5.1 – 5.6).

Таблиця 5.1 – Тестовий сценарій для перевірки функціоналу налаштування камер

Дія	Очікуваний результат	Отриманий результат
1. Користувач переходить на головний екран.	Система відображає кнопку налаштування камер.	Відповідає очікуваному.
2. Користувач натискає кнопку налаштування камер.	Система відображає список підключених до операційної системи камер.	
3. Користувач обирає камеру, яку бажає додати.	Система додає камеру до списку підключених.	
4. Користувач обирає камеру, яку бажає видалити.	Система видаляє камеру зі списку підключених.	
5. Користувач натискає кнопку повернення до головного меню.	Система зберігає налаштування та відображає оновлений список камер на головному екрані.	

Таблиця 5.2 – Тестовий сценарій для перевірки функціоналу налаштування сповіщень

Дія	Очікуваний результат	Отриманий результат
1. Користувач переходить на головний екран.	Система відображає кнопку налаштування сповіщень.	Відповідає очікуваному.
2. Користувач натискає кнопку налаштування сповіщень.	Система відображає список налаштувань сповіщень (email, Telegram ідентифікатор, беззвучний режим, беззвучний режим у Telegram).	

Продовження таблиці 5.2

Дія	Очікуваний результат	Отриманий результат
3. Користувач змінює значення та повертається на головний екран.	Система зберігає зміни	Відповідає очікуваному.
4. Користувач знов натискає кнопку налаштування сповіщень.	Система відображає список налаштувань сповіщень із внесеними раніше змінами.	

Таблиця 5.3 – Тестовий сценарій для перевірки функціоналу налаштування ситуацій

Дія	Очікуваний результат	Отриманий результат
1. Користувач переходить на головний екран.	Система відображає кнопку налаштування параметрів ситуацій.	Відповідає очікуваному.
2. Користувач натискає кнопку налаштування параметрів ситуацій.	Система відображає список налаштувань параметрів ситуацій (список небажаних об’єктів, критична кількість об’єктів, критичний час знаходження об’єктів та інші).	
3. Користувач змінює значення та повертається на головний екран.	Система зберігає зміни.	
4. Користувач знов натискає кнопку налаштування параметрів ситуацій.	Система відображає список налаштувань параметрів ситуацій із внесеними раніше змінами.	

Таблиця 5.4 – Тестовий сценарій для перевірки функціоналу перегляду журналу

Дія	Очікуваний результат	Отриманий результат
1. Користувач переходить на головний екран.	Система відображає кнопку перегляду журналу.	Відповідає очікуваному.
2. Користувач натискає кнопку перегляду журналу.	Система відображає список записів журналу подій, а також кнопку очистки журналу.	

Продовження таблиці 5.4

Дія	Очікуваний результат	Отриманий результат
3. Користувач натискає кнопку очистки журналу.	Система видаляє записи зі списку.	

Таблиця 5.5 – Тестовий сценарій для перевірки функціоналу перегляду відеоаналітики

Передумови	Дія	Очікуваний результат	Отриманий результат
До системи додано камеру.	1. Користувач переходить на головний екран.	Система відображає список підключених камер, а також вихідний потік з камери з обмежувальними рамками об’єктів та їх ідентифікаторами.	Відповідає очікуваному.
До системи додано більш, ніж одну камеру.	2. Користувач змінює активну камеру.	Система змінює камеру, з якої відображається вихідний потік з обмежувальними рамками об’єктів та їх ідентифікаторами, на обрану.	

Таблиця 5.6 – Тестовий сценарій для перевірки функціоналу розпізнавання небезпечних ситуацій та сповіщення про них

Передумови	Дія	Очікуваний результат	Отриманий результат
Користувач налаштував параметри сповіщення для відправки звітів на електронну пошту, а параметри розпізнавання ситуацій – на реагування на появу людини у зоні огляду. Беззвучний режим не активовано. У зоні огляду за певний час з'являється людина.	1. Користувач запустив систему.	Система виявляє наявність об’єкту класу, встановленого у налаштуваннях як небажаного, у зоні огляду. Система відображає на екрані повідомлення про ситуацію у вигляді її назви («поява небажаного об’єкту»), часу виникнення та ідентифікатору об’єкту. Та сама інформація додається до журналу подій та разом із знімком з камери відсилається на електрону пошту, вказану у налаштуваннях.	Відповідає очікуваному.

Як бачимо з результатів виконання тестових сценаріїв, робота основного функціоналу програмної системи відбувається згідно опису функціональних вимог та відповідає очікуваним результатам.

5.5 Висновки до розділу

У розділі наведено опис використаних програмних технологій та технічних рішень при реалізації системи. Створено інструкції для користувачів з встановлення програмної системи та її використання. Проведено функціональне тестування програмної системи, за результатами якого встановлено відповідність системи функціональним вимогам.

6 ВИПРОБУВАННЯ СИСТЕМИ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Оскільки метою роботи є підвищення точності розпізнавання небезпечних ситуацій, точність буде основним параметром випробування системи. Однак розпізнавання ситуацій включає етапи виявлення та трекінгу, для кожного з яких необхідно оцінити вплив внесених удосконалень та довести їх доцільність. Окрім того, необхідно протестувати швидкість роботи розпізнавання ситуацій для виявлення можливих негативних впливів внесених змін та перевірки відповідності вимогам до продуктивності.

6.1 Випробування точності виявлення об’єктів

Перший етап випробування - перевірка точності виявлення об’єктів. Оскільки удосконалення алгоритму виявлення включає донавчання нейронної мережі на нових даних, необхідно порівняти точність до та після його здійснення. Виконувати донавчання та подальше тестування є сенс лише на зображеннях, знятих з однієї (або кількох) камер, тому звичайні набори даних для навчання нейронних мереж застосовувати недоцільно. Було використано набори даних Multiple Object Tracking Benchmark (MOT) [73], що включають розмічені відеодані для змагань з оцінки алгоритмів відстеження, обравши для тестування ті відеозаписи, як зняті зі статичної камери та довжина яких дозволяє набрати достатню кількість даних для навчання. Зображення для навчання відібрано вручну з урахуванням особливостей випадкового відбору, який виконуватиме система.

В якості метрики тестування використано параметр AP50, що використовується також у тестуванні оригінальної версії YOLOv4 (на датасеті COCO) [47]. Метрика AP визначається як площа під кривою, яку створює двовимірний графік значень precision та recall зі збільшенням значення cutoff для оцінки впевненості мережі [74]. Значення precision визначається як відношення правильних позитивних виявлень до загальної

кількості виявлень, значення recall – відношення правильних позитивних виявлень до загальної кількості реальних позитивних випадків. Значення cutoff - поріг впевненості, у разі відсутності перевищення якого вихідним значенням нейронної мережі confidence score для об’єкту результат виявлення вважається хибнопозитивним. Для виявлення об’єктів правильність чи хибність визначається як перевищення метрикою IoU для об’єкту певного порогу – для AP50 це, відповідно назві, 0.5 (50%).

Результати оцінки метрики AP50 для кожного з тестових відео окремо та у середньому з використанням нейронної мережі до та після донавчання на даних з усіх відео наведено у таблиці 6.1. Порівняльну діаграму отриманих в результаті тестування значень метрики AP50 для кожного відео та у середньому до та після донавчання наведено на рисунку 6.1.

Таблиця 6.1 – Результати оцінки метрики AP50 до та після донавчання на нових даних

Назва відео	AP50 до донавчання, %	AP50 після донавчання, %	Різниця, %
MOT15 Venice-1	59.82	61.24	1.42
MOT15 PETS09-S2L1	58.22	63.96	5.74
MOT16-03	55.00	55.33	0.33
MOT16-09	62.24	62.47	0.23
MOT17-13-FRCNN	59.12	61.39	2.29
MOT20-02	55.91	61.76	5.85
Середнє значення	58.38	61.02	2.64

Як видно з таблиці, середнє значення AP50 до донавчання становить 58.38%, тоді як після – 61.02%, тобто збільшується на 2.64 відсотки. При цьому покращення результату наявне для усіх камер, а для деяких камер підвищення становить більше 5%. Для порівняння, значення AP50 оригінальної мережі YOLOv4 на датасеті COCO дорівнює 64.9% [47].

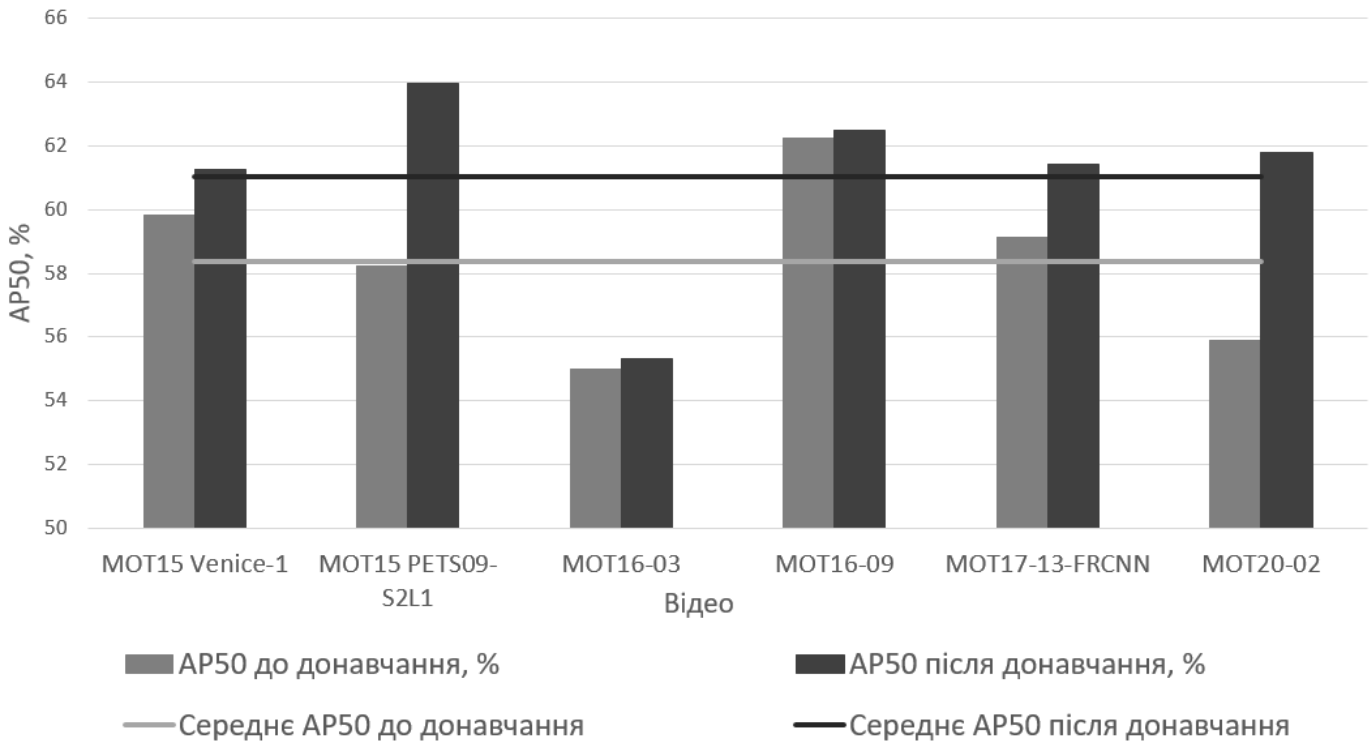

Рисунок 6.1 – Діаграма результатів тестування метрики AP50 до та після донавчання на нових даних

Такі показники демонструють, що ефект від донавчання на нових даних значно залежить від обраних даних, але у будь-якому випадку покращує наявні результати.

6.2 Випробування точності відстеження об’єктів

Наступний етап випробування – перевірка точності відстеження об’єктів. Оскільки на точність відстеження впливають як зміни у моделі виявлення об’єктів, так і у методі відстеження, необхідно виконати тестування як разом з донавчанням виявлення, так і лише зі змінами у методі відстеження. Для перевірки точності застосовано той самий датасет, що й для тестування оригінальних методів SORT та DeepSORT – MOT16 [73]. З усього датасету використано лише відео зі статичних камер, що найбільше схожі на камери відеоспостереження. Модель виявлення буде містити зміни у результаті донавчання на усіх відео, наведених при тестуванні

виявлення у табл. 6.1, оскільки зайві відео відповідатимуть реальній ситуації, коли деякі камери можуть бути видалені після донавчання.

Точність відстеження складно оцінити за допомогою лише однієї метрики, тому для оцінки застосовано набір метрик, що пропонується в оригінальній роботі SORT [44] та у роботі з описом бенчмарки MOT [73]:

• Multi-object tracking accuracy (MOTA): підсумок загальної точності відстеження з точки зору хибнопозитивних та хибнонегативних результатів, а також перемикань ідентифікаторов. Збільшення метрики відбувається при збільшенні кількості вірних відстежень та означає покращення роботи трекінгу.

• Multi-object tracking precision (MOTP): підсумок загальної точності відстеження з точки зору перетину реальної та передбаченої обмежувальних рамок. Збільшення метрики MOTP відповідає підвищенню точності встановлення таких рамок.

• Mostly tracked (MT): відсоток відстежень, що відповідають дійсності, серед тих, що мають той самий ідентифікатор щонайменше протягом 80% часу існування. Збільшення метріки означає покращення стабільності довгочасного трекінгу.

• Mostly lost (ML): відсоток відстежень, які відповідають дійсності, серед тих, відстежуються протягом щонайбільше 20% часу існування. Зменшення метрики ML відповідає зменшенню кількості об’єктів, які більшу частину часу існування не були правильно відстежені.

• Перемикання ідентифікаторів (ID): кількість змін ідентифікатора реально незмінного відстеженого об’єкту. Зменшення кількості перемикань сигналізує про покращення стабільності трекінгу.

• Фрагментація (FM): кількість переривань відстеження через відсутність виявлення. Зменшення значення означає зменшення кількості хибнонегативних результатів виявлення.

Результати оцінки відстеження об’єктів наведені у таблиці 6.2.

Таблиця 6.2 – Результати оцінки методів відстеження об’єктів

Метод	MOTA	MOTP	MT	ML	ID	FM
Повний датасет MOT16, детектор - модифікована Faster RCNN						
SORT	59.8	79.6	25.4	22.7	1423	1835
DeepSORT	61.4	79.1	32.8	18.2	781	2008
Адаптований датасет MOT16, детектор - YOLO						
SORT	59.9	79.6	25.4	22.3	767	1233
DeepSORT	61.5	79.3	33.5	16.9	320	961
Удосконалений метод до донавчання	61.7	79.2	33.5	16.8	288	1018
Удосконалений метод після донавчання	63.0	79.6	34.9	16.1	241	893

Варто зазначити, що оцінка в оригінальних роботах SORT та DeepSORT ввиконувалась з використанням даних виявлення об'єктів з модифікованої мережі Faster RCNN [75], що демонструє 55.7% mAP на датасеті COCO [32]. Також зазначимо, що абсолютні показники метрик ID та FM залежать від кількості об’єктів, тому має сенс порівнювати їх тільки для методів, що виявляють однакову кількість класів, та на наборах даних, що не відрізняються.

Графічне представлення змін відносних значень метрик наведено на рис. 6.2.

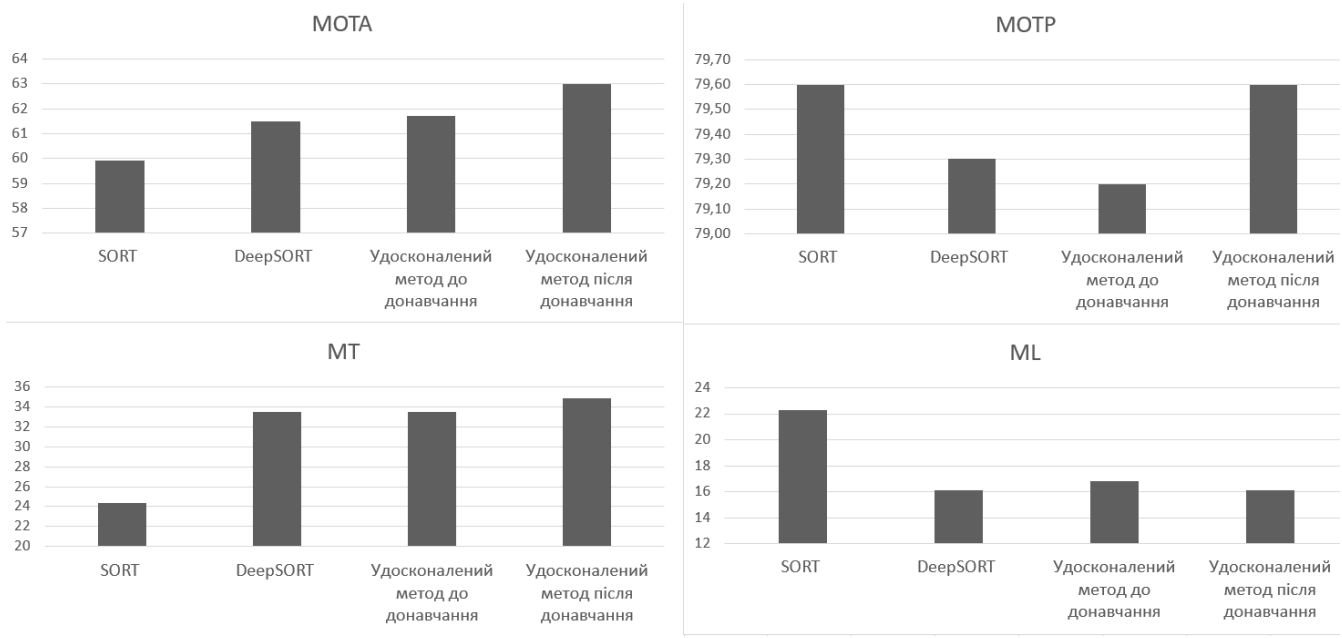

Рисунок 6.2 – Діаграми порівняння метрік оцінки відстеження об’єктів

Зміни у методі відстеження незначно вплинули на його точність (метрика MOTA +0.2, метрика MOTP -0.1), але донавчання на нових даних суттєво покращує результати – MOTA +1.5, MOTP +0.3, MT +1.4, ML -0.8. Такі зміни метрик демонструють зменшення кількості хибнонегативних результатів, підвищення точності передбачення обмежувальних рамок, та зменшення кількості втрат ідентифікаторів об’єктів. В абсолютному вираженні значно зменшилися кількість перемикань ідентифікаторів (на 526) та кількість втрат об’єкту, що відстежується, через помилки виявлення (на 340).

6.3 Випробування точності розпізнавання небезпечних ситуацій

Наступний етап перевірки системи ключовий – це оцінка точності розпізнавання небезпечних ситуацій. Для такої перевірки знов використано датасет MOT, а в систему додано налаштування, що штучно створюють небезпечні ситуації різних типів у тестових відео. Реальну кількість та час виникнення ситуацій підраховано вручну. Значення для оцінки – кількість правильних розпізнавань ситуацій, середня похибка часу розпізнавання (середня різниця між реальним часом виникнення ситуації та часом, що вказала система), а також кількість хибних спрацювань. Результати перевірки наведено у таблиці 6.3.

Таблиця 6.3 – Результати перевірки точності розпізнавання небезпечних ситуацій

Тип ситуації	Поява об’єкту	Скупчення об’єктів	Перетин ліній	Раптове зникнення	Підозріла поведінка	Всього
Кількість виникнень	20	10	15	5	5	55
Кількість розпізнавань незмінним методом	19	10	8	4	2	43
Кількість розпізнавань удосконаленим методом	19	10	15	5	4	53

Продовження таблиці 6.3

Тип ситуації	Поява об’єкту	Скупчення об’єктів	Перетин ліній	Раптове зникнення	Підозріла поведінка	Всього
Середня похибка часу при розпізнаванні незмінним методом, с	~0	0.54	1.12	~0	3.52	1.72
Середнія похибка часу при розпізнаванні удосконаленим методом, с	~0	0.4	0.49	~0	1.33	0.74
Кількість помилкових розпізнавань незмінним методом	2	0	4	0	1	7
Кількість помилкових розпізнавань удосконаленим методом	1	0	1	1	0	3

Результати перевірки показують, що удосконалення методу відстеження об’єктів дозволило підвищити кількість правильних розпізнавань небезпечних ситуацій на 17%, при цьому зменшивши похибку часу розпізнавання у середньому на 0.98 секунд та кількість хибних спрацювань на 57 %.

6.4 Випробування швидкості обробки кадрів

Хоча швидкість роботи не є основним параметром оцінки системи, вона має відповідати значенням, наведеним у вимогах до продуктивності. Тому було випробувано також час роботи усієї системи від моменту зчитування кадру до закінчення його повної обробки. Перевірку проведено шляхом замірів середнього

часу обробки кадрів для 3 одночасно підключених камер з відеоданими з набору MOT на 3х пристроях різної продуктивності. Результати перевірки наведено у таблиці 6.4.

Таблиця 6.4 – Результати перевірки часу обробки кадрів

Пристрій	Nvidia GT 430 (Tesla) 1 Gb, Intel Pentium Gold 6805 1.7 GHz	Nvidia GTX 1060 (Pascal) 3 Gb, Intel Core i7 7500U 2.7 GHz	Nvidia GTX 1660 Ti (Turing) 6 Gb, AMD Ryzen 5 3600 3.6 GHz
Середній час обробки кадру, с	0.17	0.09	0.06

Середній час обробки одного кадру становить від 0.17 до 0.06 секунд, що не перевищує наведеного в описі нетехнічних вимог значення 0.2 секунди.

6.5 Висновки до розділу

У даному розділі проведено випробування ключових характеристик системи – точності виявлення та відстеження об’єктів та розпізнавання небезпечних ситуацій, а також швидкості обробки кадрів.

У результаті випробування системи виявлено покращення метрик виявлення та відстеження об’єктів, що дозволило збільшити кількість правильно розпізнаних небезпечних ситуацій на 17%, зменшивши час реагування системи на приблизно 1 секунду та кількість хибних спрацювань на 57%. При цьому швидкість обробки кадрів на пристрої з мінімальними допустимими апаратними характеристиками становить 0.17 секунд, що відповідає вимогам до системи.

За результатами випробування визначено, що система відповідає описаним вимогам, а мету створення удосконаленого методу трекінгу об’єктів виконано.

ВИСНОВКИ

В даній роботі було проведено удосконалення методу трекінгу об’єктів у даних відеоспостереження в реальному часі та створено програмну систему для автоматичного розпізнавання небезпечних ситуацій на його основі. У процесі роботи було виконано наступні задачі:

- проаналізовано предметну область та існуючі рішення, що надають функціонал відеоспостереження та відеоаналітики;
- проведено огляд наявних способів трекінгу, моделей та методів виявлення та відстеження об’єктів;
- обрано найпридатніші для подальшого дослідження методи та моделі виявлення та відстеження, на базі яких спроектовано та реалізовано удосконалений метод трекінгу об’єктів;
- створено алгоритм розпізнавання небезпечних ситуацій на основі даних трекінгу;
- описано вимоги до програмної системи для розпізнавання небезпечних ситуацій у даних відеоспостереження;
- спроектовано архітектуру високорівневих компонентів програмної системи та описано деталі їх програмної реалізації;
- виконано програмну реалізацію та проведено тестування функціоналу створеної системи;
- випробувано характеристики отриманого методу трекінгу та створеної програмної системи загалом.

Результати випробування демонструють, що алгоритм розпізнавання небезпечних ситуацій на основі удосконаленого методу трекінгу та донавчання мережі для виявлення об’єктів на нових даних дозволив збільшити кількість правильних розпізнавань небезпечних ситуацій на 17% та зменшити кількість хибних

спрацювань на 57%. При цьому середній час обробки кадрів при використанні пристрою з мінімально допустимими апаратними характеристиками складає 0.17 секунд.

Подальший розвиток розробленого методу включає підвищення точності трекінгу шляхом випробування підходів до додаткового попереднього навчання нейронних мереж, використання автоматичних методів конфігурації параметрів методу відстеження, а також розширення методу на використання для рухомих камер та передачу даних трекінгу між камерами.

Напрямки розвитку створеної системи – розширення функціоналу аналітики шляхом збільшення кількості оброблюваних ситуацій, розширення можливостей функціоналу сповіщення, оптимізація потоків та методів перетворення даних і покращення синхронізації паралельних частин з метою збільшення кількості одночасно підтримуваних камер та загальної швидкості роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. L. Fennelly. Effective Physical Security, Fifth Edition. Butterworth-Heinemann, MA, United States, pp. 105-112.
2. A. Caputo. Digital Video Surveillance and Security. Butterworth-Heinemann, MA, United States, pp. 40-88.
3. A. Adams, J. Ferryman. The future of video analytics for surveillance and its ethical implications. Security Journal 28(3), November 2012.
4. G. F. Shidik. A Systematic Review of Intelligence Video Surveillance: Trends, Techniques, Frameworks, and Datasets. IEEE Access, 25 November 2019, Vol. 7, pp. 170457-170473.
5. R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. Research Institute for Advanced Study, Baltimore, Md, 1960.
6. H W. Kuhn. The Hungarian method for the assignment problem. Naval research logistics, March 1955, Volume 2, Issue 1-2.
7. O. Elharrouss, N. Almaadeed, S. Al-Maadeed. A review of video surveillance systems. Journal of Visual Communication and Image Representation. Vol. 77, May 2021.
8. L. J. Fennelly, M. A. Perry. Physical Security: 150 Things You Should Know (Second Edition). Butterworth-Heinemann, 2017.
9. Cisco Meraki [Електронний ресурс] – Режим доступу до ресурсу: https://meraki.cisco.com/products/smart-cameras/.
10. Video surveillance software review [Електронний ресурс] – Режим доступу до ресурсу: https://www.getkisi.com/blog/best-video-surveillance-software.
11. Rhombus [Електронний ресурс] – Режим доступу до ресурсу: https://www.rhombussystems.com/.
12. EyeLine Video Management Software [Електронний ресурс] – Режим доступу до ресурсу: https://www.nchsoftware.com/surveillance/index.html.

13. ContaCam [Електронний ресурс] – Режим доступу до ресурсу: https://www.contaware.com/contacam.html.
14. iSpy [Електронний ресурс] – Режим доступу до ресурсу: https://www.ispyconnect.com/features.aspx
15. C. L. Smith, D. J.Brook. Security Science. The Theory and Practice of Security. Edith Cowan University, 2013, pp. 153-175.
16. Nik Gagvani. Introduction to video analytics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.eetimes.com/introduction-to-video-analytics/
17. Intelligent Video and Remote Monitoring | Verizon [Електронний ресурс] – Режим доступу до ресурсу: https://www.verizon.com/business/products/internet-of-things/connected-smart-cities-communities/intelligent-video/
18. BriefCam [Електронний ресурс] – Режим доступу до ресурсу: https://www.briefcam.com/technology/video-analytics/.
19. 5725-H94 IBM Intelligent Video Analytics V3.0 [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_sm/4/877/ENUS5725-H94/index.html&lang=en&request_locale=en
20. Technical documentation for Video Analytics. Bosch Security and Safety Systems | Global. [Електронний ресурс] – Режим доступу до ресурсу: https://resources-boschsecurity-cdn.azureedge.net/public/documents/DS_IVA_7.10_Data_sheet_enUS_69630079883.pdf
21. Product | Calipsa Pro Analytics | Advanced video analytics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.calipsa.io/product/calipsa-pro-analytics
22. S. Dasiopoulou and others. Knowledge-assisted semantic video object detection. IEEE Transactions on Circuits and Systems for Video Technology. 15.10.2005, pp. 1210–1224.

23. Paul Viola, Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. Accepted conference on computer vision and pattern recognition, 2001.
24. Ruvinskaya V. M., Timkov Y. Y. Deep Learning Technology for Videoframe Processing in Face Segmentation on Mobile Devices. Herald of Advanced Information Technology. 2021; Vol.4 No.2: 185–194. DOI: https://doi.org/10.15276/hait.02.2021.7.
25. D. G. Lowe. Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision. 20-27 Sept. 1999, pp. :1150-1157.
26. D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, January 5, 2004; Vol.60 Isuue 2, pp 91-110.
27. Navneet Dalal, Bill Triggs. Histograms of Oriented Gradients for Human Detection. International Conference on Computer Vision & Pattern Recognition, Jun 2005, San Diego, United States. pp.886–893.
28. Kobayashi, T., Hidaka, A., & Kurita, T. Selection of histograms of oriented gradients features for pedestrian detection. International conference on neural information processing Springer, Berlin, Heidelberg, 2007, pp. 598-607.
29. Girshick, Ross & Donahue, Jeff & Darrell, Trevor & Malik, Jitendra. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 10.1109/CVPR.2014.81.
30. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. Understanding object detection algorithms. [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.
31. R. Girshick. Fast R-CNN. IEEE International Conference on Computer Vision (ICCV). 7-13 Dec. 2015. 10.1109/ICCV.2015.169.

32. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. 06 June 2016; pp 1137 – 1149.
33. W. Liu, D. Anguelov, D. Erhan, C. Szegedy and others. SSD: Single Shot MultiBox Detector. European Conference on Computer Vision. October 2016. DOI:10.1007/978-3-319-46448-0_2
34. J. Huang and others. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. IEEE Conference on Computer Vision and Pattern Recognition. 21-26 July 2017.
35. J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. IEEE Conference on Computer Vision and Pattern Recognition. 27-30 June 2016. DOI: 10.1109/CVPR.2016.91.
36. J. Redmon, A. Farhadi. YOLO9000: Better, Faster, Stronger. IEEE Conference on Computer Vision and Pattern Recognition. 21-26 July 2017. DOI:10.1109/CVPR.2017.690.
37. Shifeng Zhang, Longyin Wen. Single-Shot Refinement Neural Network for Object Detection. IEEE/CVF Conference on Computer Vision and Pattern Recognition. 18-23 June 2018. DOI:10.1109/CVPR.2018.00442.
38. Tsung-Yi Lin, P. Goyal, R. Girshick and others. Focal Loss for Dense Object Detection. IEEE Conference on Computer Vision and Pattern Recognition. 21-26 July 2017. DOI: 10.1109/ICCV.2017.324.
39. Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Tae-Kyun Kim. Multiple object tracking: A literature review. Artificial Intelligence, Volume 293, 2021, 103448, ISSN 0004-3702.
40. Chen Shengyong and others. Deep Learning for Multiple Object Tracking: A Survey. IET Computer Vision 13(4), January 2019. 13. 10.1049/iet-cvi.2018.5598.

41. Object Tracking – An Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://viso.ai/deep-learning/object-tracking/.
42. Hyeonseob Nam, Bohyung Han. Learning Multi-Domain Convolutional Neural Networks for Visual Tracking. IEEE Conference on Computer Vision and Pattern Recognition. 27-30 June 2016. DOI: 10.1109/CVPR.2016.465.
43. I. Jung, J. Son, M. Baek, B. Han. Real-time MDNet. European Conference on Computer Vision (ECCV). 8-14 September 2018. arXiv:1808.08834.
44. A. Bewley, Z. Ge, L. Ott, F. Ramos. Simple Online and Realtime Tracking. IEEE International Conference on Image Processing (ICIP). 25-28 Sept. 2016. DOI: 10.1109/ICIP.2016.7533003.
45. N. Wojke; A. Bewley; D. Paulus. Simple online and realtime tracking with a deep association metric. IEEE International Conference on Image Processing. 7-20 Sept. 2017. DOI: 10.1109/ICIP.2017.8296962.
46. J. Redmon, A. Farhadi. YOLOv3: An Incremental Improvement. University of Washington, 8 April 2018. arXiv:1804.02767.
47. A. Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. 23 April 2020. arXiv:2004.10934.
48. D. Bolya, C. Zhou, F. Xiao, Yong Jae Lee. YOLACT: Real-time Instance Segmentation. University of California, 4 April 2019. arXiv:1904.02689.
49. YoloV5 [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ultralytics/yolov5.
50. Oleksii I. Sheremet,Oleksandr Ye. Korobov,Oleksandr V. Sadovoi,Yuliia V. Sokhina. Intelligent System Based on a Convolu-tional Neural Network for Identifying People without Breathing Masks.Applied Aspects of InformationTechnology. 2020;Vol.3 No.3: 133–134.DOI: https://doi.org/10.15276/aait.03.2020.2

51. He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, Sun, Jian. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2014, 37. 10.1109/TPAMI.2015.2389824.
52. K. Wang, J. Liew. PANet: Few-Shot Image Semantic Segmentation With Prototype Alignment. IEEE/CVF International Conference on Computer Vision (ICCV), October 2019.
53. Sangdoo Yun, Dongyoon Han and others. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. Accepted at ICCV 2019. arXiv:1905.04899.
54. G. Ghiasi, Tsung-Yi Lin, Quoc V. Le. DropBlock: a regularization method for convolutional networks Proceedings of the 32nd International Conference on Neural Information Processing Systems. December 2018, pp. 10750–10760.
55. Mahalanobis, P. C. On the generalised distance in statistics. Proceedings of the National Institute of Sciences of India. 1936, 2, pp. 49–55.
56. L. Metcalf, W. Casey. Cybersecurity and Applied Mathematics (1st. ed.). Syngress Publishing, 2016, pp. 5-20.
57. Николенко А. Глубокое обучение. Погружение в мир нейронных сетей. / А. Николенко Е. Кадурин, С. Архангельская — СПб.: Питер, 2018. – с. 449.
58. Тимков Ю.Ю., Рувинская В.М. Ускорение сегментации лица на видео для мобильных устройств с помощью глубинного обучения / Современные информационные технологии 2020 (MIT2020). Матеріали десятої Міжнародної конференції студентів і молодих науковців, 14-15 травня 2020. – с. 56-57.
59. YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection [Електронний ресурс] – Режим доступу до ресурсу:https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

60. Petrosiuk D. V., Arsirii O. O., Babilunha O. Ju., Nikolenko A. O. Deep Learning Technology of Convolutional Neural Networks for Facial Expression Recognition. Applied Aspects of Information Technology. 2021; Vol.4 No.2: 192–201. DOI:https://doi.org/10.15276/aait.02.2021.6
61. H. Rezatofighi and others. Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression. IEEE/CVF Conference on Computer Vision and Pattern Recognition. June 2019. DOI:10.1109/CVPR.2019.00075.
62. ISO – ISO/IEC/IEEE 29148:2018 - Systems and software engineering – Life cycle processes — Requirements engineering [Електронний ресурс] – Режим доступу до ресурсу: https://www.iso.org/standard/72089.html
63. The component diagram – IBM Developer engineering [Електронний ресурс] – Режим доступу до ресурсу: https://developer.ibm.com/articles/the-component-diagram
64. Р. С. Мартин, Дж. В. Ньюкирк. Быстрая разработка программ. Принципы, примеры, практика. Вильямс, 2004. – 752 с.
65. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Санкт-Петербург, 2001 – 344 с.
66. Larman Cr. Applying UML and Patterns: Introduction to Object Oriented Analysis and Design. Prentice Hall Inc., 1998 – 616 c.
67. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/
68. PyTorch [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/
69. Qt for Python [Електронний ресурс] – Режим доступу до ресурсу: https://doc.qt.io/qtforpython/
70. NumPy [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/

71. Home – OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://opencv.org/
72. MatPlotlib – Visualization with Python [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/
73. A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler. MOT16: A benchmark for multi-object tracking. arXiv:1603.00831, 2016.
74. Padilla, Rafael & Netto, Sergio & da Silva, Eduardo. A Survey on Performance Metrics for Object-Detection Algorithms. International Conference on Systems, Signals and Image Processing. July 2020, DOI:10.1109/IWSSIP48289.2020.
75. F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, Poi: Multiple object tracking with high performance detection and appearance feature. ECCV, Springer, 2016, pp. 36–42.

ДОДАТОК А. ЛІСТИНГ ПРОГРАМНОГО КОДУ

```
    from yolov4.models.experimental import attempt_load
    from yolov4.utils.downloads import attempt_download
    from yolov4.models.common import DetectMultiBackend
    import os
    import numpy as np
    import matplotlib
    matplotlib.use('TkAgg')
    import matplotlib.pyplot as plt
    import matplotlib.patches as patches
    from skimage import io
    import glob
    import time
    import argparse
    from filterpy.kalman import KalmanFilter
    from yolov4.utils.datasets import LoadImages, LoadStreams
    from yolov4.utils.general import LOGGER, check_img_size,
non_max_suppression, scale_coords, check_imshow, xyxy2xywh
    from yolov4.utils.torch_utils import select_device, time_sync
    from yolov4.utils.plots import Annotator, colors
    from deep_sort_pytorch.utils.parser import get_config
    from deep_sort_pytorch.deep_sort import DeepSort
    import argparse
    import os
    import platform
    import shutil
    from pathlib import Path
    import torch
    import torch.backends.cudnn as cudnn

    from PyQt5.QtGui import *
    from PyQt5.QtCore import *
    from PyQt5.QtWidgets import *
    import qdarkstyle
    import cv2


    np.random.seed(0)


    class ObjectTrackingData:
     def __init__(self, id, cls, position, size):
          self.id = id
          self.class = cls
          self.position = position
          self.size = size
          self.timeCreated = getTime()

    class ObjectAnalyticsData(ObjectTrackingData):
     def __init__(self, id, cls, position, size, speed, avgSpeed,
avgAcceleration,\
          timeExists, timesInView \
```

```
            durationInView, avgDurationInView):
            super(ObjectTrackingData, self).__init__(id, cls, position, size)
            self.speed = speed
            self.avgSpeed = avgAcceleration
            self.timeExists = timeExists
            self.timesInView = timesInView
            self.durationInView = durationInView
            self.avgDurationInView = avgDurationInView

      class CommonObjectData:
       def __init__(self, cls, curCount, avgCount, avgSpeed, avgAcceleration,
            avgTimeExists, avgTimesInView, avgDurationInView)
            self.cls = cls
            self.curCount = curCount
            self.avgSpeed = avgSpeed
            self.avgAcceleration = avgAcceleration
            self.avgTimeExists = avgTimeExists
            self.avgTimesInView = avgTimesInView
            self.avgDurationInView = avgDurationInView

      class SituationReport:
       def __init__(self):
            self.typeId = -1
            self.time = getTime()
            self.objectsInvolved = []
            self.description = ''

      class BaseTrackingResultProcessor:
       def process(self, data, cameraHId)
            raise Exception("Method not implemented")

      class SituationDetectionNotifier:
       def __init__(self, arg):
            self.arg = arg
            self.observers = []

       def addObserver(observer):
            if observer not in self.observers:
                 self.observers.append(observer)

       def notifyObservers(reports):
            for observer in self.observers:
                 observer.onNotified(reports)

      class            SituationDetectionProcessor(BaseTrackingResultProcessor,
SituationDetectionNotifier):
       def __init__(self, settingsProvider):
            super(SituationDetectionProcessor, self).__init__()
            self.objectsData = {}
            self.commonData = {}
            self.situationDetectors = []
            self.situationDetectors.append(AppearanceDetector())
            self.situationDetectors.append(TimesCntDetector())
```

```
            self.situationDetectors.append(DurationDetector())
            self.situationDetectors.append(ZoneDetector())
            self.situationDetectors.append(DisapearanceDetector())
            self.situationDetectors.append(SuspectsDetector())
            self.settingsProvider = settingsProvider


      def processTrackingResult(self, data, cameraHId):
            self.collectObjectAnalytics(data, cameraHId)
            reports = self.checkSituations(cameraHId)
            reports =self.aggregateSituations(reports)
            self.notifyObservers(reports)


      def collectObjectAnalytics(data, cameraHId):
            updatedObjs = []
            for cd in commonData:
                  cd.curCount = 0

            for obj in data:
                  updatedObjs.append(obj.id)
                  prevObj = self.objectsData[obj.id]
                  obj.speed = (obj.position - prevObj.position) * getTime()
                  obj.timeExists = obj.timeExists + getDeltaTime()
                  obj.avgSpeed  =  (prevObj.avgSpeed  *  prevObj.getTime()  +
obj.speed * getDeltaTime()) / obj.timeExists
                  obj.timeExists = prevObj.timeExists + getDeltaTime()
                  obj.durationInView = prevObj.durationInView + getDeltaTime()
                  cd = commonData[obj.cls]
                  cd.curCount = commonData[obj.cls].curCount + 1
                  cd.avgSpeed = (cd.avgSpeed + obj.avgSpeed) / cd.curCount
                  cd.avgAcceleration = (cd.avgAcceleration * (cd.curCount - 1) +
obj.avgAcceleration) / cd.curCount
                  cd.avgTimeExists = (cd.avgTimeExists * (cd.curCount - 1) +
obj.avgTimeExists) / cd.curCount
                  cd.avgTimesInView = (cd.avgTimesInView * (cd.curCount - 1) +
obj.avgTimesInView) / cd.curCount
                  cd.avgDurationInView = (cd.avgDurationInView * (cd.curCount -
1) + obj.avgDurationInView) / cd.curCount
                  if obj not in cd.objs:
                        cd.objs.append(obj)

                  for obj in self.objectsData:
                  if obj.id not in updatedObjs:
                        obj.timesInView = obj.timesInView + 1
                        obj.timeExists = obj.timeExists + getDeltaTime()
                        obj.avgDurationInView   =   (obj.avgDurationInView   +
obj.durationInView) / obj.timeExists

      def checkSituations(cameraHId):
            reportsList = []
            for detector in self.situationDetectors:
                  for obj in self.objectsData:
```

```
                    reports = detector.detect(obj, commonData[obj.cls],
settingsProvider.getSituationParams(detector.type, cameraHId))
                    if reports is not None:
                         reportsList.append(reports)
          return reports

     def aggregateSituations(reports):
          for report in reports:
               for objId in report.objectsInvolved:
                    for otherReport in reports:
                         if objId in otherReport.objectsInvolved or
((report.time - otherReport.time) < gDelta and report.type is
otherReport.type):
                              report.append(otherReport)
                              reports.remove(otherReport)
          return reports

    class SituationsDetector:
     def __init__():
          self.type = -1

     def detect(object, commonData, params, cameraHId):
          raise Exception("Method not implemented")

    class AppearanceDetector(SituationDetector):
     def __init__():
          super(AppearanceDetector, self).__init__()
          self.type = 0

     def detect(obj, commonData, params, cameraHId):
          if obj.cls in params.prohibitedClasses:
               rep = SituationReport()
               rep.typeId = self.type
               rep.objectsInvolved.append(obj)
               return rep

    class TimesCntDetector(SituationDetector):
     def __init__():
          super(AppearanceDetector, self).__init__()
          self.type = 1

     def detect(obj, commonData, params, cameraHId):
          if obj.cls in params.cntLimitClasses:
               if params.cntLimitClasses[obj.cls] >=
commonData[obj.cls].curCount:
                    rep = SituationReport()
                    rep.typeId = self.type
                    rep.objectsInvolved = commonData.objs
                    return rep

    class DurationDetector(SituationDetector):
     def __init__():
          super(AppearanceDetector, self).__init__()
```

```
            self.type = 2

      def detect(obj, commonData, params, cameraHId):
            if obj.cls in params.timeLimitClasses:
                  if (obj.durationInView - params.timeLimitClasses[obj.cls]) >=
gDelta:
                        rep = SituationReport()
                        rep.typeId = self.type
                        rep.objectsInvolved = [obj]
                        return rep

     class ZoneDetector(SituationDetector):
      def __init__():
            super(AppearanceDetector, self).__init__()
            self.type = 3

      def detect(obj, commonData, params, cameraHId):
            if len(params.zones[cameraHId]) > 0:
                  for zone in params.zones[cameraHId]:
                        if (mth.checkBounding(obj.position, zone.position))
                              rep = SituationReport()
                              rep.typeId = self.type
                              rep.objectsInvolved = [obj]
                              rep.description  =  rep.description.join('\"Зона\":
\"' + zone.title + '\" \"Камера\":\"' + zone.cameraTitle)
                              return rep

     class DisppearanceDetector(SituationDetector):
      def __init__():
            super(AppearanceDetector, self).__init__()
            self.type = 4
            self.stableList = []

      def detect(obj, commonData, params, cameraHId):

            self.updateStable(obj, cameraHId)

            if  obj.timeExists  +  obj.timeCreated  -  getTime()  <  gDelta  and
isStable(obj):
                  rep = SituationReport()
                  rep.typeId = self.type
                  rep.objectsInvolved.append(obj)
                  return rep

      def updateStable(obj, params, cameraHId):
            if obj.timeExists > params[obj.cls].timeToStab:
                  if obj.avgSpeed < gSDelta and obj.avgAcceleration < gADelta:
                        self.stableList[cameraHId].append(obj)
                  else if obj in self.stableList[cameraHId]:
                        self.stableList[cameraHId].remove(obj)


      def isStable(obj):
```

```
            for stableCList in self.stableList:
                 return obj in stableCList


    def linear_assignment(cost_matrix):
      try:
        import lap
        _, x, y = lap.lapjv(cost_matrix, extend_cost=True)
        return np.array([[y[i],i] for i in x if i >= 0])
      except ImportError:
        from scipy.optimize import linear_sum_assignment
        x, y = linear_sum_assignment(cost_matrix)
        return np.array(list(zip(x, y)))


    def iou_batch(bb_test, bb_gt):

      bb_gt = np.expand_dims(bb_gt, 0)
      bb_test = np.expand_dims(bb_test, 1)

      xx1 = np.maximum(bb_test[..., 0], bb_gt[..., 0])
      yy1 = np.maximum(bb_test[..., 1], bb_gt[..., 1])
      xx2 = np.minimum(bb_test[..., 2], bb_gt[..., 2])
      yy2 = np.minimum(bb_test[..., 3], bb_gt[..., 3])
      w = np.maximum(0., xx2 - xx1)
      h = np.maximum(0., yy2 - yy1)
      wh = w * h
      o = wh / ((bb_test[..., 2] - bb_test[..., 0]) * (bb_test[..., 3] -
bb_test[..., 1])
        + (bb_gt[..., 2] - bb_gt[..., 0]) * (bb_gt[..., 3] - bb_gt[..., 1]) -
wh)
      return(o)


    def convert_bbox_to_z(bbox):

      w = bbox[2] - bbox[0]
      h = bbox[3] - bbox[1]
      x = bbox[0] + w/2.
      y = bbox[1] + h/2.
      s = w * h
      r = w / float(h)
      return np.array([x, y, s, r]).reshape((4, 1))


    def convert_x_to_bbox(x,score=None):

      w = np.sqrt(x[2] * x[3])
      h = x[2] / w
      if(score==None):
        return                                        np.array([x[0]-w/2.,x[1]-
h/2.,x[0]+w/2.,x[1]+h/2.]).reshape((1,4))
      else:
```

```
        return np.array([x[0]-w/2.,x[1]-
h/2.,x[0]+w/2.,x[1]+h/2.,score]).reshape((1,5))


    class KalmanBoxTracker(object):

      count = 0
      def __init__(self,bbox):


        self.kf = KalmanFilter(dim_x=7, dim_z=4)
        self.kf.F =
np.array([[1,0,0,0,1,0,0],[0,1,0,0,0,1,0],[0,0,1,0,0,0,1],[0,0,0,1,0,0,0],
[0,0,0,0,1,0,0],[0,0,0,0,0,1,0],[0,0,0,0,0,0,1]])
        self.kf.H =
np.array([[1,0,0,0,0,0,0],[0,1,0,0,0,0,0],[0,0,1,0,0,0,0],[0,0,0,1,0,0,0]])

        self.kf.R[2:,2:] *= 10.
        self.kf.P[4:,4:] *= 1000.
        self.kf.Q[-1,-1] *= 0.01
        self.kf.Q[4:,4:] *= 0.01

        self.kf.x[:4] = convert_bbox_to_z(bbox)
        self.time_since_update = 0
        self.id = KalmanBoxTracker.count
        KalmanBoxTracker.count += 1
        self.history = []
        self.hits = 0
        self.hit_streak = 0
        self.age = 0

      def update(self,bbox):

        self.time_since_update = 0
        self.history = []
        self.hits += 1
        self.hit_streak += 1
        self.kf.update(convert_bbox_to_z(bbox))

      def predict(self):

        if((self.kf.x[6]+self.kf.x[2])<=0):
          self.kf.x[6] *= 0.0
        self.kf.predict()
        self.age += 1
        if(self.time_since_update>0):
          self.hit_streak = 0
        self.time_since_update += 1
        self.history.append(convert_x_to_bbox(self.kf.x))
        return self.history[-1]

      def get_state(self):
```

```
        return convert_x_to_bbox(self.kf.x)


    def  associate_detections_to_trackers(detections,trackers,iou_threshold  =
0.3):

      if(len(trackers)==0):
        return     np.empty((0,2),dtype=int),     np.arange(len(detections)),
np.empty((0,5),dtype=int)

      iou_matrix = iou_batch(detections, trackers)

      if min(iou_matrix.shape) > 0:
        a = (iou_matrix > iou_threshold).astype(np.int32)
        if a.sum(1).max() == 1 and a.sum(0).max() == 1:
            matched_indices = np.stack(np.where(a), axis=1)
        else:
          matched_indices = linear_assignment(-iou_matrix)
      else:
        matched_indices = np.empty(shape=(0,2))

      unmatched_detections = []
      for d, det in enumerate(detections):
        if(d not in matched_indices[:,0]):
          unmatched_detections.append(d)
      unmatched_trackers = []
      for t, trk in enumerate(trackers):
        if(t not in matched_indices[:,1]):
          unmatched_trackers.append(t)

      matches = []
      for m in matched_indices:
        if(iou_matrix[m[0], m[1]]<iou_threshold):
          unmatched_detections.append(m[0])
          unmatched_trackers.append(m[1])
        else:
          matches.append(m.reshape(1,2))
      if(len(matches)==0):
        matches = np.empty((0,2),dtype=int)
      else:
        matches = np.concatenate(matches,axis=0)

      return             matches,             np.array(unmatched_detections),
np.array(unmatched_trackers)


    class ObjectTracker(object):
      def __init__(self, max_age=1, min_hits=3, iou_threshold=0.3):

        self.max_age = max_age
        self.min_hits = min_hits
        self.iou_threshold = iou_threshold
        self.trackers = []
```

```
        self.frame_count = 0

      def update(self, dets=np.empty((0, 5))):

        self.frame_count += 1

        trks = np.zeros((len(self.trackers), 5))
        to_del = []
        ret = []
        for t, trk in enumerate(trks):
          pos = self.trackers[t].predict()[0]
          trk[:] = [pos[0], pos[1], pos[2], pos[3], 0]
          if np.any(np.isnan(pos)):
            to_del.append(t)
        trks = np.ma.compress_rows(np.ma.masked_invalid(trks))
        for t in reversed(to_del):
          self.trackers.pop(t)
        matched,          unmatched_dets,          unmatched_trks          =
associate_detections_to_trackers(dets,trks, self.iou_threshold)

        for m in matched:
          self.trackers[m[1]].update(dets[m[0], :])


        for i in unmatched_dets:
            trk = KalmanBoxTracker(dets[i,:])
            self.trackers.append(trk)
        i = len(self.trackers)
        for trk in reversed(self.trackers):
            d = trk.get_state()[0]
            if (trk.time_since_update < 1) and (trk.hit_streak >= self.min_hits
or self.frame_count <= self.min_hits):
              ret.append(np.concatenate((d,[trk.id+1])).reshape(1,-1))
            i -= 1

            if(trk.time_since_update > self.max_age):
              self.trackers.pop(i)
        if(len(ret)>0):
          return np.concatenate(ret)
        return np.empty((0,5))

    def loadSort(args):

      display = args.display
      phase = args.phase
      total_time = 0.0
      total_frames = 0
      colours = np.random.rand(32, 3)
      if(display):
        if not os.path.exists('mot_benchmark'):

          return
        plt.ion()
```

```
        fig = plt.figure()
        ax1 = fig.add_subplot(111, aspect='equal')

      if not os.path.exists('output'):
        os.makedirs('output')
      pattern = os.path.join(args.seq_path, phase, '*', 'det', 'det.txt')
      for seq_dets_fn in glob.glob(pattern):
        mot_tracker = Sort(max_age=args.max_age,
                           min_hits=args.min_hits,
                           iou_threshold=args.iou_threshold)
        seq_dets = np.loadtxt(seq_dets_fn, delimiter=',')
        seq = seq_dets_fn[pattern.find('*'):].split(os.path.sep)[0]

        with open(os.path.join('output', '%s.txt'%(seq)),'w') as out_file:

          for frame in range(int(seq_dets[:,0].max())):
            frame += 1
            dets = seq_dets[seq_dets[:, 0]==frame, 2:7]
            dets[:, 2:4] += dets[:, 0:2]
            total_frames += 1

            if(display):
              fn   =   os.path.join('mot_benchmark',   phase,   seq,   'img1',
'%06d.jpg'%(frame))
              im =io.imread(fn)
              ax1.imshow(im)
              plt.title(seq + ' Tracked Targets')

            start_time = time.time()
            trackers = mot_tracker.update(dets)
            cycle_time = time.time() - start_time
            total_time += cycle_time

            for d in trackers:
              print('%d,%d,%.2f,%.2f,%.2f,%.2f,1,-1,-1,-
1'%(frame,d[4],d[0],d[1],d[2]-d[0],d[3]-d[1]),file=out_file)
              if(display):
                d = d.astype(np.int32)
                ax1.add_patch(patches.Rectangle((d[0],d[1]),d[2]-d[0],d[3]-
d[1],fill=False,lw=3,ec=colours[d[4]%32,:]))

            if(display):
              fig.canvas.flush_events()
              plt.draw()
              ax1.cla()

    import threading

    img = []
    cap = None

    def get_image():
        retval, buffer = cv2.imencode('.png', img)
```

```
    response = make_response(buffer.tobytes())
    response.headers['Content-Type'] = 'image/png'
    return response


class CameraManager():
    def __init__(self, name):

        self.name = name

    def load(self):
        global img
        global app_not_done

        cap = cv2.VideoCapture(0)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
        cnt = 0
        while(app_not_done):
            # Capture frame-by-frame
            ret, frame = cap.read()
            if ret:
                img = frame
            else:
                cnt += 1
                if cnt < 4:
                    print("Could not read camera")

        cap.release()


def connectNew():
    thrd = CameraThread('camera_thread')
    thrd.daemon = True
    thrd.start()

    thrd.join()

import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["VECLIB_MAXIMUM_THREADS"] = "1"
os.environ["NUMEXPR_NUM_THREADS"] = "1"

import sys
sys.path.insert(0, './yolov4')

class ObjectTrackingManager:

    def __init__(self, drawer):
        self.drawer = drawer
```

```
        def track(self, opt):
            out, source, yolo_weights, deep_sort_weights, show_vid, save_vid,
save_txt, imgsz, evaluate, half = \
                opt.output,           opt.source,           opt.yolo_weights,
opt.deep_sort_weights, opt.show_vid, opt.save_vid, \
                    opt.save_txt, opt.imgsz, opt.evaluate, opt.half
            webcam = source == '0' or source.startswith(
                'rtsp')        or        source.startswith('http')        or
source.endswith('.txt')

              cfg = get_config()
            cfg.merge_from_file(opt.config_deepsort)
            attempt_download(deep_sort_weights,                   repo='mikel-
brostrom/Yolov4_DeepSort_Pytorch')
            deepsort = DeepSort(cfg.DEEPSORT.REID_CKPT,
                                max_dist=cfg.DEEPSORT.MAX_DIST,
min_confidence=cfg.DEEPSORT.MIN_CONFIDENCE,

max_iou_distance=cfg.DEEPSORT.MAX_IOU_DISTANCE,
                                max_age=cfg.DEEPSORT.MAX_AGE,
n_init=cfg.DEEPSORT.N_INIT, nn_budget=cfg.DEEPSORT.NN_BUDGET,
                                use_cuda=True)


            device = select_device(opt.device)
            half &= device.type != 'cpu'


            if not evaluate:
                if os.path.exists(out):
                    pass
                    shutil.rmtree(out)
                os.makedirs(out)


            device = select_device(device)
            model   =   DetectMultiBackend(opt.yolo_weights,   device=device,
dnn=opt.dnn)
            stride, names, pt, jit, onnx = model.stride, model.names, model.pt,
model.jit, model.onnx
            imgsz = check_img_size(imgsz, s=stride)

            half &= pt and device.type != 'cpu'
            if pt:
                model.model.half() if half else model.model.float()


            vid_path, vid_writer = None, None

            if show_vid:
                show_vid = check_imshow()
```

```
            if webcam:
                view_img = check_imshow()
                cudnn.benchmark = True
                dataset = LoadStreams(source, img_size=imgsz, stride=stride,
auto=pt and not jit)
                bs = len(dataset)
            else:
                dataset = LoadImages(source, img_size=imgsz, stride=stride,
auto=pt and not jit)
                bs = 1
            vid_path, vid_writer = [None] * bs, [None] * bs


            names = model.module.names if hasattr(model, 'module') else
model.names

            save_path = str(Path(out))

            txt_file_name = source.split('/')[-1].split('.')[0]
            txt_path = str(Path(out)) + '/' + txt_file_name + '.txt'

            if pt and device.type != 'cpu':
                model(torch.zeros(1,                                        3,
*imgsz).to(device).type_as(next(model.model.parameters())))  # warmup
            dt, seen = [0.0, 0.0, 0.0], 0
            for frame_idx, (path, img, im0s, vid_cap, s) in enumerate(dataset):
                t1 = time_sync()
                img = torch.from_numpy(img).to(device)
                img = img.half() if half else img.float()  # uint8 to fp16/32
                img /= 255.0  # 0 - 255 to 0.0 - 1.0
                if img.ndimension() == 3:
                    img = img.unsqueeze(0)
                t2 = time_sync()
                dt[0] += t2 - t1


                visualize = increment_path(save_dir / Path(path).stem,
mkdir=True) if opt.visualize else False
                pred = model(img, augment=opt.augment, visualize=visualize)
                t3 = time_sync()
                dt[1] += t3 - t2


                pred = non_max_suppression(pred, opt.conf_thres,
opt.iou_thres, opt.classes, opt.agnostic_nms, max_det=opt.max_det)
                dt[2] += time_sync() - t3

                for i, det in enumerate(pred):
                    seen += 1
                    if webcam:
                        p, im0, frame = path[i], im0s[i].copy(), dataset.count
                        s += f'{i}: '
                    else:
```

```
                    p, im0, frame = path, im0s.copy(), getattr(dataset,
'frame', 0)

                s += '%gx%g ' % img.shape[2:]
                save_path = str(Path(out) / Path(p).name)

                annotator = Annotator(im0, line_width=2, pil=not ascii)

                if det is not None and len(det):

                    det[:, :4] = scale_coords(
                        img.shape[2:], det[:, :4], im0.shape).round()


                    for c in det[:, -1].unique():
                        n = (det[:, -1] == c).sum()
                        s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "

                    xywhs = xyxy2xywh(det[:, 0:4])
                    confs = det[:, 4]
                    clss = det[:, 5]


                    outputs = deepsort.update(xywhs.cpu(), confs.cpu(),
clss.cpu(), im0)


                    if len(outputs) > 0:
                        for j, (output, conf) in enumerate(zip(outputs,
confs)):

                            bboxes = output[0:4]
                            id = output[4]
                            cls = output[5]

                            c = int(cls)  # integer class
                            label = f'{id} {names[c]} {conf:.2f}'
                            clr=colors
                             if g_situation:
                                  clr = (112, 112, 112)
                                  if id in g_situation_red_frames:
                                       clr = (56, 56, 255)

                            annotator.box_label(bboxes, label, color=clr)

                            if save_txt:

                                bbox_left = output[0]
                                bbox_top = output[1]
                                bbox_w = output[2] - output[0]
                                bbox_h = output[3] - output[1]

                                with open(txt_path, 'a') as f:
```

```
                                        f.write(('%g ' * 10 + '\n') % (frame_idx
+ 1, id, bbox_left,
                                                                       bbox_top,
bbox_w, bbox_h, -1, -1, -1, -1))  # label format

                    else:
                        deepsort.increment_ages()


                    LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')


                    im0 = annotator.result()

                    if show_vid:
                        #cv2.imshow(p, im0)
                        self.drawer.updateImage(im0)
                        self.drawer.update()

                        if cv2.waitKey(1) == ord('q'):  # q to quit
                            raise StopIteration

                    Save results (image with detections)
                    if save_vid:
                        if vid_path != save_path:
                            vid_path = save_path
                            if isinstance(vid_writer, cv2.VideoWriter):
                                vid_writer.release()
                            if vid_cap:
                                fps = vid_cap.get(cv2.CAP_PROP_FPS)
                                w                                          =
int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                                h                                          =
int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                            else:  # stream
                                fps, w, h = 30, im0.shape[1], im0.shape[0]
                                save_path += '.mp4'

                            vid_writer      =      cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                        vid_writer.write(im0)

            t = tuple(x / seen * 1E3 for x in dt)
            LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms
NMS per image at shape {(1, 3, *imgsz)}' % t)
            if save_txt or save_vid:
                print('Results saved to %s' % os.getcwd() + os.sep + out)


    class MyDialog(QDialog):
        def __init__(self, parent=None):
            super(MyDialog, self).__init__(parent)
            self.mQImage = None
```

```
        def updateImage(self, img):
            self.cvImage = img
            self.cvImage = self.resizeWithAspectRatio(self.cvImage, 640, 480)
            height, width, byteValue = self.cvImage.shape
            byteValue = byteValue * width

            cv2.cvtColor(self.cvImage, cv2.COLOR_BGR2RGB, self.cvImage)

            self.mQImage = QImage(self.cvImage, width, height, byteValue,
QImage.Format_RGB888)

        def paintEvent(self, QPaintEvent):
            painter = QPainter()
            painter.begin(self)
            if self.mQImage:
                painter.fillRect(5 + int(640 / 2) - int(self.mQImage.width() /
2), 5 + int(480 / 2) - int(self.mQImage.height() / 2), \
                    self.mQImage.width() + 10, self.mQImage.height() + 10,
QColor(150, 150, 150, 150))
                painter.drawImage(10 + int(640 / 2) - int(self.mQImage.width()
/ 2), \
                    10 + int(480 / 2) - int(self.mQImage.height() / 2),
self.mQImage)
            painter.end()

        def keyPressEvent(self, QKeyEvent):
            super(MyDialog, self).keyPressEvent(QKeyEvent)


        def resizeWithAspectRatio(self, image, width=None, height=None,
inter=cv2.INTER_AREA):
            dim = None
            (h, w) = image.shape[:2]

            if width is None and height is None:
                return image
            if width is None:
                r = height / float(h)
                dim = (int(w * r), height)
            else:
                r = width / float(w)
                dim = (width, int(h * r))

            return cv2.resize(image, dim, interpolation=inter)

    if __name__=="__main__":
        import sys
        app = QApplication(sys.argv)
        app.setStyleSheet(qdarkstyle.load_stylesheet())
        widget = MyDialog()

        button2 = QPushButton(widget)
```

```
        button2.setText("Налаштування камер")
        button2.setGeometry(680, 20, 250, 50)


        button1 = QPushButton(widget)
        button1.setText("Налаштування повідомлень")
        button1.setGeometry(680, 90, 250, 50)

        button3 = QPushButton(widget)
        button3.setText("Налаштування параметрів ситуацій")
        button3.setGeometry(680, 160, 250, 50)

        button3 = QPushButton(widget)
        button3.setText("Журнал подій")
        button3.setGeometry(680, 440, 250, 50)

        cb = QComboBox(widget)
        cb.addItem(gCameraContainer.getItems())
        cb.setGeometry(10, 505, 400, 30)

        label1 = QLabel(widget)
        label1.setText("Не додано жодної камери. \nБудь ласка, додайте камери
у меню \"Налаштування Камер\"")
        label1.setGeometry(15, 15, 450, 40)

        label1.setVisible(false)

        if gCameraContainer.cameraCount() is 0:
            label1.setVisible(true)

        widget.setGeometry(50,50,950,550)
        widget.setWindowTitle("Video Surveillance System - головний екран")
        widget.show()

        elist = gParams.detectorOptions()

        tracker = ObjectTrackingManager(widget)
        with torch.no_grad():
            tracker.track(elist)

        app.exec_()
```