

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДСИСТЕМИ
ВІРТУАЛЬНИХ ДАТЧИКІВ ПРИСУТНОСТІ ДЛЯ РОЗУМНОГО БУДИНКУ
DEVELOPMENT OF SOFTWARE FOR A VIRTUAL PRESENCE SENSOR
SUBSYSTEM FOR A SMART HOME

Виконала: студентка 4 курсу, групи КНТ-132

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

СИМАК А.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ПАРХОМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

СИМАК Анжеліки Віталіївни

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення підсистеми віртуальних датчиків присутності для розумного будинку. Development of Software for a Virtual Presence Sensor Subsystem for a Smart Home

керівник проєкту (роботи) к.т.н., доцент, ПАРХОМЕНКО Анжеліка Володимирівна,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 01 червня 2026 року

3. Вихідні дані до проєкту (роботи) завдання, документація бібліотек OpenCvSharp та YOLOv8, специфікація протоколу MQTT.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдань дипломної роботи. 2. Проєктування програмного забезпечення. 3. Конструювання програмного забезпечення. 4. Тестування та експериментальне дослідження підсистеми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ПАРХОМЕНКО А.В., доцент		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	ПЗ, Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студентка

_____ Анжеліка СІМАК
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Анжеліка ПАРХОМЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 12 табл., 30 рис., 2 дод., 26 джерел.

РОЗУМНИЙ БУДИНОК, КОМП'ЮТЕРНИЙ ЗІР, ВІРТУАЛЬНИЙ ДАТЧИК, НЕЙРОННІ МЕРЕЖІ, YOLO, MQTT, ІОТ, ЕНЕРГОЕФЕКТИВНІСТЬ.

Об'єкт дослідження – процеси програмної інженерії щодо створення підсистеми віртуальних датчиків присутності для розумного будинку.

Предмет дослідження – методи та програмні засоби детекції статичної та динамічної присутності людини у приміщенні з використанням технологій комп'ютерного зору.

Мета роботи – підвищення точності детекції присутності людини в приміщеннях для зменшення нераціональних витрат електроенергії на основі розроблення програмної підсистеми віртуальних датчиків присутності з використанням нейромережевих алгоритмів.

Матеріали, методи та технічні засоби: мова C# (WPF), бібліотека OpenCvSharp, нейромережа YOLOv8, протокол MQTT, середовище розробки Visual Studio.

Результати. Створено десктопний застосунок, який обробляє відеопотік, детектує людей за допомогою нейромережі, визначає їх знаходження у заданих зонах та передає статус у систему домашньої автоматизації через протокол MQTT.

Висновки. Розроблена підсистема дозволяє замінити фізичні датчики руху однією камерою, вирішуючи проблему детекції статичних об'єктів та зменшуючи нераціональні витрати електроенергії.

Галузь використання – системи автоматизації жилих приміщень, офісів, будівель.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 12 tables, 30 figures, 2 appendixes, 26 sources.

SMART HOME, COMPUTER VISION, VIRTUAL SENSOR, NEURAL NETWORKS, YOLO, MQTT, IoT, ENERGY EFFICIENCY.

Object of the study is software engineering processes for creating a subsystem of virtual presence sensors for a smart home.

Subject of the study is methods and software tools for detecting static and dynamic human presence in a room using computer vision technologies.

The purpose of the work is to improve the accuracy of human presence detection in rooms to reduce wasteful electricity consumption by developing a software subsystem of virtual presence sensors using neural network algorithms.

Materials, methods, and technical tools: C# (WPF), OpenCvSharp library, YOLOv8 neural network, MQTT protocol, Visual Studio development environment.

Results. A desktop application was created that processes a video stream, detects people using a neural network, determines their location within specified zones, and transmits the status to the home automation system via MQTT protocol.

Conclusions. The developed subsystem allows replacing physical motion sensors with a single camera, solving the problem of detecting static objects and reducing unnecessary energy consumption.

The field of application is automation systems for residential premises, offices, and buildings.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області та постановка завдань дипломної роботи	10
1.1 Аналіз проблем енергоефективності в сучасних системах домашньої автоматизації.....	10
1.2 Порівняльний аналіз методів детекції присутності.....	11
1.3 Аналіз алгоритмів комп'ютерного зору для задач моніторингу	12
1.4 Аналіз аналогічних програмних рішень	15
1.5 Обґрунтування вибору платформи домашньої автоматизації.....	20
1.6 Постановка завдань дипломної роботи.....	21
2 Проєктування програмного забезпечення	22
2.1 Аналіз вимог до програмної підсистеми	22
2.2 Моделювання прецедентів використання	23
2.3 Проєктування архітектури програмного забезпечення.....	24
2.4 Обґрунтування вибору інструментарію розробки.....	25
2.5 Висновки до розділу 2	31
3 Конструювання програмного забезпечення	32
3.1 Розробка алгоритмів функціонування підсистеми	32
3.2 Програмна реалізація класів та модулів	37
3.3 Опис основних функцій та структур даних підсистеми	40
3.4 Розробка графічного інтерфейсу користувача	44
3.5 Висновки до розділу 3	47
4 Тестування та експериментальне дослідження підсистеми	49
4.1 Керівництво користувача та налаштування підсистеми	49
4.2 Тестування функціональності та мережевої інтеграції з платформою автоматизації.....	52
4.3 Експериментальне дослідження розробленої підсистеми	54

4.4 Висновки до розділу 4	58
Висновки	59
Перелік джерел посилання	61
Додаток А Текст програми.....	63
Додаток Б Слайди презентації	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CNN	– Convolutional Neural Network;
CPU	– Central Processing Unit;
DNN	– Deep Neural Network;
FPS	– Frames Per Second;
GUI	– Graphical User Interface;
HOG	– Histogram of Oriented Gradients;
HTTP	– HyperText Transfer Protocol;
IoT	– Internet of Things;
mAP	– Mean Average Precision;
MJPEG	– Motion JPEG;
MQTT	– Message Queuing Telemetry Transport;
ONNX	– Open Neural Network Exchange;
PIR	– Passive Infrared;
ROI	– Region of Interest;
SMTP	– Simple Mail Transfer Protocol;
SVM	– Support Vector Machine;
UML	– Unified Modeling Language;
WPF	– Windows Presentation Foundation;
YOLO	– You Only Look Once;
НВЧ	– надвисокочастотний;
ООП	– об'єктно-орієнтоване програмування;
ПЗ	– програмне забезпечення.

ВСТУП

Актуальність теми. В умовах енергетичної кризи в Україні раціональне використання електроенергії є критично важливим завданням [1]. Системи класу Smart Home дозволяють автоматизувати управління освітленням та кліматом, проте їх ефективність залежить від точності датчиків присутності [1]-[2]. Традиційні пасивні інфрачервоні сенсори реагують лише на активний рух і часто помилково вимикають прилади, коли людина перебуває у стані спокою, що знижує комфорт та енергоефективність [3]-[4]. Вирішенням проблеми є застосування технологій комп'ютерного зору [3].

Альтернативні рішення, зокрема мікрохвильові датчики (mmWave), забезпечують вищу точність визначення присутності, проте характеризуються більшою вартістю та складністю інтеграції [5].

Перспективним підходом до вирішення зазначеної проблеми є використання методів комп'ютерного зору та глибокого навчання [6]. На відміну від класичних детекторів руху, алгоритми Object Detection виконують семантичний аналіз відеозображення та дозволяють визначати наявність людини у кадрі незалежно від її активності [7].

Використання нейронних мереж сприяє підвищенню точності визначення присутності та розширенню функціональних можливостей систем домашньої автоматизації, оскільки дозволяє використовувати одну камеру для точної детекції статичних та динамічних об'єктів, замінюючи набір апаратних сенсорів [8]-[9]. Тому тема дослідження є актуальною.

Метою роботи є підвищення точності детекції присутності людини в приміщеннях для зменшення нераціональних витрат електроенергії на основі розроблення програмної підсистеми віртуальних датчиків присутності з використанням нейромережових алгоритмів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДИПЛОМНОЇ РОБОТИ

1.1 Аналіз проблем енергоефективності в сучасних системах домашньої автоматизації

Концепція «Розумного будинку» (Smart Home) сьогодні виходить за межі простої автоматизації побутових процесів. В умовах глобального зростання цін на енергоносії та, зокрема, дефіциту генеруючих потужностей в Україні, пріоритетним завданням таких систем стає енергоменеджмент. Основною статтею витрат електроенергії у житлових та офісних приміщеннях, окрім роботи потужних приладів, є нераціональне використання освітлення та систем клімат-контролю (опалення, кондиціонування) у порожніх приміщеннях [4]-[11].

Існуючі системи домашньої автоматизації (Home Automation) можна класифікувати за типом управління та рівнем інтелектуальності (рис. 1.1).



Рисунок 1.1 – Класифікація типів управління в системах домашньої автоматизації [4]-[5]

Найбільш ефективним підходом є подієве керування, де система приймає рішення про вимкнення навантаження на основі даних про наявність людини в приміщенні. Однак надійність такої системи критично залежить від

точності сенсорів. Помилкове визначення відсутності людини (False Negative) призводить до вимкнення світла, коли користувач перебуває в кімнаті, що викликає дискомфорт. Помилкове визначення присутності (False Positive) призводить до марних витрат енергії [5]-[12].

1.2 Порівняльний аналіз методів детекції присутності

Для вирішення задачі виявлення людини в приміщенні використовуються різні фізичні принципи. Нижче наведено аналіз існуючих апаратних рішень для виявлення їхніх недоліків.

1.2.1 Піроелектричні інфрачервоні датчики

PIR-сенсори (від англ. Passive Infrared, PIR) є стандартом де-факто в індустрії завдяки низькій вартості та простоті. Принцип їх роботи базується на фіксації зміни теплового випромінювання при переміщенні об'єкта в зоні чутливості (лінзи Френеля).

Недолік: датчик реагує виключно на диференціал (зміну) температури в просторі. Якщо людина сидить на дивані нерухомо (статична поза) або рухається мінімально (робота за комп'ютером), датчик «втрачає» об'єкт. Це вимагає встановлення великих тайм-аутів на вимкнення (15–30 хвилин), що знижує енергоефективність [5]-[13].

1.2.2 Мікрохвильові (радарні) та ультразвукові датчики

Активні сенсори, що працюють на ефекті Доплера. Вони випромінюють хвилі та аналізують частоту відбитого сигналу.

Недолік: висока чутливість до перешкод (рух штор від вітру, вентилятори), можливий негативний вплив на здоров'я (для потужних

надвисокочастотних, НВЧ-датчиків) та хибні спрацювання через тонкі стіни [4]-[14].

Порівняльний аналіз методів детекції присутності, узагальнений на основі джерела [6], наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння методів детекції присутності [6]

Критерій порівняння	PIR-датчики	Ультразвукові датчики	Відеоаналітика (Computer Vision)
Детекція руху	Відмінно	Відмінно	Відмінно
Детекція статичності	Неможливо	Погано (потрібен мікрорух)	Відмінно (за візуальними ознаками)
Класифікація об'єкта	Ні (реагує на тварин)	Ні	Так (Людина / Тварина)
Зонування	Потрібен окремий датчик на кожну зону	Складно	Програмні зони (Region of Interest, ROI) на одному кадрі
Вартість обладнання	Низька	Середня	Середня (наявність камери)

Як видно з таблиці 1.1, використання відеоаналітики дозволяє усунути головний недолік апаратних датчиків – нездатність детектувати статичні об'єкти та розрізняти людей і домашніх тварин.

1.3 Аналіз алгоритмів комп'ютерного зору для задач моніторингу

Комп'ютерний зір (Computer Vision) пропонує кілька підходів до вирішення задачі детекції людей. Еволюція цих методів включає наступні підходи: метод віднімання фону, метод гістограм орієнтованих градієнтів, згорткові нейронні мережі та архітектура одноетапної детекції.

1.3.1 Метод віднімання фону

Найпростіший алгоритм, відомий як метод віднімання фону (від англ. Background Subtraction), який порівнює поточний кадр з попереднім або з усередненою моделлю фону. Якщо пікселі змінилися – фіксується рух.

Проблема: цей метод, по суті, дублює роботу PIR-датчика. Він не бачить нерухому людину (вона стає частиною фону) і дає хибні спрацювання на зміну освітлення (тіні від хмар, увімкнення лампи). Для нашої задачі він не підходить [15].

1.3.2 Метод гістограм орієнтованих градієнтів

Класичний підхід машинного навчання, що поєднує метод гістограм орієнтованих градієнтів (від англ. Histogram of Oriented Gradients, HOG) та метод опорних векторів (від англ. Support Vector Machine, SVM), шукає силуети людей за контурами.

Проблемою даного підходу є висока обчислювальна складність та погана робота при частковому перекритті об'єкта (наприклад, людина сидить за столом і видно тільки верхню частину тіла) [16].

1.3.3 Згорткові нейронні мережі та архітектура одноетапної детекції

Сучасний підхід базується на згорткових нейронних мережах (від англ. Convolutional Neural Networks, CNN) та глибокому навчанні (Deep Learning). Архітектура одноетапної детекції YOLO (від англ. You Only Look Once) стала значним етапом розвитку методів розпізнавання об'єктів у цій сфері, дозволяючи виконувати детекцію в реальному часі [7]. На відміну від методів, що сканують зображення вікном ковзання, алгоритми YOLO розглядають весь кадр цілком, розбиваючи його на сітку і передбачаючи

обмежувальні рамки (Bounding Boxes) та ймовірності класів для кожної комірки [17]-[18].

Для обґрунтування вибору алгоритму детекції було проведено порівняльний аналіз альтернативних методів комп'ютерного зору для роботи на пристроях без спеціалізованих графічних прискорювачів (використання лише центрального процесора – Central Processing Unit, CPU). Узагальнені результати порівняння методів, відповідно до [6]-[7], подано в таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз алгоритмів детекції об'єктів [6]-[7]

Критерій порівняння	HOG + SVM	MobileNet SSD	YOLOv8 Nano
Швидкодія на CPU (Frames Per Second, FPS)	Низька (1-5 FPS)	Середня (10-15 FPS)	Висока (15-30 FPS)
Точність детекції (mean Average Precision, mAP)	Низька	Середня	Висока
Стійкість до перекриття	Дуже низька	Середня	Висока
Ресурсомісткість	Високі вимоги	Низькі	Низькі (оптимізовано)
Складність у .NET	Низька (OpenCV)	Середня	Середня (через ONNX)

Як свідчить аналіз існуючих підходів, наведений у таблиці 1.2, архітектура YOLOv8 (модифікація Nano) забезпечує найкращий баланс між швидкістю обробки відеопотоку на центральному процесорі та високою точністю розпізнавання.

Переваги використання YOLOv8 для розроблюваної підсистеми:

- швидкодія: оптимізація для роботи на CPU дозволяє запускати підсистему на звичайному ноутбучі без дорогої відеокарти;
- точність: висока ймовірність розпізнавання класу "Person" навіть у складних позах та при частковому перекритті;
- універсальність: можливість отримання координат центру об'єкта для програмної реалізації логіки "зон" (віртуальних датчиків) [10]-[19].

Приклад візуалізації результатів роботи алгоритму YOLOv8 (знаходження об'єктів різних класів та виділення їх обмежувальними рамками на вулиці) наведено на рисунку 1.2.

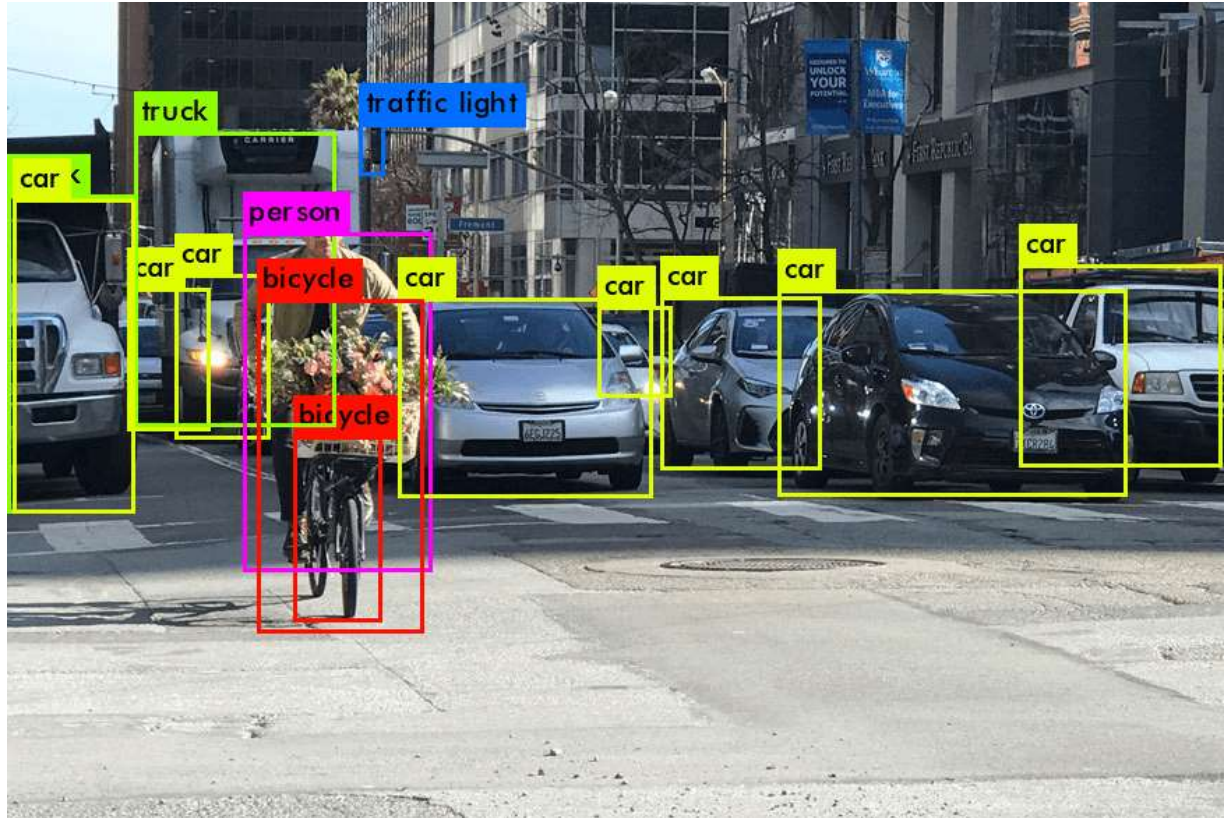


Рисунок 1.2 – Приклад детекції та класифікації об'єктів за допомогою алгоритму YOLO [8]

1.4 Аналіз аналогічних програмних рішень

Для обґрунтування доцільності розробки було проведено аналіз існуючих комерційних та відкритих (Open Source) систем-аналогів у сфері домашньої автоматизації, які вирішують проблему детекції присутності. Більшість класичних рішень покладаються на апаратні сенсори, проте на ринку з'являються системи, що використовують мікрохвилі або комп'ютерний зір.

1.4.1 Комерційні апаратні рішення

Типовим прикладом сучасного комерційного підходу є система Aqara FP2 (рис. 1.3), яка є одним із провідних представників класу радарних датчиків міліметрових хвиль (від англ. millimeter-wave, mmWave). На відміну від класичних інфрачервоних сенсорів, ця технологія базується на безперервному випромінюванні високочастотних радіохвиль (на частоті близько 60 ГГц) та аналізі їх відбиття від об'єктів у просторі. Такий підхід дозволяє пристрою будувати карту приміщення та відстежувати координати людини у реальному часі [7].



Рисунок 1.3 – Інтерфейс налаштування зон у додатку Aqara FP2 [9]

Переваги цього рішення:

- надзвичайно висока точність виявлення мікрорухів, що дозволяє фіксувати присутність людини навіть за її диханням або серцебиттям;
- підтримка програмного зонування приміщення у фірмовому додатку (можливість налаштування до 30 незалежних зон моніторингу простору).

Недоліки подібних апаратних систем:

- висока вартість обладнання (близько 80–100 доларів за один датчик), що робить масове впровадження у багатокімнатних приміщеннях фінансово затратним;
- замкнута екосистема виробника, яка вимагає обов'язкового використання фірмових хмарних сервісів або застосування складних неофіційних методів для інтеграції з платформою локальної автоматизації;
- схильність до хибних спрацювань на нецільові рухомі об'єкти (наприклад, коливання штор від протягу або обертання лопатей вентиляторів) [9].

1.4.2 Програмні комплекси з відкритим вихідним кодом

Яскравим представником цього напрямку є програмний комплекс Frigate (рис. 1.4), який належить до систем мережевого відеоспостереження (від англ. Network Video Recorder, NVR) з відкритим вихідним кодом (від англ. Open Source). Архітектура подібних рішень базується на багатокроковому конвеєрі обробки даних: спочатку виконується апаратне декодування відеопотоку, потім застосовуються класичні алгоритми виявлення руху пікселів, і лише після цього кадри з підозрілим рухом передаються до тензорного процесора для остаточної класифікації об'єктів за допомогою глибоких згорткових нейромереж [12]–[20].

Переваги подібних систем:

- відкритий вихідний код, локальна обробка відеоданих (без використання хмарних серверів) та гнучкість налаштувань конвеєра;
- використання потужних нейромереж для класифікації десятків типів об'єктів (людина, кіт, автомобіль тощо);
- підтримка протоколу Message Queuing Telemetry Transport (MQTT) для безшовної передачі подій детекції у платформи домашньої автоматизації.

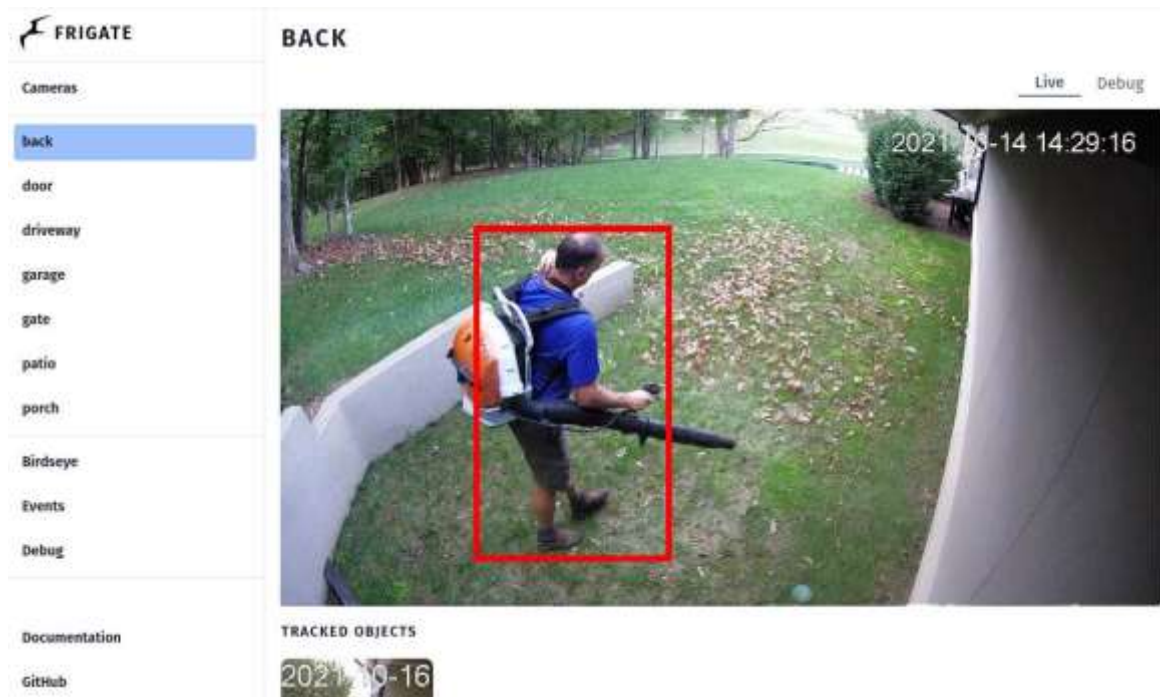


Рисунок 1.4 – Інтерфейс детекції об'єктів у системі Frigate NVR [10]

Недоліки:

- надзвичайно високі вимоги до апаратного забезпечення (для забезпечення стабільної детекції в реальному часі без перевантаження центрального процесора вимагається обов'язкова наявність спеціалізованого апаратного прискорювача, наприклад, Google Coral TPU – Tensor Processing Unit);
- високий поріг входу та складність конфігурації – налаштування зон інтересу, масок та параметрів камер виконується виключно через редагування текстових конфігураційних файлів (у форматі YAML), оскільки зручний графічний візуальний редактор (Graphical User Interface, GUI) для цих завдань відсутній [10].

Узагальнюючи проведений огляд, можна зробити висновок, що наявні на ринку рішення мають суттєві обмеження для масового впровадження в системах розумного будинку. Комерційні радарні сенсори забезпечують зручність, але є економічно не вигідними при масштабуванні на декілька кімнат та часто залежать від сторонніх хмарних сервісів.

З іншого боку, існуючі відкриті системи відеоаналітики орієнтовані переважно на професійне відеоспостереження та запис архівів (NVR), а не на легке керування автоматизацією. Відсутність візуальних інструментів налаштування зон та високі вимоги до обчислювальних потужностей (необхідність придбання дефіцитних апаратних прискорювачів) роблять їх надто складними для пересічного користувача.

Тому існує обґрунтована потреба у розробленні проміжного рішення, яке б поєднувало точність розпізнавання нейромереж із низькими системними вимогами (робота на звичайному центральному процесорі) та простотою налаштування через зручний графічний інтерфейс.

Порівняльний аналіз існуючих рішень детекції присутності, виконаний на основі джерел [9]-[10], наведено в таблиці 1.3.

Таблиця 1.3 – Порівняльний аналіз систем детекції присутності [9]-[10]

Критерій порівняння	Aqara FP2 (Комерційна)	Frigate NVR (Open Source)	Розроблювана підсистема
Технологія детекції	Радіохвильовий радар (mmWave)	Комп'ютерний зір (камери)	Комп'ютерний зір (камери)
Визначення статичної пози	Так	Так	Так
Можливість зонування	Так (через закритий додаток)	Так (через YAML файли)	Так (через власний GUI редактор)
Інтеграція з OpenHAB	Обмежена (потребує хмари)	Складна	Нативна та проста (через MQTT)
Головний недолік	Висока ціна, хибні спрацювання	Високі вимоги до "заліза"	Залежність від освітлення

1.5 Обґрунтування вибору платформи домашньої автоматизації

Для інтеграції розроблюваної підсистеми віртуальних датчиків присутності в екосистему розумного будинку необхідно обрати платформу автоматизації. На сучасному ринку представлено багато рішень, серед яких найбільш популярними є OpenHAB, Home Assistant та Apple HomeKit. Для обґрунтованого вибору платформи було проведено їх порівняльний аналіз. Результати порівняння за ключовими для даної роботи критеріями наведено в таблиці 1.4 [21].

Таблиця 1.4 – Порівняльний аналіз платформ домашньої автоматизації

Критерій порівняння	OpenHAB	Home Assistant	Apple HomeKit
Відкритість вихідного коду	Так (Open Source)	Так (Open Source)	Ні (Пропрієтарна)
Підтримка протоколу MQTT	Повна (нативна)	Повна	Обмежена (через мости)
Вимоги до екосистеми	Немає обмежень	Немає обмежень	Прив'язка до Apple
Архітектура та мова ядра	Гнучка, на базі Java	На базі Python	Закрита
Локальне виконання	Так	Так	Так

Як видно з таблиці 1.4, платформа Apple HomeKit не підходить для вирішення поставленого завдання через свою закритість та обмежену підтримку стандартизованого протоколу MQTT. Платформи OpenHAB та Home Assistant мають схожі потужні можливості. Однак OpenHAB має архітектуру на базі Java, що забезпечує високу стабільність роботи на різноманітних апаратних платформах, а також пропонує надзвичайно гнучку систему створення правил (Rules) та нативну інтеграцію з MQTT-брокером. З огляду на це, для інтеграції розроблюваної підсистеми обрано платформу OpenHAB.

1.6 Постановка завдань дипломної роботи

На основі проведеного аналізу предметної області можна зробити висновок, що існуючі апаратні засоби не забезпечують достатньої точності для енергоефективного управління розумним будинком через проблему детекції статичних об'єктів. Використання комп'ютерного зору на базі нейромережі YOLO є перспективним рішенням, яке дозволяє замінити фізичні датчики програмними алгоритмами. Тому тема дослідження є актуальною.

Метою роботи є підвищення точності детекції присутності людини в приміщеннях для зменшення нераціональних витрат електроенергії на основі розроблення програмної підсистеми віртуальних датчиків присутності з використанням нейромережевих алгоритмів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі методи детекції присутності та їхні недоліки в контексті раціонального використання енергоресурсів;
- сформулювати вимоги до програмного забезпечення підсистеми та обґрунтувати вибір інструментарію розроблення;
- розробити алгоритмічне забезпечення та виконати програмну реалізацію підсистеми віртуальних датчиків присутності;
- провести тестування та експериментальне дослідження точності й енергоефективності розробленого рішення.

Отже, аналіз предметної області та існуючих технічних рішень показав наявність обмежень у застосуванні традиційних апаратних датчиків присутності та складність використання існуючих систем відеоаналітики. Це зумовлює доцільність розробки програмного забезпечення (ПЗ) підсистеми віртуальних датчиків присутності на основі методів комп'ютерного зору, що забезпечить підвищення точності визначення присутності людини та ефективну інтеграцію із системами автоматизації розумного будинку.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Проєктування програмного ПЗ є ключовим етапом розробки, який визначає архітектуру, функціональність та надійність майбутньої підсистеми. У цьому розділі проведено аналіз вимог, моделювання прецедентів використання та обґрунтування вибору технологічного стеку для реалізації підсистеми віртуальних датчиків присутності [11]-[22].

2.1 Аналіз вимог до програмної підсистеми

Проєктована підсистема призначена для автоматизації процесів керування розумним будинком шляхом розпізнавання присутності людей у заданих зонах відеоспостереження. На основі аналізу предметної області та технічного завдання, сформулюємо вимоги до ПЗ.

Функціональні вимоги:

- підсистема повинна отримувати відеопотік з локальної USB-камери або IP-камери;
- підсистема повинна надавати користувачу графічний інтерфейс для налаштування зон інтересу (ROI – Region of Interest) шляхом малювання прямокутників на зображенні;
 - підсистема повинна виконувати детекцію об'єктів класу «Людина» (Person) у реальному часі з використанням нейромережі;
 - підсистема повинна визначати входження центру виявленого об'єкта в одну з налаштованих зон;
 - підсистема повинна передавати статус кожної зони (активна/неактивна) на MQTT-брокер для інтеграції з платформою OpenHAB [22].

Нефункціональні вимоги:

- швидкодія: обробка не менше 10 кадрів за секунду (FPS) на процесорі середньої потужності (CPU);

- точність: поріг довіри (confidence threshold) не нижче 0,5 з можливістю налаштування;
- відмовостійкість (стабільність): автоматичне відновлення з'єднання з камерою та MQTT-брокером у разі обриву зв'язку.

Виконання цих вимог дозволить забезпечити надійну роботу підсистеми в реальних умовах експлуатації [12]-[23].

2.2 Моделювання прецедентів використання

Для формалізації функціональних вимог розроблено діаграму прецедентів (Use Case Diagram) уніфікованою мовою моделювання (Unified Modeling Language, UML) [24]-[25]. Вона відображає взаємодію основних акторів з підсистемою та внутрішню логіку обробки даних (рис. 2.1).

Основними акторами є:

- адміністратор підсистеми: виконує налаштування параметрів підключення, конфігурацію зон інтересу (ROI) та запуск процесу моніторингу;
- камера (відеопотік): зовнішнє джерело даних, що забезпечує підсистему зображенням у реальному часі;
- система OpenHAB: зовнішня платформа розумного будинку, що отримує сформовані події про присутність через протокол MQTT.

На діаграмі також відображено залежності між прецедентами: відношення <<include>> показують обов'язкові етапи алгоритму (наприклад, детекція об'єктів), а <<extend>> – умовні розширення (наприклад, фільтрація "мерехтіння" статусу) [25].

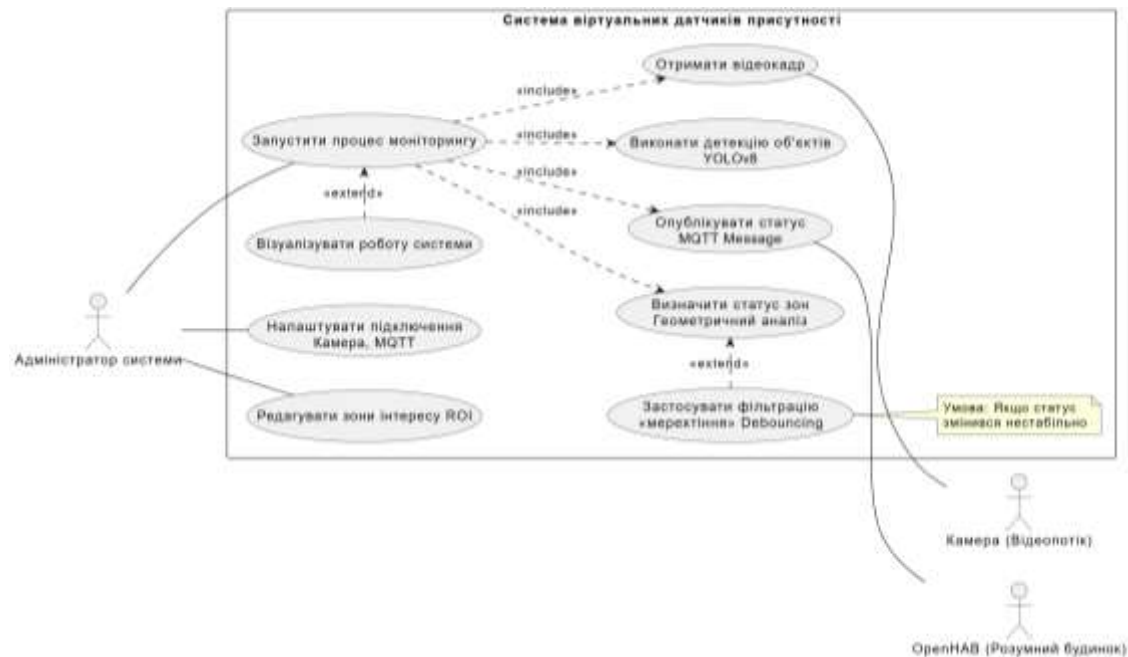


Рисунок 2.1 – Діаграма прецедентів використання розробленого ПЗ

2.3 Проектування архітектури програмного забезпечення

Архітектура ПЗ базується на принципах модульної взаємодії із зовнішнім середовищем та слабкої зв'язаності компонентів. Розроблювана підсистема виконує функцію проміжного програмного компонента між відеоджерелом та платформою автоматизації (Intelligent Gateway), що отримує "сирі" дані з відеосенсорів та передає оброблені події в систему домашньої автоматизації.

Оскільки OpenHAB є платформою високого рівня, а не просто контролером, взаємодія з нею організована через проміжний шар обміну повідомленнями. Загальна високорівнева архітектура рішення, що відображає потоки даних між компонентами, наведена на рисунку 2.2 [2]-[26].

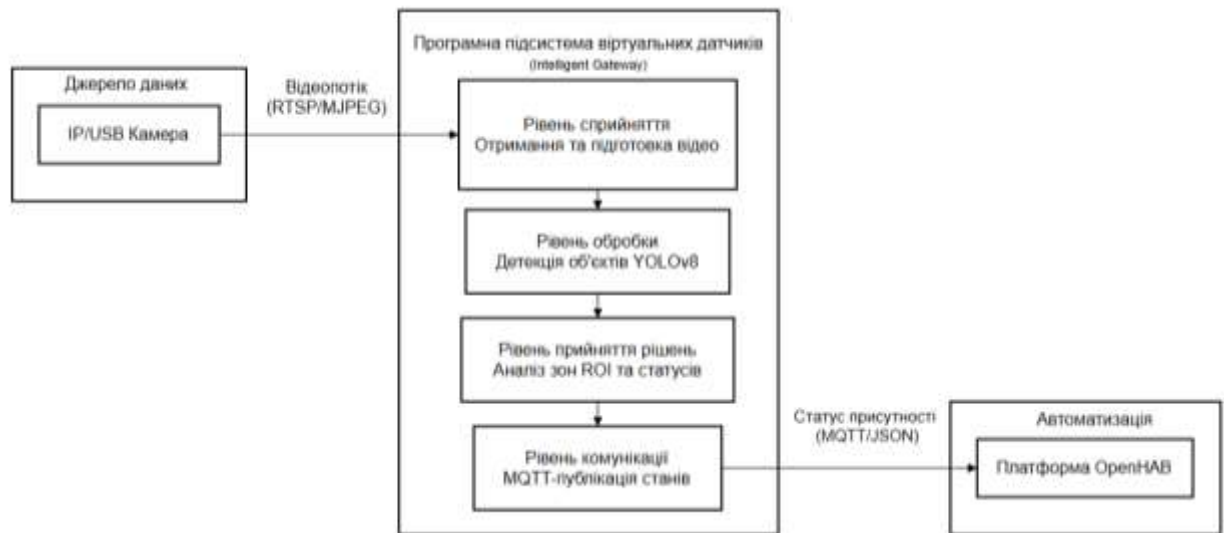


Рисунок 2.2 – Узагальнена архітектура ПЗ

Архітектура підсистеми побудована за ієрархічним принципом і може бути представлена у вигляді таких логічних рівнів:

- рівень сприйняття: відповідає за отримання відеопотоку від фізичних пристроїв (камер) та його попередню підготовку;
- рівень обробки: ядро підсистеми, що містить модуль штучного інтелекту для розпізнавання образів та класифікації об'єктів;
- рівень прийняття рішень: виконує аналіз сцени, зіставляючи координати об'єктів із визначеними зонами інтересу;
- рівень комунікації: забезпечує асинхронну взаємодію з платформою автоматизації через брокер повідомлень.

2.4 Обґрунтування вибору інструментарію розробки

Для програмної реалізації підсистеми віртуальних датчиків було проведено аналіз сучасних технологій та обрано оптимальний технологічний стек, який забезпечує баланс між швидкістю розробки, продуктивністю та сумісністю з апаратним забезпеченням «Розумного будинку».

2.4.1 Вибір та обґрунтування мови програмування та платформи розробки

Під час вибору базової мови програмування та платформи для розробки клієнтського додатка було розглянуто кілька альтернативних варіантів: Python, C++ та C#. Головними критеріями вибору були швидкість створення графічного інтерфейсу (GUI) для налаштування зон інтересу (ROI), продуктивність обробки відеопотоку та зручність реалізації багатопотоковості.

Порівняльну оцінку розгляданих засобів розробки, відповідно до джерела [11], наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння технологічних рішень [11]

Критерій порівняння	Python	C++ (Qt / ImGui)	C# (.NET WPF)
Швидкодія виконання коду	Низька (інтерпретована)	Дуже висока	Висока (завдяки JIT-компіляції)
Швидкість розробки інтерфейсу (GUI)	Середня (Tkinter / PyQt)	Низька (вимагає часу)	Висока (XAML / WPF)
Інтеграція з OpenCV та неймережами	Відмінна (нативна)	Відмінна (нативна)	Добра (через OpenCvSharp)
Реалізація багатопотоковості	Середня (GIL обмежує)	Складна	Висока (async/await)

Результати порівняльного аналізу свідчать, що мова Python, попри широкі можливості у сфері Data Science, має критичні обмеження для даної задачі. Зокрема, механізм GIL (Global Interpreter Lock) ускладнює ефективну паралелізацію процесів обробки відео та мережевої взаємодії, що може призводити до затримок у передачі команд.

Мова C++ забезпечує найвищу продуктивність, однак вимагає значно більших часових витрат на розробку користувацького інтерфейсу та ручне

керування пам'яттю, що є нераціональним в умовах обмеженого часу на проєктування.

Відтак, C# та платформа .NET визначено як оптимальний компроміс. Вибір цього стеку обґрунтований трьома ключовими факторами.

По-перше, наявність підсистеми WPF (Windows Presentation Foundation) дозволяє швидко реалізувати ергономічний інтерфейс для малювання полігонів (ROI) поверх відеопотоку без втрати продуктивності.

По-друге, середовище виконання CLR (Common Language Runtime) завдяки JIT-компіляції (Just-In-Time) демонструє швидкість виконання коду, наближену до нативної, що є критичним для задач Computer Vision [11].

По-третє, використання асинхронної моделі `async/await` значно спрощує архітектуру неблокуючого обміну даними з MQTT-брокером, забезпечуючи стабільність роботи додатка [11]-[14].

2.4.2 Вибір та обґрунтування бібліотеки комп'ютерного зору

Для маніпуляцій із відеопотоком обрано бібліотеку OpenCvSharp. Це кросплатформна обгортка (wrapper) для бібліотеки OpenCV. Вибір саме цього інструменту, на противагу стандартним засобам .NET (System.Drawing) або альтернативі у вигляді EmguCV, зумовлений архітектурними особливостями.

Порівняльний аналіз найбільш поширених бібліотек комп'ютерного зору для платформи .NET наведено в таблиці 2.2 [13].

OpenCvSharp забезпечує прямий доступ до нативного коду C++, що гарантує високу швидкість виконання алгоритмів обробки зображень. На відміну від EmguCV, обрана бібліотека має більш «легку» структуру та простішу модель ліцензування (Apache 2.0 / BSD), що спрощує її використання в академічних та комерційних проєктах.

Таблиця 2.2 – Порівняння бібліотек комп'ютерного зору [13]

Критерій	OpenCvSharp	EmguCV	AForge.NET
Продуктивність	Висока (прямий доступ до C++)	Середня (багато абстракцій)	Низька (застаріла)
Ліцензія	Apache 2.0 (Вільна)	GPL / Комерційна	LGPL
Підтримка DNN	Повна (через OpenCV)	Часткова	Відсутня
Складність інтеграції	Низька (NuGet пакет)	Середня (залежності)	Низька

Важливим аспектом є вбудована підтримка модуля DNN (Deep Neural Networks). Це дозволяє завантажувати нейромережеві моделі у форматі ONNX (Open Neural Network Exchange) безпосередньо в оперативну пам'ять та виконувати інференс без необхідності підключення важких фреймворків [13]. Також засоби бібліотеки дозволяють ефективно візуалізувати роботу «віртуальних датчиків», малюючи контури зон та рамки детекції безпосередньо на матриці кадру (Mat) [6]-[16].

2.4.3 Вибір та обґрунтування нейромережі

Вибір архітектури нейромережі є ключовим етапом проектування, оскільки від нього залежить точність детекції присутності людини та відсутності хибних спрацювань. У ході дослідження було проаналізовано архітектури SSD (Single Shot Detector), MobileNet-SSD та алгоритми сімейства YOLO.

Порівняння характеристик розглянутих архітектур нейронних мереж для роботи на центральному процесорі (CPU) подано в таблиці 2.3 [6].

Згідно з порівняльним аналізом [6], алгоритми SSD часто демонструють недостатню точність при детекції дрібних об'єктів. Водночас MobileNet-SSD, хоч і має низьке обчислювальне навантаження, поступається

сучасним версіям YOLO за показником mAP (mean Average Precision) та стабільністю роботи в умовах змінного освітлення [9]-[11].

Таблиця 2.3 – Порівняльний аналіз архітектур нейромереж [6]

Характеристика	YOLOv8 Nano	SSD MobileNet V2	Faster R-CNN
Швидкість на CPU	Висока (~20-30 FPS)	Середня (~15 FPS)	Низька (< 5 FPS)
Точність (mAP)	Висока (37.3)	Середня (22.1)	Дуже висока
Розмір моделі	Дуже малий (~6 MB)	Малий (~20 MB)	Великий (>100 MB)
Детекція малих об'єктів	Добра	Посередня	Відмінна

Для реалізації підсистеми обрано архітектуру YOLOv8 (модифікація Nano – yolov8n) у форматі ONNX. Цей вибір спирається на три стратегічні переваги.

Вирішення проблеми статичних об'єктів. Як зазначається у дослідженні [5], головним недоліком класичних PIR-датчиків та алгоритмів Motion Detection є неможливість виявлення нерухомої людини. YOLOv8 виконує повноцінну класифікацію об'єктів, тому підсистема продовжує фіксувати присутність людини, навіть якщо вона спить або читає, що кардинально підвищує точність та безперебійність автоматизації.

Ефективність на CPU. Версія Nano є найбільш оптимізованою в лінійці YOLO [3]. Тестування показують [12], що ця модель дозволяє досягати швидкості обробки 10–30 FPS на звичайних центральних процесорах, що робить підсистему економічно доцільною та придатною для побутового використання.

Універсальність. Експорт моделі у формат ONNX [14] забезпечує її повну сумісність з обраною бібліотекою OpenCvSharp [15]-[19].

2.4.4 Вибір та обґрунтування протоколу обміну даними

Для інтеграції розробленого ПЗ з екосистемою розумного будинку (зокрема, платформою OpenHAB або Home Assistant) обрано протокол MQTT (Message Queuing Telemetry Transport).

Для обґрунтування вибору було проведено порівняння MQTT з іншими поширеними протоколами передачі даних в сфері Інтернету речей (Internet of Things, IoT), результати якого наведено в таблиці 2.4 [20].

Таблиця 2.4 – Порівняльний аналіз протоколів обміну даними [15]

Критерій	MQTT	HTTP (REST)	CoAP
Архітектура	Pub/Sub (Подієва)	Request/Response	Request/Response
Накладні витрати	Мінімальні (2 байти)	Високі (Заголовки)	Низькі
Гарантія доставки	Так (QoS 0, 1, 2)	Ні	Так (Confirmable)
Придатність для realtime	Висока	Низька (Polling)	Середня

У порівнянні з класичним підходом на базі HTTP-запитів REST API – Representational State Transfer Application Programming Interface), використання MQTT для задач керування освітленням є більш обґрунтованим з огляду на специфіку IoT-систем [15].

Зокрема, вибір обумовлений такими перевагами:

- мінімальні затримки (Low Latency): використання постійного TCP-з'єднання (Transmission Control Protocol) дозволяє миттєво передавати статус присутності, уникнувши затримок, характерних для моделі HTTP-polling;
- гарантовану доставку (QoS): підтримка рівнів якості обслуговування (Quality of Service, QoS 1) гарантує доставку команди навіть

при короткочасних збоях Wi-Fi мережі, що підвищує стабільність зв'язку підсистеми [4]-[20];

– енергоефективність: мінімізація службових заголовків пакетів знижує навантаження на локальну мережу, що є важливим фактором при масштабуванні підсистеми розумного будинку [1].

2.5 Висновки до розділу 2

У розділі виконано комплексне проєктування ПЗ підсистеми віртуальних датчиків присутності. Сформульовано функціональні та нефункціональні вимоги, що визначають продуктивність, точність і стабільність роботи підсистеми в реальному часі. На основі UML-моделювання описано взаємодію користувача та платформи автоматизації з програмним модулем.

Запропонована архітектура побудована за модульним та ієрархічним принципом із розподілом на рівні сприйняття, обробки, прийняття рішень і комунікації, що забезпечує слабку зв'язаність компонентів та можливість масштабування.

Обґрунтовано вибір технологічного стеку: C#/.NET як платформи розробки, OpenCvSharp для роботи з відеопотоком, нейромережі YOLOv8n у форматі ONNX для детекції людини та протоколу MQTT для інтеграції з системами розумного будинку. Обрані рішення забезпечують необхідний баланс між швидкодією, точністю, стабільністю та сумісністю з IoT-інфраструктурою.

Отже, прийняті архітектурні та технологічні рішення відповідають поставленим вимогам і створюють основу для реалізації ефективної підсистеми автоматизації на базі комп'ютерного зору.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі виконано детальне конструювання ПЗ підсистеми віртуальних датчиків присутності. Описано алгоритми обробки відеопотоку, логіку детекції об'єктів та прийняття рішень, а також наведено програмну реалізацію класів згідно з принципами об'єктно-орієнтованого програмування. Розглянуто процес розробки графічного інтерфейсу користувача та механізми інтеграції підсистеми з платформою розумного будинку.

3.1 Розробка алгоритмів функціонування підсистеми

Для забезпечення виконання поставлених завдань було розроблено алгоритмічне забезпечення підсистеми віртуальних датчиків присутності [2]-[6]. Логіку роботи ПЗ можна розділити на три основні складові: загальний цикл обробки відеопотоку, алгоритм детекції об'єктів та алгоритм фільтрації хибних спрацювань (debouncing).

Алгоритм роботи головного вікна програми (MainWindow) відповідає за ініціалізацію компонентів, підключення до зовнішніх сервісів та безперервну обробку кадрів. Узагальнену схему цього алгоритму наведено на рисунку 3.1. Поблочний опис алгоритму роботи підсистеми наступний.

Блок 1. Початок. Початок виконання загального циклу роботи програмної підсистеми.

Блок 2. Ініціалізація GUI. Полягає у завантаженні графічного інтерфейсу користувача та відображенні головного вікна програми із завантаженими налаштуваннями віртуальних зон.

Блок 3. Ініціалізація з'єднання з MQTT. Здійснюється спроба асинхронного підключення до локального або віддаленого брокера повідомлень для подальшої інтеграції з платформою OpenHAB.

Блок 4. Підключення до камери. Відбувається звернення до обраного пристрою відеозахоплення (USB-камери або IP-поток) для ініціалізації передачі даних.

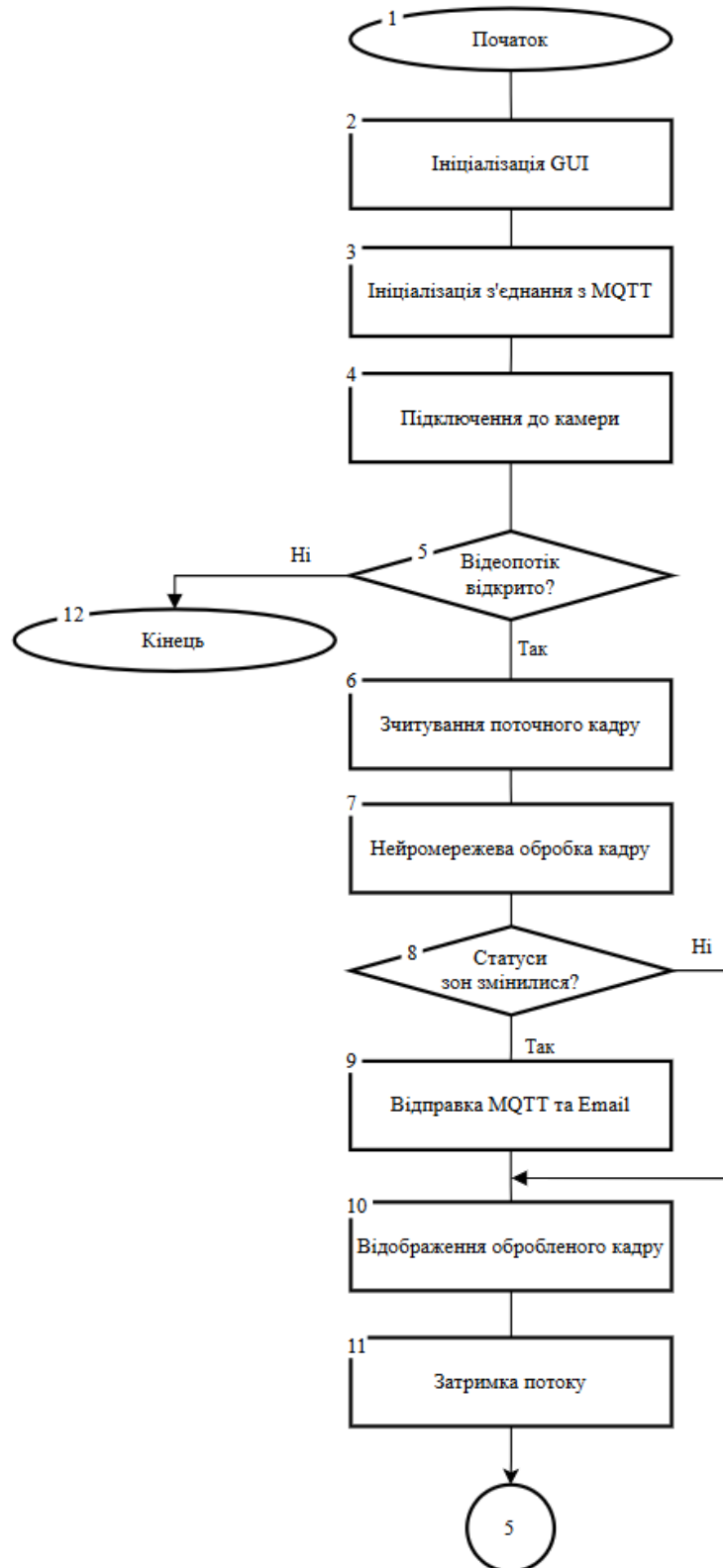


Рисунок 3.1 – Узагальнена схема алгоритму роботи підсистеми

Блок 5. Перевірка умови «Відеопотік відкрито?». Якщо камеру успішно ініціалізовано і кадри доступні для читання (Так), алгоритм переходить до блоку 6. Якщо доступ до камери втрачено або її не знайдено (Ні), алгоритм перериває цикл та переходить до завершення (блок 12).

Блок 6. Зчитування поточного кадру. З відеопотоку захоплюється один поточний кадр у вигляді матриці пікселів для подальшого аналізу.

Блок 7. Нейромережева обробка кадру. Полягає у передачі отриманого кадру до моделі YOLOv8 для виявлення об'єктів класу «Людина», розрахунку їх геометричних центрів та перевірки входження у задані зони інтересу (ROI).

Блок 8. Перевірка умови «Статуси зон змінилися?». Якщо логічний статус хоча б однієї віртуальної зони (ON/OFF) змінився порівняно з попереднім кадром (Так), алгоритм переходить до блоку 9. Якщо змін не відбулося (Ні), алгоритм уникає дублювання мережевих запитів і переходить безпосередньо до блоку 10.

Блок 9. Відправка MQTT та Email. Здійснюється публікація оновлених статусів датчиків у відповідні топіки MQTT-брокера та, за наявності відповідних налаштувань, генерується тривожне сповіщення на електронну пошту.

Блок 10. Відображення обробленого кадру. На екран користувача виводиться відеокادر з накладеними рамками детекції (Bounding Boxes) та динамічним підсвічуванням активних зон.

Блок 11. Затримка потоку. Виконується технічна пауза для синхронізації швидкості обробки зі стабільною частотою кадрів (FPS), після чого алгоритм повертається до блоку 5 для зчитування наступного кадру.

Блок 12. Кінець. Завершення роботи алгоритму та звільнення апаратних ресурсів.

Ключовим елементом підсистеми є алгоритм детекції та розрахунку входження об'єкта до налаштованих віртуальних зон (Region of Interest, ROI). Схему цього алгоритму наведено на рисунку 3.2 [10]-[15].

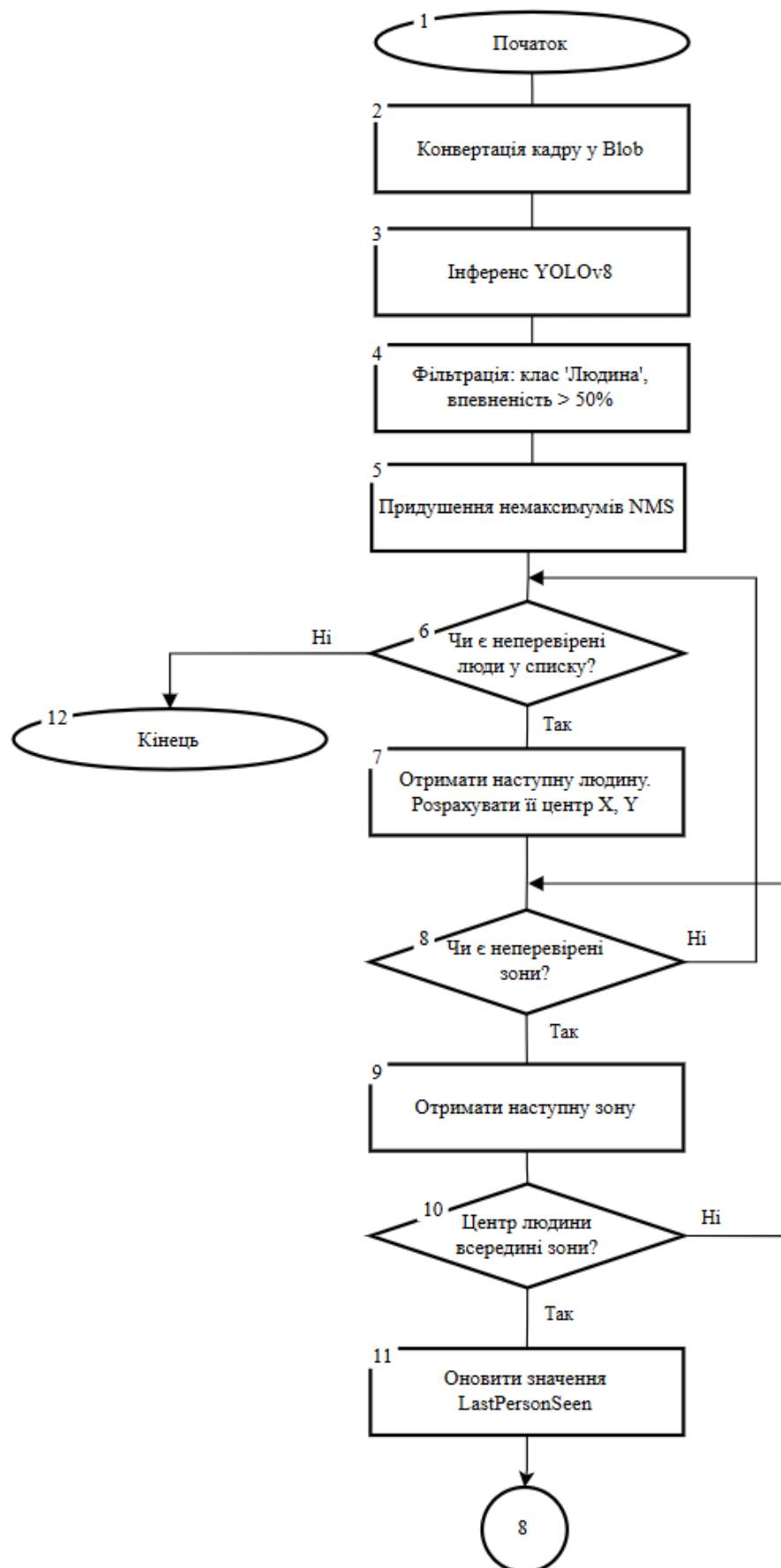


Рисунок 3.2 – Схема алгоритму детекції та логіки віртуальних датчиків

Поблочний опис алгоритму детекції та логіки віртуальних датчиків наступний.

Блок 1. Початок. Початок обробки поточного кадру відеопотоку.

Блок 2. Конвертація кадру у Blob. Полягає у перетворенні отриманої матриці пікселів (Mat) у спеціальний формат (Blob) з необхідним масштабуванням для подачі на вхід нейромережі.

Блок 3. Інференс YOLOv8. Здійснюється пряме поширення (Forward pass) тензора через згорткову нейромережу для отримання передбачень щодо знайдених об'єктів та їх координат.

Блок 4. Фільтрація: клас «Людина», впевненість $> 50\%$. З усіх знайдених об'єктів відбираються лише ті, що класифіковані як «Людина» (Person) із рівнем довіри (Confidence), який перевищує поріг 0.5.

Блок 5. Придушення немаксимумів NMS. Виконується алгоритм Non-Maximum Suppression для видалення дублюючих рамок детекції (Bounding Boxes), які вказують на одну й ту саму людину.

Блок 6. Перевірка умови «Чи є неперевірені люди у списку?». Якщо у відфільтрованому списку ще залишилися виявлені люди для аналізу (Так), алгоритм переходить до блоку 7. Якщо всіх людей перевірено або їх не знайдено (Ні), алгоритм переходить до завершення обробки кадру (блок 12).

Блок 7. Отримати наступну людину. Розрахувати її центр X , Y . Зі списку вибирається наступний об'єкт та обчислюються геометричні координати його центру на основі розмірів рамки детекції.

Блок 8. Перевірка умови «Чи є неперевірені зони?». Перевіряється, чи залишилися віртуальні датчики (зони), з якими ще не зіставлялися координати поточної людини. Якщо так (Так), відбувається перехід до блоку 9. Якщо ні (Ні), алгоритм повертається до блоку 6 для вибору наступної людини.

Блок 9. Отримати наступну зону. З колекції віртуальних датчиків вибирається наступна зона інтересу (ROI) для перевірки.

Блок 10. Перевірка умови «Центр людини всередині зони?». Виконується перевірка геометричного входження раніше розрахованих координат центру людини (X, Y) у межі поточної зони. Якщо об'єкт всередині (Так), виконується перехід до блоку 11. Якщо ні (Ні), алгоритм повертається до блоку 8 для перевірки наступної зони.

Блок 11. Оновити значення LastPersonSeen. Для поточної зони оновлюється часова мітка останньої фіксації людини (LastPersonSeen = DateTime.Now), що є основою алгоритму debouncing. Після цього алгоритм повертається до перевірки наступних зон (перехід до блоку 8).

Блок 12. Кінець. Завершення алгоритму обробки поточного кадру.

Однією з головних проблем систем комп'ютерного зору є короткочасна втрата об'єкта (наприклад, коли людина частково перекривається меблями або нахиляється) [5]. Для вирішення цієї проблеми та запобігання «мерехтінню» освітлення в розумному будинку було розроблено алгоритм фільтрації (Debouncing). Логіка алгоритму базується на відкладеному вимкненні. Статус віртуального датчика (IsActive) розраховується не за поточною наявністю людини в кадрі, а за різницею між поточним часом та часом останньої фіксації людини у цій зоні (LastPersonSeen). Якщо ця різниця не перевищує встановлений поріг (наприклад, 5 секунд), датчик залишається в стані "ON". Це дозволяє системі стабільно працювати навіть при тимчасових втратах об'єкта нейромережею, забезпечуючи комфорт користувача [4]-[13].

3.2 Програмна реалізація класів та модулів

Архітектура розробленого ПЗ базується на принципах об'єктно-орієнтованого програмування (ООП) [11]-[22]. Для забезпечення гнучкості, можливості подальшого масштабування та спрощення підтримки коду було застосовано принцип єдиної відповідальності (Single Responsibility Principle).

Згідно з цим принципом логіку графічного інтерфейсу, обробку відеопотоку та мережеву взаємодію розділено на незалежні класи (сервіси) [13]-[24].

Для деталізації внутрішньої організації ПЗ та взаємодії його основних компонентів розроблено структурну модульну схему підсистеми (рис. 3.3). Вона відображає конвеєр обробки даних: від отримання «сирого» відеопотоку до формування вихідних сигналів для зовнішніх сервісів.



Рисунок 3.3 – Структурна схема підсистеми

Згідно з наведеною схемою (рис. 3.3), робота підсистеми розділена на такі функціональні модулі:

- модуль захоплення відео – забезпечує стабільне отримання кадрів із вебкамери та їх передачу в оперативну пам'ять у форматі Mat;
- модуль детекції (YOLOv8) – виконує неймережевий аналіз кожного кадру для пошуку об'єктів класу «Людина» (Person);
- блок логіки віртуальних датчиків – центральний вузол, який зіставляє координати знайдених людей із налаштованими зонами інтересу (ROI) та керує статусами датчиків;
- сервісні модулі (MQTT, Email, HTTP) – відповідають за зовнішню комунікацію – передачу подій у OpenHAB через протокол MQTT, відправку сповіщень та трансляцію відео для віддаленого перегляду.

Діаграму класів розробленого ПЗ, що відображає структуру та зв'язки між основними модулями, наведено на рисунку 3.4 [22]-[26].

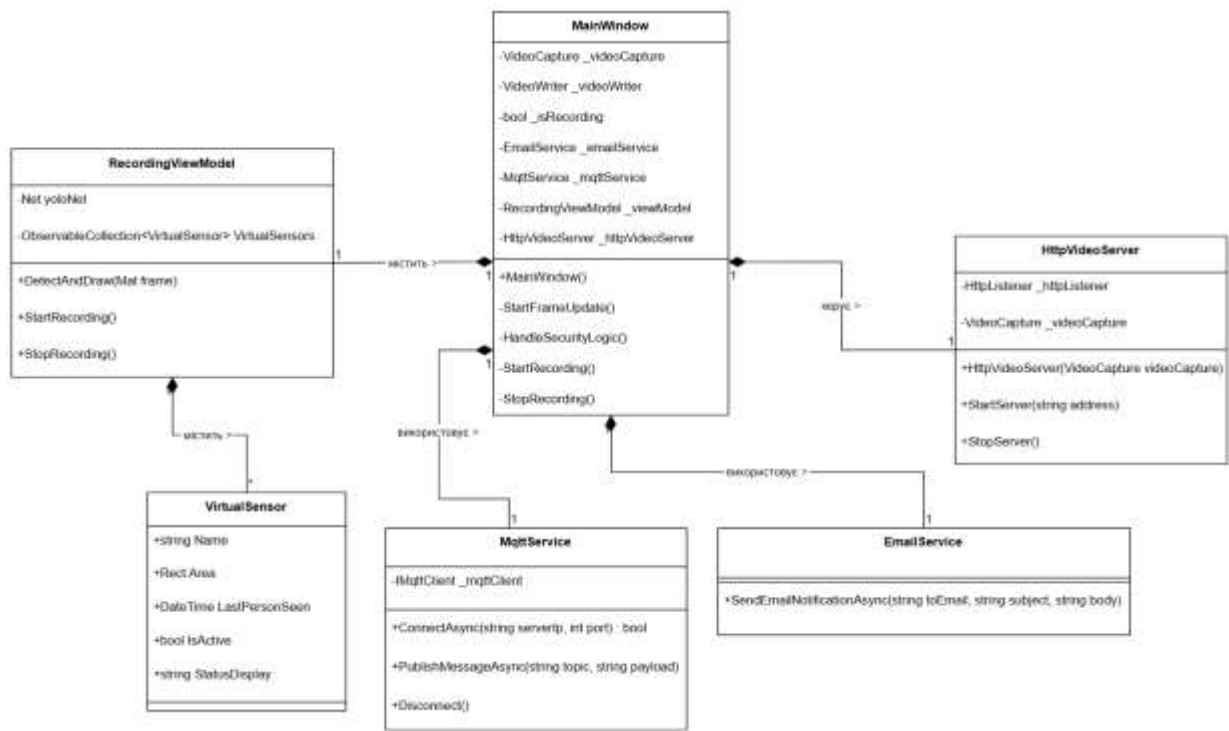


Рисунок 3.4 – Діаграма класів розробленого ПЗ

Архітектура розробленого ПЗ побудована на взаємодії низки спеціалізованих класів та сервісів, кожен з яких відповідає за окремий рівень логіки. Розглянемо детальніше призначення основних структурних компонентів підсистеми наведені у таблиці 3.1 [21]-[22].

Таблиця 3.1 – Опис основних класів та сервісів розробленої підсистеми [21]-[22]

Назва класу / модулю	Функціональне призначення
1	2
MainWindow	Головний модуль графічного інтерфейсу (GUI). Забезпечує взаємодію з користувачем, малювання зон інтересу (ROI) та управління налаштуваннями
RecordingViewModel	Модуль бізнес-логіки. Відповідає за ініціалізацію нейромережі YOLOv8, обробку відеопотоку та виконання інференсу для детекції об'єктів

Кінець таблиці 3.1

1	2
VirtualSensor	Клас, що описує логічний віртуальний датчик, зберігає його геометричні координати, поточний статус та час останньої активності людини
MqttService	Сервісний клас для асинхронної взаємодії з MQTT-брокером. Забезпечує публікацію статусів віртуальних датчиків для інтеграції з OpenHAB
EmailService	Модуль для формування та відправки тривожних сповіщень на електронну пошту користувача через протокол SMTP
HttpVideoServer	Клас, що реалізує локальний вебсервер. Забезпечує трансляцію відеопотоку з накладеними рамками детекції у форматі MJPEG для віддаленого моніторингу
BitmapSourceConverter	Допоміжний клас для швидкої конвертації кадрів між бібліотекою OpenCV (Mat) та графічною підсистемою WPF

Застосований модульний підхід дозволив уникнути створення надмірно зв'язаного коду (антипатерн God Object). Завдяки виділенню класів MqttService та EmailService, підсистема має високу готовність до змін: у разі необхідності міграції на інший протокол передачі даних достатньо буде модифікувати лише один ізольований сервісний клас без втручання в логіку графічного інтерфейсу чи обробки відео [21]-[23].

3.3 Опис основних функцій та структур даних підсистеми

Для практичної реалізації бізнес-логіки розробленої програмної підсистеми було створено комплекс спеціалізованих методів та структур даних.

Відповідно до парадигми об'єктно-орієнтованого програмування, внутрішній стан об'єктів (координати віртуальних зон, параметри нейромережі, тайм-аути та мережеві налаштування) інкапсульовано у відповідні змінні та властивості. Зі свого боку, обробка цих даних, захоплення відеопотоку, виконання інференсу YOLOv8 та мережева взаємодія реалізуються через набір ізольованих функцій. Такий підхід гарантує надійність роботи підсистеми в режимі реального часу та зручність подальшого масштабування. Зведений перелік та функціональне призначення ключових змінних і методів розроблених класів, сформованих з урахуванням офіційної документації платформи .NET, наведено в таблиці 3.2 [11]-[14].

Таблиця 3.2 – Основні змінні та методи програмних класів підсистеми [11]-[14]

Назва змінної / властивості	Тип даних	Клас приналежності	Опис призначення
1	2	3	4
yoloNet	Net	Recording ViewModel	Екземпляр нейромережі (модель YOLOv8) для класифікації об'єктів на кадрі
VirtualSensors	ObservableCollection<VirtualSensor>	Recording ViewModel	Динамічна колекція віртуальних датчиків (зон), що автоматично оновлює графічний інтерфейс
Area	OpenCvSharp.Rect	VirtualSensor	Геометричні координати та розміри полігону віртуальної зони інтересу (ROI) на відеопотоці
IsActive	bool	VirtualSensor	Поточний логічний статус присутності людини в заданій зоні

Кінець таблиці 3.2

1	2	3	4
LastPersonSeen	DateTime	VirtualSensor	Часова мітка останньої успішної детекції об'єкта, для роботи алгоритму фільтрації
DebounceTimeout	TimeSpan	Recording ViewModel	Часовий поріг (встановлено 5 секунд) для алгоритму усунення «мерехтіння» статусу
_lastMqttStates	Dictionary <string, bool>	MainWindow	Словник для відстеження попередніх станів зон з метою запобігання дублюванню мережових запитів
_selectionRectangle	System.Windows.Shapes.Rectangle	MainWindow	Графічний елемент (векторний контур) для візуалізації процесу малювання нової зони мишею
_videoWriter	VideoWriter	MainWindow	Об'єкт бібліотеки OpenCV для збереження обробленого відеопотоку на жорсткий диск (формати MP4, AVI, MKV)

Обробка даних, управління потоками та мережева комунікація реалізуються через спеціалізовані методи класів. Використання асинхронного програмування (async/await) гарантує, що операції розпізнавання та відправки мережових запитів не блокують головний потік графічного інтерфейсу (UI Thread) [14]-[21].

Основні методи та функції, що реалізують бізнес-логіку розробленого ПЗ, наведено в таблиці 3.3.

Таблиця 3.3 – Основні методи та функції класів підсистеми

Назва методу	Клас приналежності	Опис призначення
1	2	3
InitializeYolo()	RecordingViewModel	Завантаження ваг нейромережі у форматі ONNX та налаштування бекенду для виконання розрахунків на CPU
ConnectAsync()	MqttService	Встановлення асинхронного з'єднання з MQTT-брокером системи домашньої автоматизації за заданою IP-адресою та портом
StartFrameUpdate()	MainWindow	Головний асинхронний цикл отримання кадрів з камери, їх передачі до нейромережі та відмальовування результатів на екрані
DetectAndDraw(Mat)	RecordingViewModel	Виконання інференсу нейромережі (передача кадру у форматі Blob), відбір об'єктів класу «Людина», розрахунок геометричних центрів та перевірка їх входження у координати віртуальних зон
Canvas_MouseMove(), MouseLeftButtonDown(), MouseLeftButtonUp()	MainWindow	Комплекс методів обробки подій миші для реалізації графічного редактора зон інтересу (ROI Editor) поверх відеопотоку
HandleSecurityLogic()	MainWindow	Обробка логіки тривоги: фіксація часу виявлення, перевірка умов для старту відеозапису та ініціалізація відправки сповіщень на пошту

Кінець таблиці 3.3

1	2	3
PublishMessageAsync()	MqttService	Публікація логічного стану віртуального датчика (ON/OFF) у відповідний топик брокера з параметром RetainFlag = true для збереження стану
ProcessRequest()	HttpVideoServer	Метод генерації безперервного потоку байтів (HTTP-відповідь) у форматі multipart/x-mixed-replace для трансляції MJPEG-відео
ToBitmapSource(Mat)	BitmapSourceConverter	Статичний метод для перетворення матриці пікселів OpenCV у сумісний з WPF формат для відображення у графічному інтерфейсі

Отже, розроблена структура класів, змінних та методів повністю покриває вимоги технічного завдання, забезпечуючи швидку обробку кадрів, ізольовану роботу з мережевими протоколами та точну роботу логіки віртуальних датчиків.

3.4 Розробка графічного інтерфейсу користувача

Для забезпечення зручної та інтуїтивно зрозумілої взаємодії користувача з підсистемою було розроблено графічний інтерфейс користувача (Graphical User Interface, GUI). Інтерфейс реалізовано з використанням технології Windows Presentation Foundation (WPF) та мови розмітки XAML (eXtensible Application Markup Language) [17]. Вибір даної технології зумовлений можливістю гнучкого налаштування стилів елементів управління та ефективного рендерингу графіки за допомогою апаратного прискорення.

Головне вікно програми концептуально розділене на дві основні області: панель конфігурації (зліва) та область візуалізації відеопотоку (справа). Зовнішній вигляд головного вікна розробленого ПЗ наведено на рисунку 3.5.

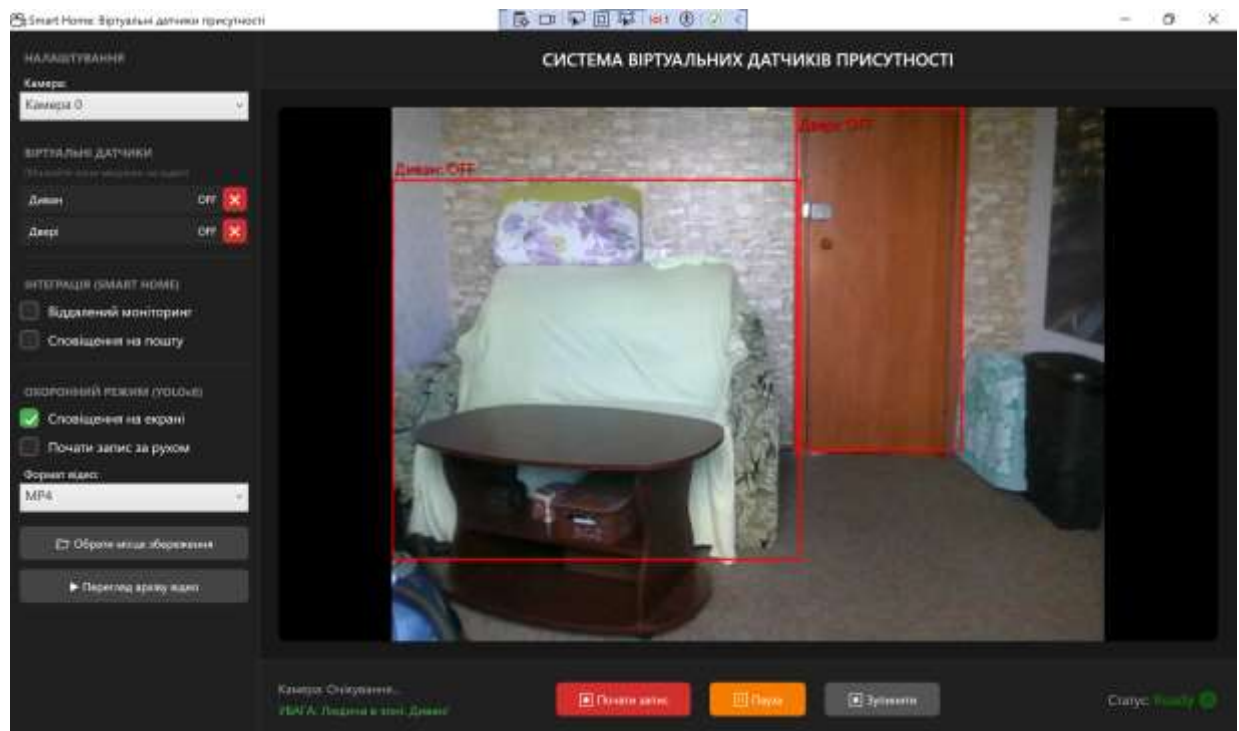


Рисунок 3.5 – Головне вікно розробленої підсистеми

Ліва панель містить основні елементи управління та налаштування, які згруповані за логічними категоріями:

- блок «Налаштування» – містить випадаючий список для вибору активної камери системи;
- блок «Віртуальні датчики» – містить динамічний список створених зон інтересу (Region of Interest, ROI). Кожен елемент списку відображає власну назву зони (наприклад, «Диван», «Двері»), поточний логічний статус (ON/OFF) та кнопки управління (видалення або редагування назви зони);

- блок «Інтеграція (Smart Home)» – дозволяє вмикати або вимикати додаткові модулі: HTTP-відеосервер для віддаленого моніторингу та систему надсилання тривожних сповіщень на електронну пошту;
- блок «Охоронний режим (YOLOv8)» – містить налаштування локального запису відеоданих та вибору формату збереження файлів (MP4, AVI, MKV).

Права частина вікна відведена під елемент Image, який відповідає за рендеринг оброблених кадрів відеопотоку. Поверх цього елемента розташовано прозорий шар Canvas, який забезпечує взаємодію користувача з відеоданими за допомогою комп'ютерної миші.

Ключовою особливістю графічного інтерфейсу є вбудований візуальний редактор зон інтересу. Замість ручного введення координат користувач має можливість виділяти необхідні області безпосередньо на відеозображенні. Процес формування нової віртуальної зони реалізовано через обробку трьох подій миші:

- `MouseDown` – фіксація початкової точки координат (X, Y) при натисканні лівої кнопки миші та створення тимчасового графічного об'єкта `Rectangle` (прямокутника) зеленого кольору із напівпрозорою заливкою;
- `MouseMove` – динамічна зміна ширини та висоти об'єкта `Rectangle` в реальному часі залежно від переміщення курсора зі збереженням обмежень (запобігання виходу за межі області відеопотоку);
- `MouseUp` – фіксація фінальних координат при відпусканні кнопки миші. Якщо площа виділеного прямокутника є достатньою (більше 5 пікселів, що запобігає випадковим клікам), система автоматично створює новий об'єкт віртуального датчика та додає його до колекції моніторингу.

Контури створених зон відмальовуються безпосередньо поверх відеопотоку. Для інтуїтивного сприйняття реалізовано колірну індикацію станів: за замовчуванням зона має червоний колір (статус OFF), проте при

успішній детекції людини в її межах рамка динамічно змінює колір на зелений.

Приклад візуалізації активного стану віртуального датчика наведено на рисунку 3.6.

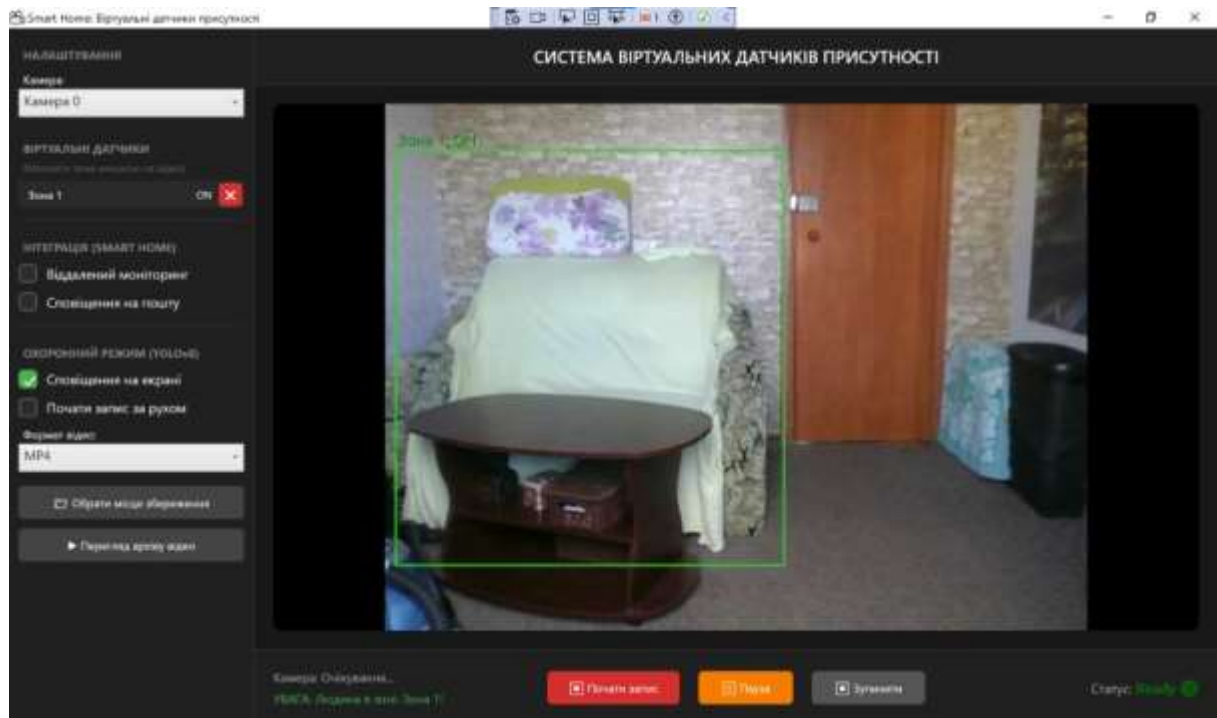


Рисунок 3.6 – Візуалізація активного стану віртуального датчика (статус ON)

Нижня частина головного вікна містить інформаційну панель (Status Bar), яка відображає поточний стан системи детекції, індикатори запису та статус з'єднання з брокером повідомлень (MQTT), забезпечуючи постійний зворотний зв'язок із користувачем [17]-[21].

3.5 Висновки до розділу 3

У даному розділі виконано конструювання ПЗ підсистеми віртуальних датчиків присутності. Розроблено та алгоритмізовано логіку обробки відеопотоку в реальному часі, що включає захоплення кадрів, інференс нейромережі та перевірку геометричного входження об'єктів у задані зони інтересу.

Застосування принципів об'єктно-орієнтованого програмування дозволило ізолювати модулі детекції (YOLOv8), мережевої взаємодії з платформою розумного будинку (MQTT) та електронних сповіщень (SMTP) у незалежні сервісні класи. Це забезпечило високу надійність підсистеми та готовність коду до подальшого масштабування без ризику виникнення надмірної зв'язності компонентів.

Програмно реалізовано алгоритм фільтрації хибних спрацювань (debouncing), що вирішує ключову проблему короткочасної втрати об'єкта нейромережею та запобігає некомфортному «мерехтінню» систем освітлення.

Окрім того, розроблено графічний інтерфейс користувача на базі технології WPF, який містить вбудований візуальний редактор для інтуїтивно зрозумілого налаштування віртуальних зон моніторингу безпосередньо поверх відеопотоку з динамічною колірною індикацією станів.

Отримані результати підтверджують повну готовність розробленої програмної підсистеми до етапу тестування та експериментального дослідження в умовах, наближених до реальної експлуатації.

4 ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПІДСИСТЕМИ

У даному розділі наведено покрокове керівництво користувача для налаштування розробленої програмної підсистеми віртуальних датчиків присутності. Проведено тестування функціональності графічного інтерфейсу, точності роботи неймережевих алгоритмів детекції та надійності мережевої інтеграції за протоколом MQTT [20]. Окрім того, виконано експериментальне порівняльне дослідження енергоефективності розробленої системи у порівнянні з класичними апаратними PIR-сенсорами [3]-[11].

4.1 Керівництво користувача та налаштування підсистеми

Розроблене ПЗ функціонує як десктопний додаток під управлінням операційної системи Windows. Для коректної роботи підсистеми комп'ютер має бути оснащений вебкамерою (вбудованою або підключеною через USB) та мати стабільне підключення до локальної мережі (для взаємодії з платформою розумного будинку).

Процес початкового налаштування підсистеми є інтуїтивно зрозумілим і складається з наступних кроків.

Крок 1. Вибір джерела відеосигналу. Після запуску програми користувач має звернути увагу на ліву панель управління. У блоці «Налаштування» з випадаючого списку необхідно обрати активну камеру (наприклад, «Камера 0»). Програма автоматично ініціалізує підключення, і в правій частині вікна з'явиться трансляція відеопотоку в реальному часі.

Крок 2. Конфігурація віртуальних датчиків (зон). Це ключовий етап налаштування. Користувачу необхідно візуально визначити на відеопотоці області, які потребують моніторингу (наприклад, диван, робоче місце, двері). Для створення зони потрібно:

- навести курсор миші на відповідну область відеозображення;

- затиснути ліву кнопку миші та протягнути курсор, формуючи прямокутну область (контур буде підсвічено зеленим кольором) [16];
- відпустити кнопку миші для фіксації зони.

Створена зона миттєво з'явиться у списку «Віртуальні датчики» на лівій панелі. За замовчуванням зоні присвоюється стандартна назва (наприклад, «Зона 1»). Користувач може змінити її на більш інформативну, натиснувши на текст назви, ввівши нове ім'я (наприклад, «Диван») та підтвердивши дію натисканням кнопки «✓». У разі помилкового створення зону можна видалити відповідною кнопкою «X» (рис. 4.1).

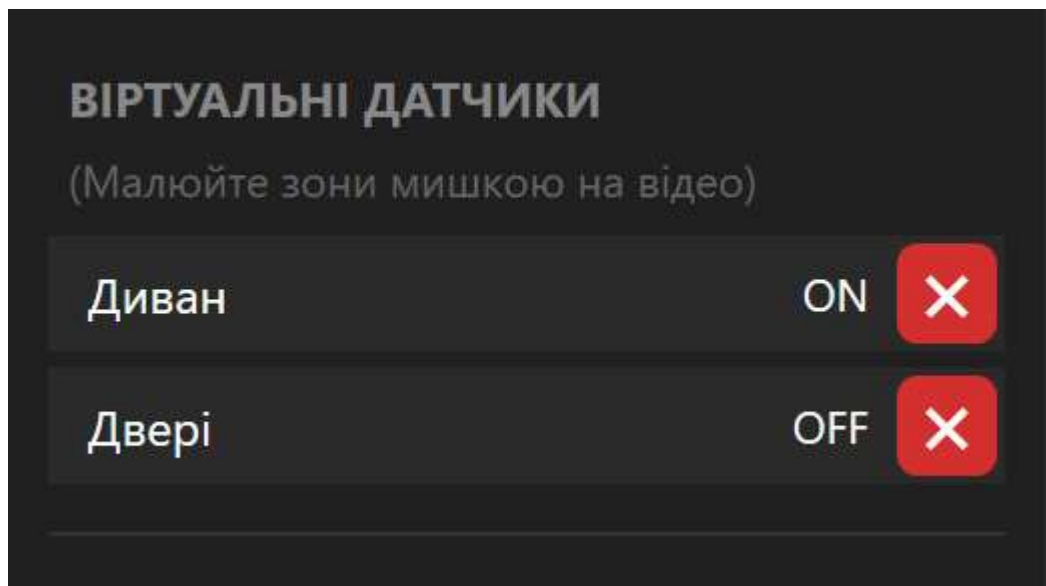


Рисунок 4.1 – Налаштування переліку віртуальних датчиків у графічному інтерфейсі

Крок 3. Налаштування інтеграції зі Smart Home. Для забезпечення зворотного зв'язку та керування автоматизацією користувач може активувати додаткові функції у блоці «Інтеграція»:

- увімкнення чекбокса «Віддалений моніторинг» запускає локальний HTTP-сервер, який дозволяє переглядати відеопотік з накладеними рамками детекції через веббраузер з будь-якого пристрою в локальній мережі (смартфона, планшета);

– увімкнення чекбокса «Сповіщення на пошту» викликає діалогове вікно, де користувач має ввести свою електронну адресу. Після підтвердження система автоматично надсилатиме тривожні листи у разі виявлення людини в охоронному режимі (рис. 4.2).

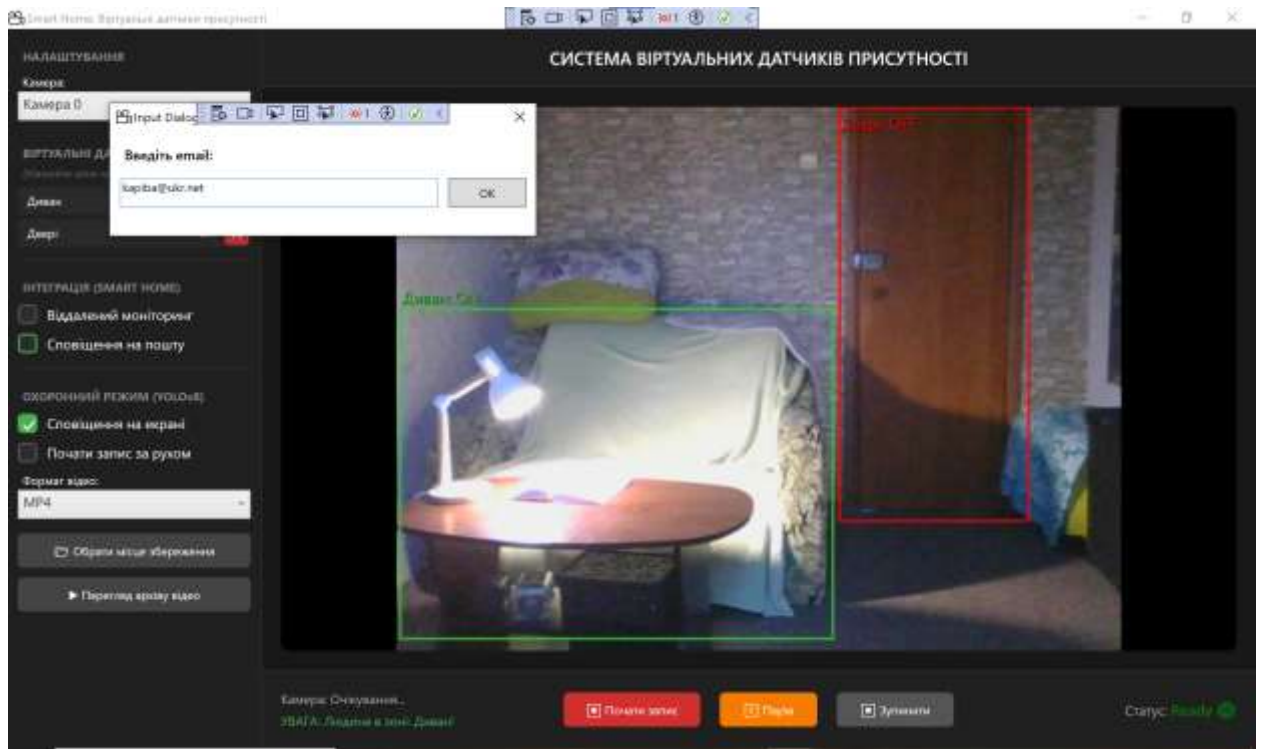


Рисунок 4.2 – Налаштування електронної пошти для отримання тривожних сповіщень

Крок 4. Управління записом. Програма підтримує функцію локального збереження відеоданих. Користувач може обрати формат запису (MP4, AVI, MKV) та директорію збереження. Керування записом здійснюється кнопками на нижній інформаційній панелі («Почати запис», «Пауза», «Зупинити») або автоматично при активації функції «Почати запис за рухом». Інформаційна панель також відображає поточний статус системи (Ready/Rec) та стан підключення до MQTT-брокера.

4.2 Тестування функціональності та мережевої інтеграції з платформою автоматизації

Для підтвердження безперебійної працездатності розробленого ПЗ в умовах, наближених до реальної експлуатації, було проведено комплексне тестування його основних функціональних модулів. Зокрема, перевірі підлягали підсистема комп'ютерного зору, відповідальна за просторову детекцію об'єктів, та підсистема мережевої комунікації, яка забезпечує надійну інтеграцію з платформою розумного будинку OpenHAB.

4.2.1 Тестування модуля комп'ютерного зору

Основою надійного функціонування віртуального датчика є здатність алгоритму безпомилково відрізняти цільовий клас «Людина» від інших динамічних об'єктів у кадрі, таких як домашні тварини, рухомі елементи інтер'єру або різка зміна освітленості (тіні). Під час етапу тестування на вхід підсистеми безперервно подавався відеопотік із локальної камери. Для обробки візуальних даних використовувалася згортова нейромережа архітектури YOLOv8 (модифікація Nano – YOLOv8n), експортована у відкритий формат представлення моделей машинного навчання (від англ. Open Neural Network Exchange, ONNX). Вона здійснювала покадровий інференс, спираючись виключно на обчислювальні ресурси центрального процесора (від англ. Central Processing Unit, CPU) [12]-[19].

Результати експериментального тестування продемонстрували високу точність детекції [8]-[10]. Алгоритм стабільно ідентифікує об'єкти класу «Людина» (внутрішній ідентифікатор класу 0) навіть у складних умовах: при частковому перекритті об'єкта меблями або під час тривалого перебування людини у статичному положенні. Оптимізація моделі дозволила досягти швидкості обробки відеопотоку на рівні понад 15 кадрів за секунду. Такий показник швидкодії є цілком достатнім для забезпечення миттєвої реакції

систем домашньої автоматизації без необхідності залучення спеціалізованих графічних прискорювачів. Додатково було підтверджено, що розроблений алгоритм розрахунку геометричного центру знайденої людини коректно співвідносить координати об'єкта з просторовими координатами заздалегідь намальованих віртуальних зон інтересу, миттєво змінюючи їхній логічний статус.

4.2.2 Тестування мережевої інтеграції

Для забезпечення надійної взаємодії розробленого додатка з платформою розумного будинку OpenHAB використовується легковаговий протокол передачі телеметрії (від англ. Message Queuing Telemetry Transport, MQTT) [20]. Під час тестування мережевого модуля програма успішно встановила стабільне з'єднання з локальним брокером повідомлень.

Логіка мережевої інтеграції була верифікована за таким сценарієм:

- при ініціалізації програмного комплексу відбувається асинхронне підключення до брокера за допомогою виклику методу `ConnectAsync()`;
- у момент, коли центр виявленого об'єкта класу «Людина» перетинає межі створеної користувачем віртуальної зони (наприклад, області з назвою «Зона 1»), статус цієї зони миттєво змінюється на активний;
- програма автоматично формує унікальний ідентифікатор топіка на основі назви зони (наприклад, `home/livingroom/virtual_sensor/зона_1`) та публікує в нього повідомлення з корисним навантаженням ON;
- при фізичному виході людини за межі зони алгоритм враховує встановлений 5-секундний таймаут фільтрації хибних спрацювань (`DebounceTimeout`), після чого відправляє у цей же топік команду OFF.

Окремим важливим аспектом інтеграції, що успішно пройшов перевірку, є використання прапорця збереження повідомлень (`RetainFlag = true`) [20]. Завдяки цій функції платформа OpenHAB здатна отримати

актуальний статус віртуального датчика одразу після свого перезавантаження або відновлення зв'язку з брокером.

Для раціонального використання пропускної здатності локальної мережі у програмі реалізовано механізм кешування станів (через структуру даних `_lastMqttStates`). Аналіз мережевого трафіку під час тестування показав, що повідомлення відправляються брокеру виключно в момент фактичної зміни статусу зони. Це повністю унеможливило інформаційний спам при тривалому знаходженні людини в одній локації [20].

Крім базового функціоналу, було успішно протестовано підсистему віддаленого моніторингу. Вбудований комунікаційний клас `HttpVideoServer` забезпечив стабільну генерацію відеопотоку у форматі Motion JPEG (MJPEG) на локальний порт 8080. Цей функціонал дозволяє виводити трансляцію з накладеними рамками детекції безпосередньо на панель управління (дашборд) системи OpenHAB або переглядати її через будь-який мобільний браузер. Додаткова функція надсилання тривожних сповіщень на електронну пошту через протокол простої передачі пошти (від англ. Simple Mail Transfer Protocol, SMTP), що містить перелік активних зон та точний час виявлення руху, також відпрацювала без збоїв і мережевих затримок.

4.3 Експериментальне дослідження розробленої підсистеми

Головним недоліком існуючих систем управління освітленням на базі інфрачервоних (PIR) датчиків руху є їхня нездатність фіксувати статичну присутність людини [4]-[5]. Це призводить до хибних вимкнень електроприладів (False Negative) або змушує користувачів встановлювати надмірно великі тайм-аути затримки вимкнення (наприклад, 15–30 хвилин), що нівелює концепцію енергозбереження [1]-[11].

Для доведення ефективності розробленої програмної підсистеми було проведено порівняльне експериментальне дослідження. Метою експерименту було порівняння часу нераціональної роботи системи освітлення при

використанні класичного PIR-датчика та розробленої відеоаналітичної підсистеми (YOLOv8 + алгоритм debouncing) в однакових побутових умовах.

Умови проведення експерименту:

- тестова зона: крісло або диван у зоні відпочинку (Zone 1: Sofa);
- електроприлад: настільна лампа для читання (інтегрована в OpenHAB);
- тип активності: читання книги (тривала статична поза з мінімальними рухами);
- тривалість сесії присутності: 20 хвилин (з 19:00 до 19:20);
- налаштування PIR-датчика: тайм-аут вимкнення після останнього зафіксованого руху – 10 хвилин;
- налаштування розробленої підсистеми: поріг впевненості (Confidence) – 0.5, тайм-аут втрати об'єкта (debouncing) – 5 секунд.

Хід експерименту – о 19:00 користувач увійшов у зону відпочинку та сів у крісло з книгою. Обидві системи (PIR та відеоаналітика) зафіксували присутність та згенерували сигнал "ON", що увімкнуло настільну лампу. Протягом наступних 20 хвилин користувач перебував у стані спокою. О 19:20 користувач покинув приміщення.

Результати хронометражу станів датчиків під час експерименту наведено в таблиці 4.1.

Таблиця 4.1 – Порівняльний хронометраж спрацювання датчиків присутності

Час (хв)	Дія користувача	Стан PIR-датчика (Тайм-аут 10 хв)	Стан розробленої підсистеми (YOLO)
1	2	3	4
19:00	Вхід у зону (початок читання)	ON	ON
19:05	Читання (статична поза)	ON	ON

Кінець таблиці 4.1

1	2	3	4
19:10	Статична поза (відсутність рухів > 10 хв)	OFF (Хибне вимкнення лампи)	ON
19:11	Користувач змушений зробити рух рукою для увімкнення лампи	ON	ON
19:20	Вихід із зони	ON	OFF (вимкнення через 5 с)
19:25	Кімната порожня	ON (Марна витрата енергії)	OFF
19:31	Кімната порожня	OFF	OFF

4.3.1 Аналіз отриманих результатів

Як видно з таблиці 4.1, класичний PIR-датчик продемонстрував нестабільну роботу під час статичної активності користувача. О 19:10 він хибно вимкнув настільну лампу через відсутність активних рухів, що змусило користувача перервати процес читання та зробити зайвий рух рукою. Більш того, після того, як користувач покинув крісло о 19:20, лампа продовжувала світити ще 10 хвилин (до 19:31) через встановлений апаратний тайм-аут. Отже, час нераціональної роботи системи освітлення склав 10 хвилин на одну сесію читання.

Отже, експеримент показав, що точність детекції розробленої підсистеми у статичному режимі (читання) становить 100%, тоді як апаратний PIR-сенсор демонструє нульову точність у цьому сценарії через помилкове вимкнення (False Negative). Це доводить виконання мети роботи щодо підвищення точності розпізнавання присутності

Теоретично, для економії енергії можна зменшити тайм-аут PIR-датчика до 1–2 хвилин. Проте на практиці це призведе до необхідності постійно виконувати надмірні рухи для підтримання освітлення увімкненим,

що категорично суперечить концепції комфорту розумного будинку. Додатковою проблемою класичних датчиків руху є відсутність семантичного розуміння сцени: вони генерують хибні спрацювання на рух домашніх тварин (котів, собак) або навіть на потоки теплого повітря.

Розроблена підсистема на базі комп'ютерного зору вирішує ці проблеми комплексно. Нейромережа YOLOv8 виконує класифікацію об'єктів і реагує виключно на клас «Людина» (Person), ігноруючи рух домашніх тварин [9]-[10]. Під час експерименту підсистема стабільно утримувала статус "ON" протягом усього часу перебування користувача в кріслі, ігноруючи статичність пози. Щойно користувач вийшов із зони (о 19:20), алгоритм debouncing вичекав лише 5 секунд (запобігаючи хибному спрацюванню у разі короткочасного перекриття камери), після чого миттєво згенерував сигнал "OFF" у MQTT-топік. Отже, час нераціональної роботи настільної лампи було скорочено до 5 секунд (рис. 4.3).

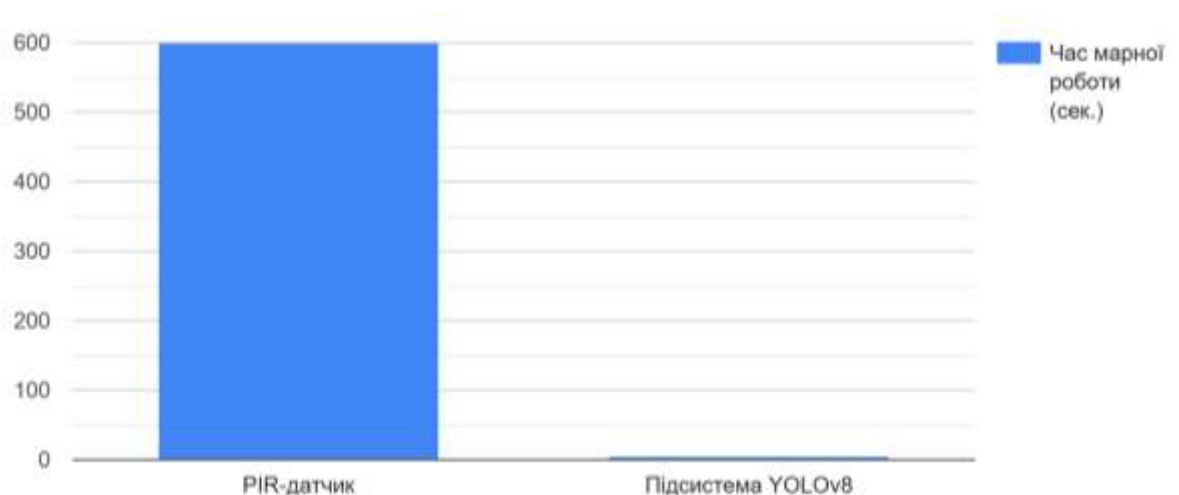


Рисунок 4.3 – Порівняння часу марної роботи системи освітлення (на 1 сесію)

Отже, експериментально доведено, що розроблена підсистема віртуальних датчиків присутності значно підвищує енергоефективність розумного будинку шляхом мінімізації часу нераціональної роботи освітлювальних приладів, усуває проблему хибних спрацювань на домашніх

тварин та забезпечує вищий рівень комфорту для користувача під час статичної активності.

4.4 Висновки до розділу 4

У даному розділі було успішно проведено комплексне тестування та експериментальне дослідження розробленої програмної підсистеми віртуальних датчиків присутності.

Складено покрокове керівництво користувача, яке підтверджує інтуїтивну зрозумілість процесу конфігурації зон моніторингу через графічний інтерфейс. Успішно протестовано всі ключові модулі: нейромережу YOLOv8 для точної детекції об'єктів класу «Людина», підсистему безперервної трансляції відеопотоку у форматі MJPEG через локальний HTTP-сервер, а також сервіс сповіщень на базі протоколу SMTP.

Особливу увагу приділено тестуванню мережевої інтеграції з системою домашньої автоматизації OpenHAB. Підтверджено стабільність обміну даними через протокол MQTT та коректність роботи оптимізованого алгоритму кешування станів, який запобігає надлишковому мережевому трафіку.

Виконано порівняльне експериментальне дослідження енергоефективності розробленої підсистеми у порівнянні з класичним апаратним PIR-датчиком руху. Доведено, що завдяки використанню семантичного аналізу відеозображення та алгоритму debouncing, розроблена підсистема здатна безпомилково фіксувати статичну присутність користувача. Це дозволило скоротити час нераціональної роботи системи освітлення з 10 хвилин (апаратний тайм-аут PIR-датчика) до 5 секунд на одну сесію, що підтверджує високу енергоефективність та практичну цінність розробленого рішення.

ВИСНОВКИ

У дипломній кваліфікаційній роботі вирішено актуальне науково-практичне завдання, мета якого полягала у підвищенні точності детекції присутності людини в приміщеннях для зменшення нераціональних витрат електроенергії на основі розроблення програмної підсистеми віртуальних датчиків присутності з використанням нейромережових алгоритмів [1]-[10].

У ході виконання роботи отримано такі результати:

- проаналізовано існуючі апаратні методи детекції присутності та виявлено їхній головний недолік – нездатність фіксувати статичну позу людини, що призводить до хибних вимкнень освітлення [3]-[4];
- сформульовано вимоги до ПЗ підсистеми та обґрунтовано вибір інструментарію розробки (мова C#, платформа WPF, бібліотека OpenCvSharp, нейромережа YOLOv8 Nano у форматі ONNX, протокол MQTT) [11]-[21];
- спроектовано та програмно реалізовано модульну архітектуру підсистеми з графічним інтерфейсом користувача для візуального налаштування зон інтересу безпосередньо поверх відеопотоку;
- розроблено алгоритм фільтрації хибних спрацювань на основі відкладеного вимкнення для уникнення «мерехтіння» освітлення при короткочасних перекриттях об'єкта, а також реалізовано незалежні мікросервіси для сповіщень на електронну пошту та віддаленого моніторингу [15]-[20];
- виконано комплексне тестування розробленого ПЗ та доведено, що підсистема на базі YOLOv8 стабільно фіксує статичну присутність та ігнорує рухи домашніх тварин [12]-[19].

Експериментальне дослідження підтвердило, що розроблена підсистема дозволяє скоротити час нераціональної роботи освітлювального приладу після виходу людини з приміщення з 10 хвилин (стандартний тайм-аут PIR-датчика) до 5 секунд на одну сесію. Отримані результати

підтверджують придатність програмної підсистеми віртуальних датчиків для успішної інтеграції з платформами домашньої автоматизації (зокрема, OpenHAB).

Можливими шляхами подальшого розвитку проєкту є розширення бази розпізнавання для ідентифікації конкретних користувачів (Face Recognition), додавання підтримки апаратного прискорення для обробки кількох відеопотоків одночасно та оптимізація підсистеми для роботи на одноплатних мікрокомп'ютерах (наприклад, Raspberry Pi).

Результати дослідження було представлено на Щорічній НПК викладачів, науковців, молодих учених, аспірантів та студентів НУ «Запорізька політехніка» «Тиждень науки-2026» (13-17 квітня 2026 року).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бондаренко О. В. Энергоефективність в системах «Розумний будинок» як фактор енергетичної безпеки. *Енергетика та автоматика*. 2023. № 2. С. 45–52.
2. Martínez R., Fernández A. Energy Management in Smart Homes: A Review of Strategies. *Energy Efficiency Journal*. 2023. Vol. 16. P. 230–245.
3. Johnson A., Williams B. Comparison of PIR and Video-Based Occupancy Detection. *IEEE Internet of Things Journal*. 2022. Vol. 9. No 14. P. 11200–11210.
4. Lee K., Smith J., Doe A. Why PIR Sensors Fail: Analysis of Stationary Occupancy Detection. *Sensors*. 2022. Vol. 22. No 5. P. 1890.
5. Martínez R. Review of Sensors for Smart Home Automation. *Journal of Sensor Systems*. 2021. Vol. 10. P. 115–128.
6. Bradski G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2000. Vol. 25. No 11. P. 120–123.
7. Aqara FP2 Presence Sensor User Manual and Technical Specifications. *Aqara Official*. URL: <https://www.aqara.com/en/product/presence-sensor-fp2> (дата звернення: 21.05.2026).
8. Zhao Y. Performance Analysis of mmWave Radars in Indoor Environments. *Technical Review*. 2023. № 4. С. 12–18.
9. Patel D. Comparative Analysis of Object Detection Algorithms: YOLO vs SSD. *Journal of Computer Vision*. 2024. Vol. 12. P. 88–95.
10. Jocher G., Chaurasia A. YOLOv8: A New State-of-the-Art Object Detection Model. 2023. URL: <https://github.com/ultralytics/ultralytics> (дата звернення: 11.05.2026).
11. Smith J. Energy Saving Strategies in Residential Buildings. *International Journal of Energy Research*. 2022. Vol. 46. P. 102–115.
12. Williams P. Smart Sensors and Sustainability. *Green Tech Journal*. 2023. № 1. С. 44–50.

13. Brown L. Analysis of PIR Sensor Limitations. *Electronic Systems*. 2021. Vol. 8. P. 22–29.
14. Miller S. Microwave Sensors in Domestic Environments. *Radar Science*. 2022. № 3. C. 77–83.
15. Davis M. Background Subtraction Methods in Video Surveillance. *Image Processing Quarterly*. 2020. Vol. 14. P. 150–162.
16. Wilson K. HOG and SVM for Human Detection. *Pattern Recognition Letters*. 2021. Vol. 32. P. 11–19.
17. Garcia R. Real-time Object Detection with CNN. *AI Review*. 2023. Vol. 5. P. 88–94.
18. YOLOv8 Modes: Export using ONNX. *Ultralytics*. URL: <https://docs.ultralytics.com/modes/export> (дата звернення: 15.05.2026).
19. Zhang M., Lin T., Zhao Y. Performance Analysis of YOLOv8 on CPU and GPU. *International Conference on Artificial Intelligence*. 2023. P. 112–118.
20. Hillar G. C. MQTT Essentials – A Lightweight IoT Protocol. Birmingham: Packt Publishing, 2017. 276 p.
21. Troelsen A., Japikse P. Pro C# 10 with .NET 6. Berkeley: Apress, 2022. 1650 p.
22. Sommerville I. Software Engineering. 10th ed. Pearson, 2015. 811 p.
23. IEEE Standard for Information Technology – Systems Design – Software Requirements. *IEEE Std 830-1998*. 1998. 40 p.
24. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Addison-Wesley, 2003. 175 p.
25. Larman C. Applying UML and Patterns. 3rd ed. Prentice Hall, 2004. 736 p.
26. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Addison-Wesley, 2012. 640 p.

ДОДАТОК А
Текст програми

A.1 Текст файлу RecordingViewModel.cs

```

using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Media;
using OpenCvSharp.Dnn;
using OpenCvSharp;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;

namespace CameraMonitoring
{
    // Клас, що інкапсулює бізнес-логіку підсистеми комп'ютерного
зору
    public class RecordingViewModel : INotifyPropertyChanged
    {
        private Net yoloNet;
        private const string ModelPath = "Models/yolov8n.onnx";

        public RecordingViewModel()
        {
            InitializeYolo();
        }

        // Ініціалізація нейромережі YOLOv8 для роботи на CPU
        private void InitializeYolo()
        {
            try

```

```

        {
            if (!System.IO.File.Exists(ModelPath)) return;
            yoloNet = CvDnn.ReadNetFromOnnx(ModelPath);
            yoloNet.SetPreferableBackend(Backend.OPENCV);
            yoloNet.SetPreferableTarget(Target.CPU);
        }
        catch (Exception ex)
        {
            System.Windows.MessageBox.Show($"Помилка завантаження
YOLO: {ex.Message}");
        }
    }

    private string _statusText = "Ready □";
    private Brush _statusColor = Brushes.Green;

    public string StatusText
    {
        get => _statusText;
        set { _statusText = value; OnPropertyChanged();
UpdateStatusColor(); }
    }

    public Brush StatusColor
    {
        get => _statusColor;
        private set { _statusColor = value; OnPropertyChanged(); }
    }

    private void UpdateStatusColor()

```

```

    {
        if (StatusText == "Ready □") StatusColor = Brushes.Green;
        else if (StatusText == "Rec ●") StatusColor = Brushes.Red;
        else if (StatusText == "Pause □") StatusColor =
Brushes.Goldenrod;
        else StatusColor = Brushes.Black;
    }

    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string
propertyName = null)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }

    public void StartRecording() => StatusText = "Rec ●";
    public void PauseRecording() => StatusText = "Pause □";
    public void StopRecording() => StatusText = "Ready □";

    // Клас-модель віртуального датчика з підтримкою оновлення UI
    public class VirtualSensor : INotifyPropertyChanged
    {
        private string _name;
        public string Name
        {
            get => _name;
            set { _name = value; OnPropertyChanged(); }
        }
    }

```

```

public OpenCvSharp.Rect Area { get; set; }
public DateTime LastPersonSeen { get; set; }
public Scalar Color { get; set; }

private bool _isActive;
public bool IsActive
{
    get => _isActive;
    set
    {
        if (_isActive != value)
        {
            _isActive = value;
            OnPropertyChanged();
            OnPropertyChanged(nameof(StatusDisplay));
        }
    }
}

public string StatusDisplay => IsActive ? "ON" : "OFF";

private bool _isEditing;
public bool IsEditing
{
    get => _isEditing;
    set { _isEditing = value; OnPropertyChanged(); }
}

private string _tempName;
public string TempName

```

```

    {
        get => _tempName;
        set { _tempName = value; OnPropertyChanged(); }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string
prop      =      null)      =>      PropertyChanged?.Invoke(this,      new
PropertyChangedEventArgs(prop));
    }

    public ObservableCollection<VirtualSensor> VirtualSensors { get;
set; } = new ObservableCollection<VirtualSensor>();

    // Налаштування таймауту для фільтрації хибних спрацювань
(debouncing)
    private readonly TimeSpan DebounceTimeout =
TimeSpan.FromSeconds(5);
    private int _frameCounter = 0;
    private List<OpenCvSharp.Rect> _foundPersons = new
List<OpenCvSharp.Rect>();

    // Головний метод детекції об'єктів та обробки логіки віртуальних
ЗОН
    public void DetectAndDraw(Mat frame)
    {
        if (yoloNet == null || frame.Empty()) return;
        _frameCounter++;
    }

```

```

// Оптимізація продуктивності: пошук виконується раз на 5
кадрів

if (_frameCounter % 5 == 0)
{
    var blob = CvDnn.BlobFromImage(frame, 1 / 255.0, new
OpenCvSharp.Size(640, 640), new Scalar(0, 0, 0), true, false);
    yoloNet.SetInput(blob);

    Mat output = yoloNet.Forward();
    Mat predictions = new Mat();
    Cv2.Transpose(output.Reshape(1, 84), predictions);

    _foundPersons.Clear();
    var boxes = new List<OpenCvSharp.Rect>();
    var confidences = new List<float>();

    for (int i = 0; i < predictions.Rows; i++)
    {
        float maxScore = 0;
        int bestClassId = -1;
        for (int c = 4; c < 84; c++)
        {
            float score = predictions.At<float>(i, c);
            if (score > maxScore) { maxScore = score; bestClassId = c -
4; }
        }

        // Відбір об'єктів виключно класу 0 («Людина») з високою
впевненістю

        if (maxScore > 0.5 && bestClassId == 0)

```

```

        {
            float cx = predictions.At<float>(i, 0), cy =
predictions.At<float>(i, 1);
            float w = predictions.At<float>(i, 2), h =
predictions.At<float>(i, 3);
            if (w < 1.0f && h < 1.0f) { cx *= 640; cy *= 640; w *= 640;
h *= 640; }

            int x = (int)((cx - w / 2) * ((float)frame.Width / 640));
            int y = (int)((cy - h / 2) * ((float)frame.Height / 640));
            int width = (int)(w * ((float)frame.Width / 640));
            int height = (int)(h * ((float)frame.Height / 640));

            boxes.Add(new OpenCvSharp.Rect(x, y, width, height));
            confidences.Add(maxScore);
        }
    }

    output.Dispose();
    predictions.Dispose();
    blob.Dispose();

    if (boxes.Count > 0)
    {
        CvDnn.NMSBoxes(boxes, confidences, 0.5f, 0.45f, out int[]
indices);

        foreach (var i in indices) _foundPersons.Add(boxes[i]);
    }
}

```

```

// Малювання кириличних назв зон та зміна кольору контуру
залежно від статусу
        using (var bitmap =
OpenCvSharp.Extensions.BitmapConverter.ToBitmap(frame))
        {
            using (var graphics =
System.Drawing.Graphics.FromImage(bitmap))
            {
                graphics.TextRenderingHint =
System.Drawing.Text.TextRenderingHint.AntiAliasGridFit;
                graphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

                System.Drawing.Font font = new
System.Drawing.Font("Segoe UI", 14, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Pixel);

                using (var shadowBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.FromArgb(180, 0, 0, 0)))
                {
                    foreach (var sensor in VirtualSensors)
                    {
                        var color = sensor.IsActive ?
System.Drawing.Color.LimeGreen : System.Drawing.Color.Red;

                        using (var pen = new System.Drawing.Pen(color, 2))
                        using (var brush = new
System.Drawing.SolidBrush(color))
                        {

```

```

        graphics.DrawRectangle(pen, sensor.Area.X,
sensor.Area.Y, sensor.Area.Width, sensor.Area.Height);
        string text = $"{sensor.Name}: {(sensor.IsActive ?
"ON" : "OFF")}";
        float textY = sensor.Area.Y > 20 ? sensor.Area.Y - 20 :
sensor.Area.Y + 5;
        graphics.DrawString(text, font, shadowBrush, new
System.Drawing.PointF(sensor.Area.X + 1, textY + 1));
        graphics.DrawString(text, font, brush, new
System.Drawing.PointF(sensor.Area.X, textY));
    }
}
}
}

var matWithText =
OpenCvSharp.Extensions.BitmapConverter.ToMat(bitmap);
matWithText.CopyTo(frame);
matWithText.Dispose();
}

// Розрахунок геометричного центру людини та перевірка її
входження до зони
foreach (var rect in _foundPersons)
{
    Cv2.Rectangle(frame, rect, Scalar.LightGreen, 1);
    int centerX = rect.X + (rect.Width / 2), centerY = rect.Y +
(rect.Height / 2);
    Cv2.Circle(frame, new OpenCvSharp.Point(centerX, centerY), 5,
Scalar.Red, -1);

```

```

        foreach (var sensor in VirtualSensors)
        {
            if (sensor.Area.Contains(new OpenCvSharp.Point(centerX,
centerY)))
            {
                sensor.LastPersonSeen = DateTime.Now;
            }
        }
    }

    // Алгоритм усунення мерехтіння статусу (debouncing)
    foreach (var sensor in VirtualSensors)
    {
        sensor.IsActive = (DateTime.Now - sensor.LastPersonSeen) <=
DebounceTimeout;
    }
}

public bool IsAnyZoneActive()
{
    return VirtualSensors.Any(s => s.IsActive);
}
}

```

A.2 Текст файлу MqttService.cs

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MQTTnet;
using MQTTnet.Client;
namespace CameraMonitoring.Services
{
    internal class MqttService
    {
        private IMqttClient _mqttClient;

        // Повертає true, якщо підключення успішне
        public async Task<bool> ConnectAsync(string serverIp = "localhost",
int port = 1883)
        {
            var factory = new MqttFactory();
            _mqttClient = factory.CreateMqttClient();

            var options = new MqttClientOptionsBuilder()
                .WithTcpServer(serverIp, port)
                .Build();

            try
            {
                await _mqttClient.ConnectAsync(options,
System.Threading.CancellationToken.None);
                return true;
            }
            catch
            {

```

```

        return false;
    }
}

public async Task PublishMessageAsync(string topic, string payload)
{
    if (_mqttClient != null && _mqttClient.IsConnected)
    {
        var message = new MqttApplicationMessageBuilder()
            .WithTopic(topic)
            .WithPayload(payload)

.WithQualityOfServiceLevel(MQTTnet.Protocol.MqttQualityOfServiceLevel.AtLeastOnce)

            .WithRetainFlag(true)
            .Build();

        await _mqttClient.PublishAsync(message,
System.Threading.CancellationToken.None);
    }
}

public void Disconnect()
{
    if (_mqttClient != null && _mqttClient.IsConnected)
    {
        _mqttClient.DisconnectAsync().Wait();
        _mqttClient.Dispose();
    }
}

```

```

    }
}

```

A.3 Текст файлу MainWindow.xaml.cs (фрагмент обробки зон та логіки безпеки)

```

private void
HandleSecurityLogic(System.Collections.Generic.List<string> activeZones)
{
    Dispatcher.Invoke(() =>
    {
        if (activeZones.Count > 0)
        {
            _lastPresenceTime = DateTime.Now;
            string zonesText = string.Join(", ", activeZones);
            if (CbMessages.IsChecked == true)
                UpdateMotionDetectionStatus($"УВАГА: Людина в зоні: {zonesText}!");

            if (_notificationsEnabled && (DateTime.Now -
                _lastEmailTime).TotalMinutes >= 1)
            {
                _lastEmailTime = DateTime.Now;

                // Формуємо красивий текст листа
                string emailBody = $"Система Розумного Дому зафіксувала
                присутність!\n\n" +
                    $"🔴 Активні віртуальні датчики: {zonesText}\n"
                +

```



```

    }
    });
}

```

```

private void DeleteZone_Click(object sender, RoutedEventArgs e)
{
    if (sender is Button btn && btn.Tag is
RecordingViewModel.VirtualSensor sensor)
        _viewModel.VirtualSensors.Remove(sensor);
}

```

```

private void Canvas_MouseLeftButtonDown(object sender,
MouseButtonEventArgs e)
{
    _startPoint = e.GetPosition(Canvas);
    _selectionRectangle = new Rectangle
    {
        Stroke = Brushes.LimeGreen,
        StrokeThickness = 2,
        Fill = new SolidColorBrush(Color.FromArgb(50, 0, 255, 0)),
        StrokeDashArray = new DoubleCollection() { 2, 2 }
    };
    Canvas.Children.Add(_selectionRectangle);
    Canvas.SetLeft(_selectionRectangle, _startPoint.X);
    Canvas.SetTop(_selectionRectangle, _startPoint.Y);

    // Захоплюємо мишу Canvas стежить за нею навіть за межами
Вікна

    Canvas.CaptureMouse();
}

```

```

private void Canvas_MouseMove(object sender, MouseEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed && _selectionRectangle
!= null)
    {
        var pt = e.GetPosition(Canvas);

        // ОБМЕЖЕННЯ: не даємо координатам вийти за межі екрану
відео (0..Width, 0..Height)
        double x = Math.Max(0, Math.Min(pt.X, Canvas.ActualWidth));
        double y = Math.Max(0, Math.Min(pt.Y, Canvas.ActualHeight));

        double startX = Math.Max(0, Math.Min(_startPoint.X,
Canvas.ActualWidth));
        double startY = Math.Max(0, Math.Min(_startPoint.Y,
Canvas.ActualHeight));

        // Вираховуємо ширину та висоту з урахуванням обмежень
_selectionRectangle.Width = Math.Abs(x - startX);
_selectionRectangle.Height = Math.Abs(y - startY);

        // Завжди малюємо від меншої координати (щоб можна було
малювати знизу-вгору)
Canvas.SetLeft(_selectionRectangle, Math.Min(x, startX));
Canvas.SetTop(_selectionRectangle, Math.Min(y, startY));
    }
}

```

```

private void Canvas_MouseLeftButtonUp(object sender,
MouseButtonEventArgs e)
{
    if (_selectionRectangle != null)
    {
        // Відпускаємо мишу
        Canvas.ReleaseMouseCapture();

        var newArea = new CvRect(
            (int)Canvas.GetLeft(_selectionRectangle),
            (int)Canvas.GetTop(_selectionRectangle),
            (int)_selectionRectangle.Width,
            (int)_selectionRectangle.Height);

        // Створюємо зону тільки якщо вона не є випадковим мікро-
        кліком (більше 5 пікселів)
        if (newArea.Width > 5 && newArea.Height > 5)
        {
            int zoneCount = _viewModel.VirtualSensors.Count + 1;
            _viewModel.VirtualSensors.Add(new
RecordingViewModel.VirtualSensor { Name = $"Зона {zoneCount}", Area =
newArea, IsActive = false, LastPersonSeen = DateTime.MinValue });
        }

        Canvas.Children.Remove(_selectionRectangle);
        _selectionRectangle = null;
    }
}

```

A.4 Текст файлу HttpVideoServer.cs

```

using OpenCvSharp;
using System;
using System.Net;
using System.Threading.Tasks;

namespace CameraMonitoring
{
    // Клас для віддаленого перегляду відеопотоку через локальну
мережу
    public class HttpVideoServer
    {
        private HttpListener _httpListener;
        private VideoCapture _videoCapture;
        private bool _isRunning = false;

        public HttpVideoServer(VideoCapture videoCapture)
        {
            _videoCapture = videoCapture;
        }

        public void StartServer(string address = "http://localhost:8080/")
        {
            if (_isRunning) return;

            try
            {
                _httpListener = new HttpListener();
                _httpListener.Prefixes.Add(address);
            }
        }
    }
}

```

```

_httpListener.Start();
_isRunning = true;

Task.Run(() =>
{
    while (_isRunning && _httpListener != null &&
_httpListener.IsListening)
    {
        try
        {
            var context = _httpListener.GetContext();
            Task.Run(() => ProcessRequest(context));
        }
        catch (HttpListenerException) { }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }
});
}
catch (Exception ex)
{
    Console.WriteLine($"Помилка запуску сервера:
{ex.Message}");
    _isRunning = false;
}
}

```

// Асинхронне формування та відправка кадрів у форматі MJPEG

```

private async Task ProcessRequest(HttpListenerContext context)
{
    try
    {
        context.Response.ContentType = "multipart/x-mixed-replace;
boundary=--frame";
        context.Response.StatusCode = 200;

        var responseStream = context.Response.OutputStream;
        while (_isRunning && _httpListener != null &&
_httpListener.IsListening)
        {
            using (var frame = new Mat())
            {
                if (_videoCapture != null && _videoCapture.IsOpened())
                {
                    _videoCapture.Read(frame);
                    if (!frame.Empty())
                    {
                        var imageData = frame.ToBytes(".jpg");
                        var header = $"--frame\r\nContent-Type:
image/jpeg\r\nContent-Length: {imageData.Length}\r\n\r\n";
                        var headerBytes =
System.Text.Encoding.UTF8.GetBytes(header);
                        await responseStream.WriteAsync(headerBytes, 0,
headerBytes.Length);
                        await responseStream.WriteAsync(imageData, 0,
imageData.Length);
                        await responseStream.WriteAsync(new byte[] { 13, 10
}, 0, 2);

```

```

        }
    }
}
    await Task.Delay(33); // Підтримка стабільних 30 FPS
}
}
catch { }
finally
{
    try { context.Response.Close(); } catch { }
}
}
public void StopServer()
{
    if (!_isRunning) return;
    _isRunning = false;
    if (_httpListener != null)
    {
        try
        {
            _httpListener.Stop();
            _httpListener.Close();
        }
        catch { }
        _httpListener = null;
    }
}
}
}

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Факультет комп'ютерних наук і технологій
Кафедра програмних засобів

1

ДИПЛОМНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПІДСИСТЕМИ
ВІРТУАЛЬНИХ ДАТЧИКІВ ПРИСУТНОСТІ ДЛЯ РОЗУМНОГО БУДИНКУ

Виконала
ст. гр. КНТ-132
Керівник
к.т.н., доц.

Анжеліка СИМАК

Анжеліка ПАРХОМЕНКО

Рисунок Б.1 – Слайд 1

АКТУАЛЬНІСТЬ РОБОТИ

2

Критерій порівняння	Aqara FP2 (Комерційна)	Frigate NVR (Open Source)	Розроблювана підсистема
Технологія детекції	Радіохвильовий радар (mmWave)	Комп'ютерний зір (камери)	Комп'ютерний зір (камери)
Визначення статичної пози	Так	Так	Так
Можливість зонування	Так (через закритий додаток)	Так (через YAML файли)	Так (через власний GUI редактор)
Інтеграція з OpenHAB	Обмежена (потребує хмари)	Складна	Нативна та проста (через MQTT)
Головний недолік	Висока ціна, хибні спрацювання	Високі вимоги до "заліза"	Залежність від освітлення

Рисунок Б.2 – Слайд 2

3

МЕТА ТА ЗАВДАННЯ РОБОТИ

Мета роботи — підвищення точності детекції присутності людини в приміщеннях для зменшення нерациональних витрат електроенергії на основі розроблення програмної підсистеми віртуальних датчиків присутності з використанням нейромережевих алгоритмів.

Завдання роботи:

- проаналізувати існуючі методи детекції присутності та їхні недоліки в контексті раціонального використання енергоресурсів;
- сформулювати вимоги до програмного забезпечення підсистеми та обґрунтувати вибір інструментарію розроблення;
- розробити алгоритмічне забезпечення та виконати програмну реалізацію підсистеми віртуальних датчиків присутності;
- провести тестування та експериментальне дослідження точності й енергоефективності розробленого рішення.

Рисунок Б.3 – Слайд 3

4

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ ДЕТЕКЦІЇ ПРИСУТНОСТІ

Критерій порівняння	PIR-датчики	Ультразвукові датчики	Відеоаналітика (Computer Vision)
Детекція руху	Відмінно	Відмінно	Відмінно
Детекція статички	Неможливо	Погано (потрібен мікрорух)	Відмінно (за візуальними ознаками)
Класифікація об'єкта	Ні (реагує на тварин)	Ні	Так (Людина / Тварина)
Зонування	Потрібен окремий датчик на кожну зону	Складно	Програмні зони (Region of Interest, ROI) на одному кадрі
Вартість обладнання	Низька	Середня	Середня (наявність камери)

Рисунок Б.4 – Слайд 4

ОБҐРУНТУВАННЯ ВИБОРУ НЕЙРОМЕРЕЖІ

5

Критерій порівняння	HOG + SVM	MobileNet SSD	YOLOv8 Nano
Швидкодія на CPU (Frames Per Second, FPS)	Низька (1-5 FPS)	Середня (10-15 FPS)	Висока (15-30 FPS)
Точність детекції (mean Average Precision, mAP)	Низька	Середня	Висока
Стійкість до перекриття	Дуже низька	Середня	Висока
Ресурсомісткість	Високі вимоги	Низькі	Низькі (оптимізовано)
Складність у .NET	Низька (OpenCV)	Середня	Середня (через ONNX)

Рисунок Б.5 – Слайд 5

ОБҐРУНТУВАННЯ ВИБОРУ ПРОТОКОЛУ ОБМІНУ ДАНИМИ

6

Критерій	MQTT	HTTP (REST)	CoAP
Архітектура	Pub/Sub (Подієва)	Request/Response	Request/Response
Накладні витрати	Мінімальні (2 байти)	Високі (Заголовки)	Низькі
Гарантія доставки	Так (QoS 0, 1, 2)	Ні	Так (Confirmable)
Придатність для realtime	Висока	Низька (Polling)	Середня

Рисунок Б.6 – Слайд 6

ДІАГРАМА ПРЕЦЕДЕНТІВ

7

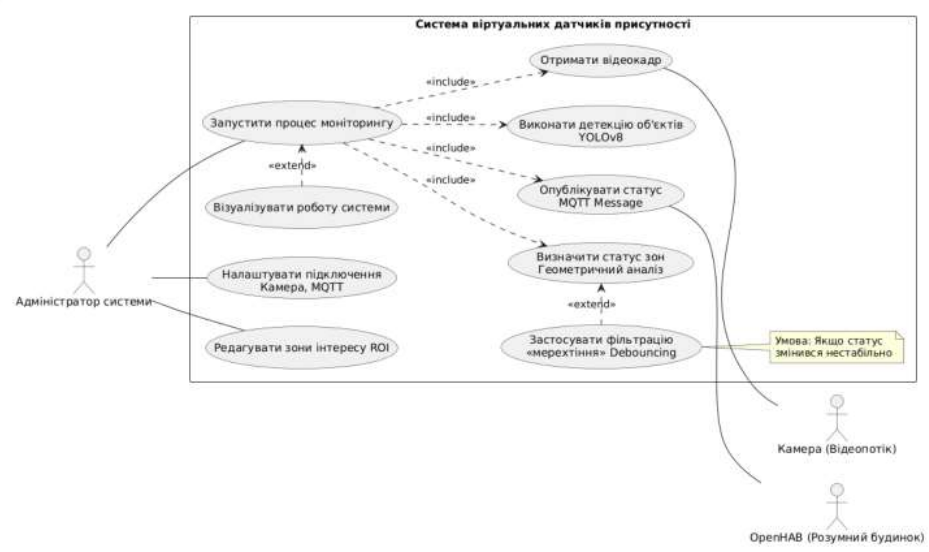


Рисунок Б.7 – Слайд 7

АРХІТЕКТУРА ПРОГРАМНОЇ ПІДСИСТЕМИ

8

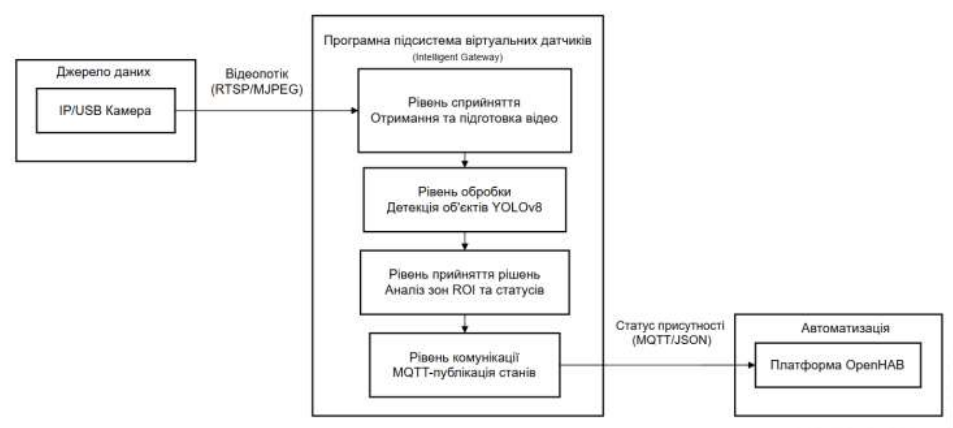


Рисунок Б.8 – Слайд 8

АЛГОРИТМ РОБОТИ ВІРТУАЛЬНОГО ДАТЧИКА

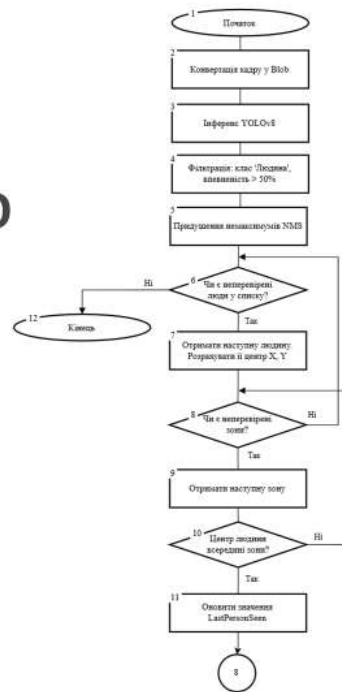


Рисунок Б.9 – Слайд 9

АЛГОРИТМ УСУНЕННЯ «МЕРЕХТІННЯ» (DEBOUNCING)

Математична логіка алгоритму фільтрації хибних спрацювань

- Проблема: Короткочасна втрата об'єкта нейромережею (перекриття, зміна пози).
- Наслідок: Дискомфортне «мерехтіння» освітлення (False Negative).
- Рішення: Застосування логіки відкладеного вимкнення (Debouncing).

Формула:

$$\text{IsActive} = (\text{DateTime.Now} - \text{LastPersonSeen}) \leq \text{DebounceTimeout}$$

Де: DebounceTimeout = 5 секунд.

Рисунок Б.10 – Слайд 10

ДІАГРАМА КЛАСІВ ТА МОДУЛІВ

11

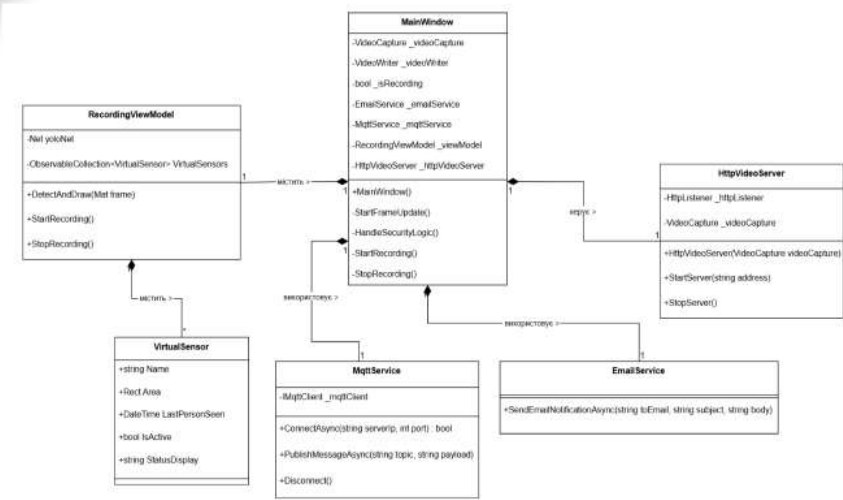


Рисунок Б.11 – Слайд 11

ОПИС ТА РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТУ

12

Час (хв)	Дія користувача	Стан PIR-датчика (Тайм-аут 10 хв)	Стан розробленої підсистеми (YOLOv8)
19:00	Вхід у зону (початок читання)	ON	ON
19:05	Читання (статична поза)	ON	ON
19:10	Статична поза (відсутність рухів > 10 хв)	OFF (Хибне вимкнення лампи)	ON
19:11	Користувач змушений зробити рух рукою для увімкнення лампи	ON	ON
19:20	Вихід із зони	ON	OFF (вимкнення через 5 с)
19:25	Кімната порожня	ON (Марна витрата енергії)	OFF
19:31	Кімната порожня	OFF	OFF

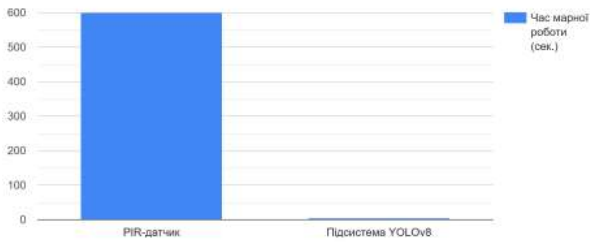


Рисунок Б.12 – Слайд 12

ДЕМОНСТРАЦІЯ РОБОТИ ПІДСИСТЕМИ 13

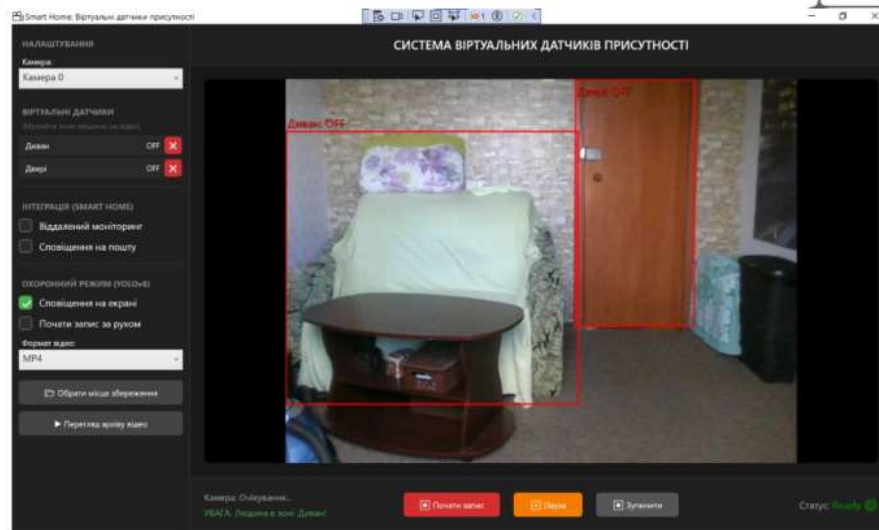


Рисунок Б.13 – Слайд 13

ВИСНОВКИ

14

1. Проаналізовано існуючі апаратні методи детекції присутності та виявлено їх недолік — нездатність фіксувати статичну позу людини, що призводить до хибних вимкнень освітлення.
2. Сформульовано вимоги до ПЗ та обґрунтовано вибір інструментарію розробки (C#, WPF, OpenCvSharp, YOLOv8 Nano у форматі ONNX, протокол MQTT).
3. Спроектовано та програмно реалізовано модульну архітектуру підсистеми з графічним інтерфейсом користувача для візуального налаштування зон інтересу.
4. Розроблено алгоритм фільтрації хибних спрацювань на основі відкладеного вимкнення для уникнення «мерехтіння» статусу детекції при короткочасних перекриттях об'єкта.
5. Експериментально підтверджено, що розроблена підсистема дозволяє скоротити час нераціональної роботи освітлення з 10 хвилин до 5 секунд на одну сесію.
6. Результати дослідження представлено на Щорічній науково-практичній конференції «Тиждень науки-2026» в НУ «Запорізька політехніка».

Рисунок Б.14 – Слайд 14