ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний застосунок для розпізнавання зображень на основі методів комп'ютерного зору»

зі спеціальності *122 Комп'ютерні науки*
(код, найменування спеціальності)

освітньо-професійної програми *штучний інтелект*
(назва програми)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр КЕРЕБ
(підпис)

Виконав: здобувач вищої освіти групи ШІД-42
Олександр КЕРЕБ
(прізвище, ім'я)

Керівник *к.т.н., доцент* Максим ФЕСЕНКО
(науковий ступінь, вчене звання, прізвище, ім'я)

Рецензент _____
(науковий ступінь, вчене звання, прізвище, ім'я)

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Штучного інтелекту
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп’ютерні науки
Освітньо-професійна програма штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедри ШІ
_____ Ольга ЗІНЧЕНКО
“___” _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

КЕРЕБ Олександр Олександрович
(прізвище, ім’я)

1. Тема кваліфікаційної роботи: «Програмний застосунок для розпізнавання зображень на основі методів комп'ютерного зору»

керівник кваліфікаційної роботи ФЕСЕНКО Максим, к.т.н., доцент
(прізвище, ім’я, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 року № 56.

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 23.05.2025 р.

3. Вихідні дані до кваліфікаційної роботи
приклади реалізації систем комп’ютерного зору для розпізнавання геометричних фігур;
наукові праці та публікації з тематики обробки зображень, комп’ютерного зору, машинного навчання;
дослідження ефективності алгоритмів детекції контурів та класифікації фігур.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз сучасних методів комп’ютерного зору та їх застосування для розпізнавання зображень.
2. Огляд та обґрунтування вибору алгоритмів виявлення контурів і класифікації геометричних фігур.

3. Тестування застосунку на синтетичних та реальних зображеннях, аналіз точності розпізнавання.

4. Перелік графічного матеріалу: презентація PowerPoint.

6. Дата видачі завдання 25.02.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз предметної області та напрямів застосування комп'ютерного зору	15.03.2025 р.	виконано
2.	Огляд літератури з методів розпізнавання зображень і технологій OpenCV, PyQt	31.03.2025 р.	виконано
3.	Вибір алгоритмів та засобів реалізації детекції контурів і класифікації фігур	10.04.2025 р.	виконано
4.	Розробка структури застосунку та генерація тестових зображень	25.04.2025 р.	виконано
5.	Реалізація основного функціоналу з обробки зображень у реальному часі	10.05.2025 р.	виконано
6.	Створення графічного інтерфейсу користувача з можливістю інтерактивного керування	13.05.2025 р.	виконано
7.	Проведення тестування, оцінка точності роботи застосунку в різних умовах	15.05.2025 р.	виконано
8.	Підготовка пояснювальної записки та доповіді до захисту	18.05.2025 р.	виконано

Здобувач вищої освіти _____ Олександр КЕРЕБ
(підпис) *(ім'я, прізвище)*

Керівник кваліфікаційної роботи _____ Максим ФЕСЕНКО
(підпис) *(ім'я, прізвище)*

РЕФЕРАТ

Текстова частина бакалаврської роботи: 60 сторінок, 26 рисунків, 45 джерел.

Об'єкт дослідження – процеси автоматизованого аналізу зображень із використанням алгоритмів комп'ютерного зору.

Предмет дослідження – методи та засоби розпізнавання геометричних фігур на зображеннях у режимі реального часу з використанням бібліотек OpenCV та графічного інтерфейсу на базі PyQt.

Мета роботи – розробити програмний застосунок для розпізнавання зображень у реальному часі з веб-камери та згенерованих сцен, що дозволяє виявляти, класифікувати та виводити інформацію про геометричні об'єкти.

Методи дослідження:

1. Аналіз існуючих підходів до реалізації комп'ютерного зору з використанням Python та OpenCV.
2. Реалізація алгоритму виявлення контурів і визначення фігур за кількістю вершин та округлістю.
3. Побудова клієнтської частини з графічним інтерфейсом для інтерактивної генерації сцен і керування процесом розпізнавання.
4. Тестування системи на реальних відеопотоках та синтетичних зображеннях, аналіз точності розпізнавання.

Галузь використання – автоматизоване розпізнавання об'єктів у навчальних, промислових і дослідницьких цілях, системи контролю, інтерфейси комп'ютерного зору.

Ключові слова: КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ФІГУР, OPENCV, PYTHON, ВЕБ-КАМЕРА, PYQT, АВТОМАТИЗОВАНЕ ВИЗНАЧЕННЯ КОНТУРІВ, ВІЗУАЛІЗАЦІЯ ДАНИХ, ГРАФІЧНИЙ ІНТЕРФЕЙС.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах стрімкого розвитку інформаційних технологій системи комп'ютерного зору стають важливим інструментом для автоматизації процесів аналізу візуальної інформації. Розпізнавання об'єктів на зображеннях знаходить широке застосування в різних галузях: від промисловості, медицини та транспорту до освіти, безпеки й розумного дому. Одним із найважливіших напрямів у цій сфері є розпізнавання геометричних фігур як базових об'єктів для подальшого аналізу сцен, просторового орієнтування та взаємодії з навколишнім середовищем.

Незважаючи на наявність великої кількості досліджень у галузі комп'ютерного зору, питання інтеграції алгоритмів розпізнавання геометричних об'єктів із графічним інтерфейсом користувача для навчальних та прикладних задач залишається актуальним. Це зумовлює необхідність створення програмного застосунку, що поєднує алгоритмічну точність, швидкодію та зручність взаємодії.

Дослідження у сфері розпізнавання зображень із використанням методів комп'ютерного зору має важливе теоретичне та практичне значення й відповідає сучасним потребам у цифровій автоматизації візуального аналізу.

Мета роботи – розробити програмний застосунок для розпізнавання зображень у реальному часі з веб-камери та згенерованих сцен, що дозволяє виявляти, класифікувати та виводити інформацію про геометричні об'єкти.

Об'єкт дослідження – процеси автоматизованого аналізу зображень із використанням алгоритмів комп'ютерного зору.

Предмет дослідження – методи та засоби розпізнавання геометричних фігур на зображеннях у режимі реального часу з використанням бібліотек OpenCV та графічного інтерфейсу на базі PyQt.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ЗАСОБАМИ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Комп'ютерний зір: розвиток, застосування та роль у візуальній обробці

Комп'ютерний зір – це міждисциплінарна галузь, що поєднує методи штучного інтелекту, обробки зображень та машинного навчання з метою автоматизованого аналізу, інтерпретації та розпізнавання візуальної інформації з навколишнього середовища. Простіше кажучи, комп'ютерний зір намагається навчити комп'ютери "бачити" та "розуміти" візуальні дані подібно до того, як це робить людина [1].

Історія комп'ютерного зору бере свій початок у 1960-х роках, коли перші дослідники почали експериментувати з аналізом цифрових зображень. Початкові проєкти були дуже обмеженими у функціональності – наприклад, розпізнавання простих геометричних фігур або аналіз зображень висококонтрастних сцен. Одним із перших помітних експериментів став проєкт MIT (1966), де студент мав за літо створити систему, яка розпізнає об'єкти на зображенні – завдання, яке виявилося набагато складнішим, ніж здавалося на перший погляд.

У 1980–1990-х роках відбувся розвиток теоретичної бази: було запропоновано методи фільтрації, виділення контурів (алгоритм Канні), морфологічної обробки зображень тощо. З'явилися перші промислові системи машинного зору, зокрема в галузі виробничого контролю якості.

Починаючи з 2000-х років, із поширенням цифрових камер, зростанням обчислювальної потужності та появою нових алгоритмів (SIFT, SURF, Haar-каскади) комп'ютерний зір почав активно використовуватись у більш складних застосуваннях – наприклад, у безпекових системах, автомобілях, мобільних пристроях [2].

Справжній прорив стався у 2010-х роках із появою методів глибинного навчання (deep learning). Особливо помітною була перемога нейромережі AlexNet у змаганні ImageNet (2012), де вона значно перевершила традиційні алгоритми. З

того часу convolutional neural networks (CNN), такі як VGG, ResNet, EfficientNet, стали стандартом для складних задач розпізнавання [2].

На сьогодні комп'ютерний зір – це одна з найдинамічніших галузей штучного інтелекту, що широко застосовується у:

– автономному водінні (розпізнавання дорожніх знаків, пішоходів, смуг руху),
– біометрії (ідентифікація облич, сканування сітківки),
– промисловій автоматизації (контроль якості, візуальний моніторинг),
– медицині (аналіз медичних знімків),
– доповненій реальності,
– робототехніці.

Основне завдання систем комп'ютерного зору – перетворити вхідні зображення або відео у значущу інформацію: визначити об'єкти, їхні властивості, просторове розташування та взаємозв'язки. Це досягається за допомогою низки етапів: фільтрації шуму, виявлення контурів, сегментації зображення, класифікації об'єктів та ін.

Загалом комп'ютерний зір охоплює такі ключові напрямки:

– розпізнавання об'єктів (object recognition);
– виявлення облич (face detection);
– трекінг об'єктів у відео;
– аналіз глибини та тривимірної структури сцени;
– інтерпретація сцен в реальному часі.

На відміну від традиційної обробки зображень, комп'ютерний зір не лише обробляє вхідні дані, а й прагне зрозуміти їх зміст та зробити висновки для прийняття рішень або управління процесами (наприклад, у робототехніці чи системах безпеки).

Розвиток комп'ютерного зору став можливим завдяки:

– потужним обчислювальним ресурсам (GPU, паралельні обчислення);
– відкритим бібліотекам (OpenCV, TensorFlow, Keras тощо) [3];

– наявності великих наборів даних для тренування;

– удосконаленню методів глибинного навчання (deep learning).

Комп’ютерний зірзнайшов своє застосування в багатьох сферах людської діяльності, де виникає потреба в автоматичному аналізі зображень і відео. Його інтеграція у практичні системи забезпечує машинну здатність до візуального сприйняття, що раніше було прерогативою лише людини. На рисунку 1.1 зображено ключові галузі, де технології комп’ютерного зору в комплексі зі штучним інтелектом знаходять найбільш ефективне застосування. У промисловості ці системи забезпечують автоматичний контроль якості виробів; у медицині – діагностику на основі зображень за допомогою глибинних нейромереж; у безпековій сфері – розпізнавання облич, трекінг об’єктів та аналіз поведінки; в освіті – створення адаптивних навчальних середовищ та віртуальних лабораторій; у транспорті – автономне керування, розпізнавання дорожніх знаків і ситуацій. В усіх цих сферах комп’ютерний зір виступає сенсорним модулем, а ШІ – інтелектуальним інтерпретатором даних, що забезпечує прийняття рішень та прогнозування.

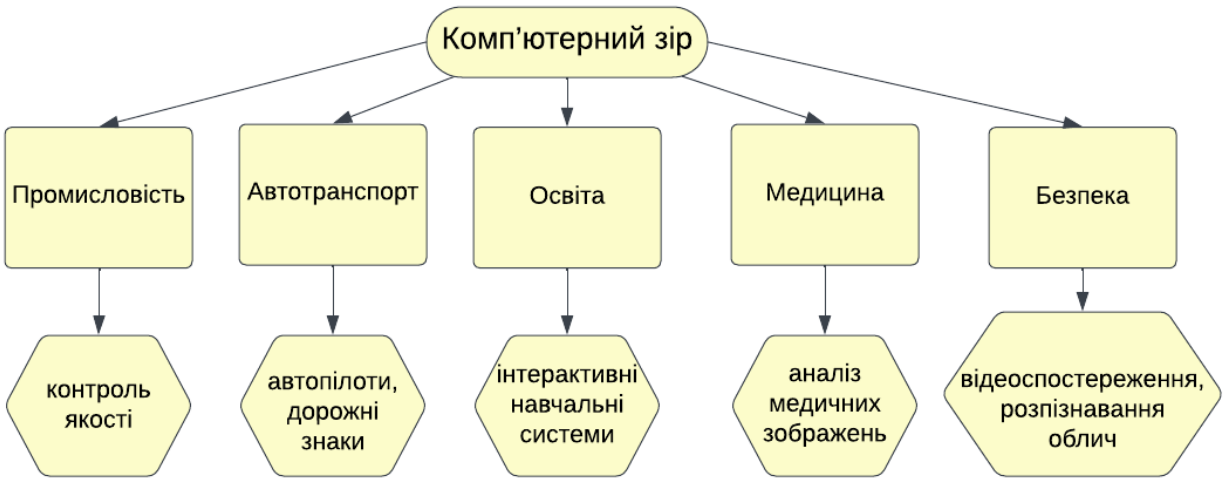


Рисунок 1.1 – Основні сфери застосування комп’ютерного зору у поєднанні зі штучним інтелектом

Однією з провідних сфер використання комп’ютерного зору є промисловість. На виробничих підприємствах системи машинного зору дозволяють

автоматизувати процес контролю якості продукції, визначати наявність дефектів, аналізувати форми, кольори, розміри об'єктів без участі оператора. Такі системи працюють із високою швидкістю та точністю, що підвищує ефективність виробництва та знижує витрати на ручну перевірку. У деяких випадках комп’ютерний зір у поєднанні з роботизованими комплексами дозволяє створювати повністю автономні лінії збирання, де кожен крок контролюється алгоритмами аналізу зображень [4].

У медицині комп’ютерний зір виконує функції, що виходять далеко за межі класичної діагностики. Сучасні алгоритми здатні аналізувати томографічні та рентгенологічні зображення з точністю, яка не поступається фахівцям. Виявлення онкологічних утворень, аномалій у структурах тканин, аналіз кровоносної системи – усе це можливо завдяки навченим моделям, які обробляють великі обсяги візуальної інформації та автоматично виділяють патогенні зміни. Комп’ютерний зір у медицині часто інтегрується із системами підтримки прийняття рішень, надаючи лікареві інструмент для швидкої та точної постановки діагнозу.

У сфері безпеки технології комп’ютерного зору стали незамінними. Вони лежать в основі систем відеоспостереження нового покоління, які не лише фіксують відео, а й здатні в режимі реального часу виявляти підозрілі дії, розпізнавати обличчя, аналізувати поведінкові шаблони, зчитувати номери автомобілів. Поєднання комп’ютерного зору з системами штучного інтелекту дозволяє створювати так звані «розумні міста», де автоматичне управління транспортом, охоронні системи та моніторинг громадських місць відбуваються за участю програмних агентів, які «бачать» світ і реагують на події відповідно до заданих сценаріїв.

Освітня сфера також не стоїть осторонь від впровадження технологій візуального аналізу. В умовах цифрової трансформації навчального процесу, комп’ютерний зір використовується для моніторингу участі студентів у заняттях, виявлення рівня їхньої залученості, розпізнавання емоційних реакцій та створення адаптивного освітнього контенту. У поєднанні з доповненою реальністю,

технології машинного зору створюють інтерактивні симуляції та тренажери, які використовуються для підготовки спеціалістів у технічних та медичних галузях [5].

Слід особливо наголосити на тому, що комп’ютерний зір нині рідко існує як ізольована технологія. Його потужність найбільш повно розкривається у поєднанні з іншими складовими штучного інтелекту. Глибинне навчання, зокрема згорткові нейронні мережі, дозволяє системам навчатися за допомогою великих обсягів даних і досягати високого рівня узагальнення. Завдяки цьому комп’ютерний зір не просто реагує на конкретні сигнали з камери, а адаптується до нових умов, коригує свої рішення на основі історії взаємодій і здатен працювати навіть у складних та змінних середовищах [6]. Наприклад, інтеграція з обробкою природної мови забезпечує можливість голосового керування системами візуального аналізу, а застосування алгоритмів підкріплювального навчання дозволяє створювати інтелектуальних агентів, здатних вчитися на основі досвіду у фізичному середовищі.

Таким чином, сучасне застосування комп’ютерного зору охоплює не лише візуальний аналіз, а й глибоку когнітивну взаємодію з навколишнім світом. Це одна з найдинамічніших галузей штучного інтелекту, що забезпечує фундамент для розвитку нових типів автономних, адаптивних і інтелектуальних систем, які формують майбутнє науки, освіти, охорони здоров’я та промисловості.

1.2 Методи та алгоритми розпізнавання об’єктів на зображеннях

Процес розпізнавання об’єктів на зображеннях є одним із ключових етапів комп’ютерного зору та полягає у виявленні, аналізі та класифікації візуальних елементів сцени. Для досягнення цієї мети використовуються різноманітні методи й алгоритми, які забезпечують автоматичну інтерпретацію піксельної інформації [7]. На практиці вибір конкретного підходу залежить від характеру завдання, якості зображень, необхідної швидкості обробки та типу об’єктів, що підлягають розпізнаванню. У контексті розпізнавання геометричних фігур особливу увагу

приділяють методам детекції контурів, аналізу кількості вершин та визначенню форми об’єкта.

Одним із найбільш поширених і базових етапів обробки є детекція контурів. Цей підхід ґрунтується на виявленні різких змін яскравості пікселів на межі об’єкта, що дозволяє локалізувати його обриси. Одним із найефективніших і водночас простих у реалізації алгоритмів є оператор Канні, який забезпечує високу точність завдяки багатоступеневій фільтрації, використанню градієнтів і процесу придушення немаксимумів. Цей метод дозволяє одержати чітке уявлення про контури об’єктів навіть на зображеннях із шумом або при слабкому контрасті. Крім Канні, у різних ситуаціях застосовуються також оператори Собеля, Прюітта, Лапласіана тощо, що мають свої переваги з точки зору обчислювальної складності та чутливості до деталей.

Після виявлення контурів наступним кроком є аналіз геометричних характеристик об’єкта. Один із підходів передбачає використання інформації про кількість вершин замкнутого контуру. Такий метод особливо ефективний при роботі з ідеальними або майже ідеальними геометричними формами: трикутниками, прямокутниками, багатокутниками, колами. Алгоритм полягає в апроксимації виявленого контуру багатокутником за допомогою функцій типу approxPolyDP з бібліотеки OpenCV [8]. На основі кількості знайдених кутів можливо ідентифікувати тип фігури – трикутник (3 вершини), прямокутник або квадрат (4 вершини), п’ятикутник (5 вершин) тощо. Цей підхід є інтуїтивно зрозумілим і добре підходить для задач навчального або демонстраційного характеру.

Параметром при класифікації об’єкта є його форма, у практиці комп’ютерного зору визначення форми виконується за допомогою таких характеристик, як ступінь округлості, співвідношення сторін, симетричність, площа, периметр, або ж через інваріанти моментів. Наприклад, для того щоб розрізнити квадрат і прямокутник, недостатньо знати лише кількість вершин – необхідно враховувати також відношення довжин сторін. Для розрізнення кола від багатокутника застосовують коефіцієнт округлості, що обчислюється як функція

від площі фігури та довжини її периметра. При розпізнаванні складніших форм часто використовують також методи машинного навчання, де класифікація здійснюється на основі векторів ознак, сформованих з описових параметрів фігури.

Розпізнавання геометричних об’єктів потребує комбінації декількох підходів: попередньої фільтрації зображення, точної детекції контурів, аналізу вершин та обчислення формальних ознак. У цьому контексті бібліотека OpenCV забезпечує широкий інструментарій для реалізації усіх зазначених етапів – від фільтрації до виводу результатів класифікації. Використання цих методів у поєднанні з графічним інтерфейсом дозволяє створювати зручні прикладні системи, здатні в режимі реального часу аналізувати сцени з веб-камери або згенеровані зображення [9].

Методи розпізнавання об’єктів на зображеннях дедалі частіше інтегрують класичні алгоритми комп’ютерного зору з сучасними технологіями штучного інтелекту. Застосування глибинного навчання дозволяє системам не лише виявляти контури чи вершини, а й навчатися на основі прикладів, автоматично адаптуючись до нових форм і умов освітлення. На рисунку представлено поєднання традиційного підходу до розпізнавання геометричних фігур із застосуванням базових елементів штучного інтелекту, яке дозволяє створити більш гнучкі та точні системи аналізу зображень у режимі реального часу (рис. 1.2).

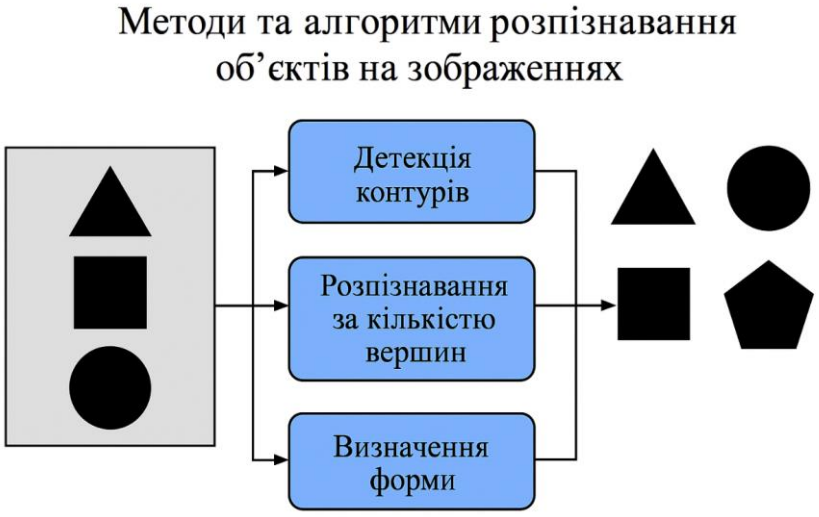


Рисунок 1.2 – Інтеграція класичних методів комп’ютерного зору та ШІ у процесі розпізнавання геометричних об’єктів

Представлена схема демонструє етапи обробки зображення: детекцію контурів, розпізнавання за кількістю вершин та визначення форми, які у поєднанні з алгоритмами машинного навчання дозволяють автоматизувати класифікацію фігур. Такий підхід є прикладом синтезу традиційної фільтраційної обробки та інтелектуального аналізу даних на основі нейронних моделей.

Одним із фундаментальних етапів у розпізнаванні об’єктів на зображеннях є їх попередня обробка, яка має на меті поліпшити якість вхідних даних, зменшити вплив шуму та виділити релевантні особливості сцени. Ефективність подальших етапів – таких як класифікація об’єктів чи побудова ознак – безпосередньо залежить від якості виконаної попередньої обробки [9]. Це допомагають робити алгоритми фільтрації, сегментації та детекції контурів, які формують основу для точного виділення об’єктів на зображенні.

На першому етапі зазвичай застосовуються фільтри згладжування для зменшення шуму, який часто виникає внаслідок зовнішніх умов (недостатнє освітлення, вібрації, оптичні спотворення). Серед найпоширеніших фільтрів можна виділити гаусів фільтр, який реалізує згладжування за нормальним розподілом, та медіанний фільтр, який ефективно видаляє імпульсний шум (так звані «сіль і перець»), зберігаючи при цьому краї зображення. Використання фільтрації дозволяє знизити ймовірність хибних спрацьовувань алгоритмів виявлення країв, підвищуючи стабільність системи.

Наступним етапом є сегментація зображення, тобто розділення сцени на області, що мають спільні характеристики – за кольором, текстурою, яскравістю чи просторовим положенням. Залежно від типу зображення та завдання сегментація може виконуватися різними методами – пороговою бінаризацією, розкладанням за кластерами (наприклад, алгоритм k-means), вододільним перетворенням або ж з використанням глибинних нейронних мереж для семантичної сегментації. У завданнях розпізнавання геометричних фігур найбільш ефективною часто є проста порогова сегментація, коли зображення переводиться у двійковий формат, де яскраві області відповідають фігурам, а темні – фону. Це значно спрощує подальшу обробку та дозволяє працювати зі структурами більшого масштабу.

Основним етапом є виділення контурів (edge detection), яке дозволяє точно окреслити межі об’єктів. Визначення контурів здійснюється за допомогою градієнтного аналізу – оцінки змін яскравості у піксельному просторі. Найвідомішим і найефективнішим у цьому аспекті є алгоритм Кенні, який виконує детекцію контурів у кілька етапів: розмиття зображення, обчислення градієнтів, придушення немаксимумів і подвійна порогова фільтрація з трасуванням з’єднань [10]. Алгоритм забезпечує високу точність виявлення навіть при наявності слабоконтрастних або частково зашумлених фігур. У більш простих випадках можуть використовуватися оператори Собеля, Робертса або Лапласіана, які мають нижчу обчислювальну складність, але також ефективні у попередній локалізації об’єктів.

У системах, що використовують штучний інтелект, традиційні алгоритми обробки зображень часто комбінуються з попередньо навченими моделями, які виконують сегментацію або детекцію контурів як частину глибокої згорткової мережі. Наприклад, архітектура U-Net забезпечує сегментацію з високою точністю завдяки симетричній структурі, що дозволяє моделі враховувати як локальні, так і глобальні характеристики зображення [11].

Алгоритми попередньої обробки зображень становлять невід’ємну складову ефективного комп’ютерного зору, забезпечуючи необхідну підготовку даних для подальшого інтелектуального аналізу. Їх грамотне поєднання дозволяє адаптувати систему до різних умов – від лабораторних до реальних, із шумами та викривленнями, – що є особливо важливим у задачах розпізнавання геометричних об’єктів у реальному часі.

Процес розпізнавання об’єктів у сучасних інтелектуальних системах комп’ютерного зору реалізується шляхом послідовної обробки зображення з використанням алгоритмів, що поєднують класичні методи та моделі штучного інтелекту. На першому етапі застосовується сегментація для відокремлення об’єктів від фону, після чого виконується детекція контурів, що забезпечує точне окреслення меж фігур. Завершальним кроком є класифікація об’єктів за допомогою алгоритмів, які можуть базуватись як на геометричних характеристиках

(наприклад, кількість вершин чи ступінь округлості), так і на нейронних моделях, навчених на відповідних вибірках [12]. Завдяки інтеграції цих етапів з компонентами ШІ, система здатна адаптивно аналізувати зображення навіть в умовах змінного освітлення, шуму чи спотворень, демонструючи стабільну точність у реальному часі (рис. 1.3).

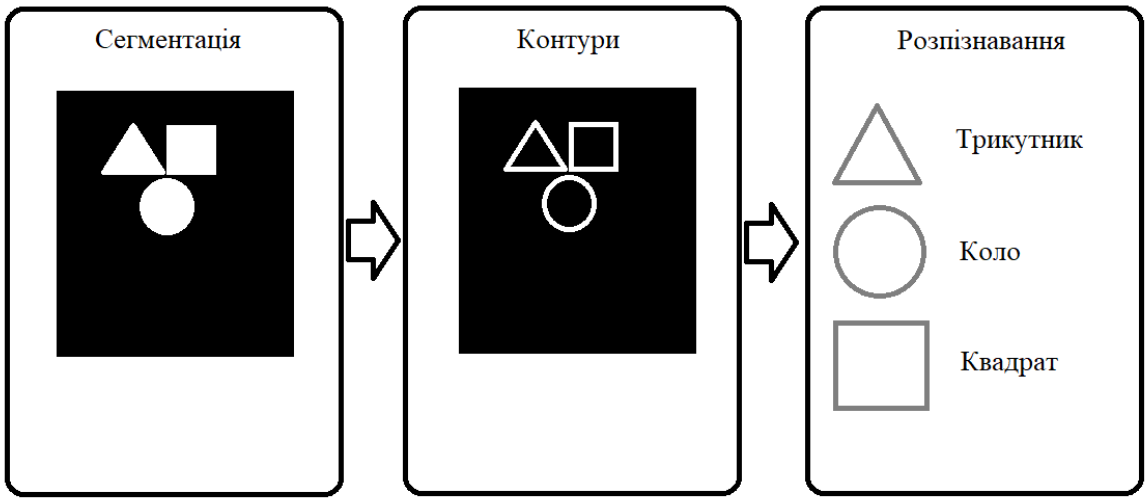


Рисунок 1.3 – Етапи обробки зображення для розпізнавання фігур із використанням методів штучного інтелекту

Схема демонструє ключові компоненти процесу комп'ютерного зору: сегментацію сцени, детекцію контурів та класифікацію геометричних об'єктів. Застосування моделей штучного інтелекту дозволяє забезпечити автоматичне, надійне та масштабоване розпізнавання фігур, навіть за наявності візуальних спотворень або шуму.

1.3 Огляд бібліотек OpenCV та PyQt для побудови систем комп'ютерного зору

OpenCV (Open Source Computer Vision Library) – це потужна, кросплатформна бібліотека з відкритим вихідним кодом, яка широко використовується для реалізації завдань комп'ютерного зору, цифрової обробки зображень та машинного навчання. Вона підтримує десятки мов програмування, зокрема C++, Java, Python, і включає понад 2500 оптимізованих алгоритмів.

Завдяки високій продуктивності та широкому функціоналу, OpenCV є галузевим стандартом для створення як дослідницьких, так і комерційних застосунків [13].

На рисунку 1.4 представлено приклад класифікації геометричних об’єктів на основі контурного аналізу, реалізованої з використанням бібліотеки OpenCV. Цей підхід поєднує класичні методи комп’ютерного зору з алгоритмічною логікою, що є базовим компонентом інтелектуальних систем розпізнавання зображень.

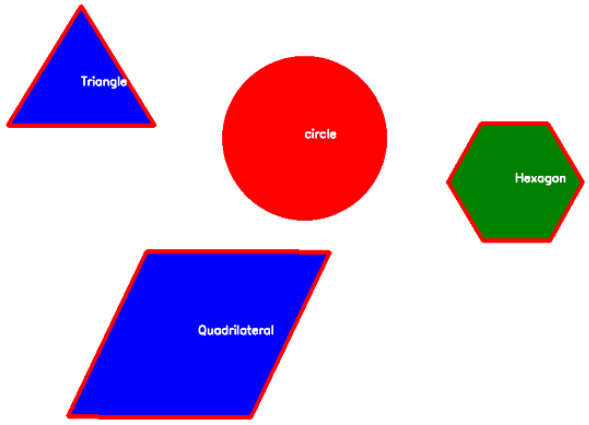


Рисунок 1.4 – Розпізнавання геометричних фігур за допомогою методів комп’ютерного зору на базі бібліотеки OpenCV як складової систем штучного інтелекту

Однією з ключових переваг OpenCV є її модульність і підтримка всіх етапів обробки візуальної інформації – від захоплення зображень до високорівневої інтерпретації даних. На етапі попередньої обробки OpenCV надає засоби для фільтрації шумів (Гаусів фільтр, медіанна фільтрація), перетворення кольорових просторів (BGR → Grayscale, HSV), нормалізації яскравості, контрасту, гістограмного вирівнювання, що дозволяє суттєво покращити якість зображення для подальшого аналізу [14].

Для детекції та сегментації об’єктів бібліотека пропонує широкий набір методів. Зокрема, функції cv2.Canny() для виявлення контурів, cv2.findContours() для побудови кривих об’єктів, cv2.threshold() і cv2.adaptiveThreshold() для бінаризації зображення та cv2.HoughCircles() або cv2.HoughLines() для виявлення простих геометричних об’єктів. Завдяки цим інструментам можливе точне

виявлення меж, визначення структурних елементів сцени та подальша геометрична інтерпретація зображення.

Крім того, OpenCV містить функціонал для геометричних трансформацій зображення: масштабування, обертання, афінні й перспективні перетворення. Це забезпечує інваріантність до положення об’єкта, що є критичним у завданнях реального часу або при обробці відео з камери.

Особливо важливими є можливості для аналізу форми об’єктів. Наприклад, за допомогою функцій cv2.approxPolyDP() можна апроксимувати контури багатокутниками та аналізувати кількість вершин. Також доступні обчислення площі (cv2.contourArea()), периметра (cv2.arcLength()), мінімальної прямокутної або еліптичної області, що охоплює фігуру, та обчислення моментів (cv2.moments()), які використовуються для обчислення центру мас або Hu-моментів для інваріантного розпізнавання [15].

Для роботи з відеопотоками OpenCV підтримує захоплення кадрів у реальному часі за допомогою cv2.VideoCapture() та обробку кожного кадру у циклі, що дозволяє реалізовувати динамічні системи виявлення, трекінгу та взаємодії з користувачем.

OpenCV також інтегрується з такими бібліотеками, як NumPy (для чисельних обчислень), TensorFlow і PyTorch (для глибинного навчання), що дозволяє розширити можливості обробки та додати інтелектуальні компоненти на основі нейромереж.

OpenCV, дозволяє реалізувати повний цикл обробки зображень – від низькорівневої обробки до високорівневої інтерпретації. Її застосування у поєднанні з інтерфейсними рішеннями (наприклад, PyQt) формує основу для створення гнучких, інтерактивних систем комп’ютерного зору, здатних до ефективної роботи як в експериментальних умовах, так і в промислових сценаріях.

На рис. 1.5 зображено основні компоненти програмного застосунку: обробка зображень і детекція об’єктів за допомогою бібліотеки OpenCV, інтеграція з інтерфейсом користувача через PyQt, що забезпечує візуалізацію результатів у режимі реального часу. Такий підхід поєднує класичні методи комп’ютерного зору

з інструментами графічного програмування, утворюючи інтелектуальну систему з інтерфейсною взаємодією.

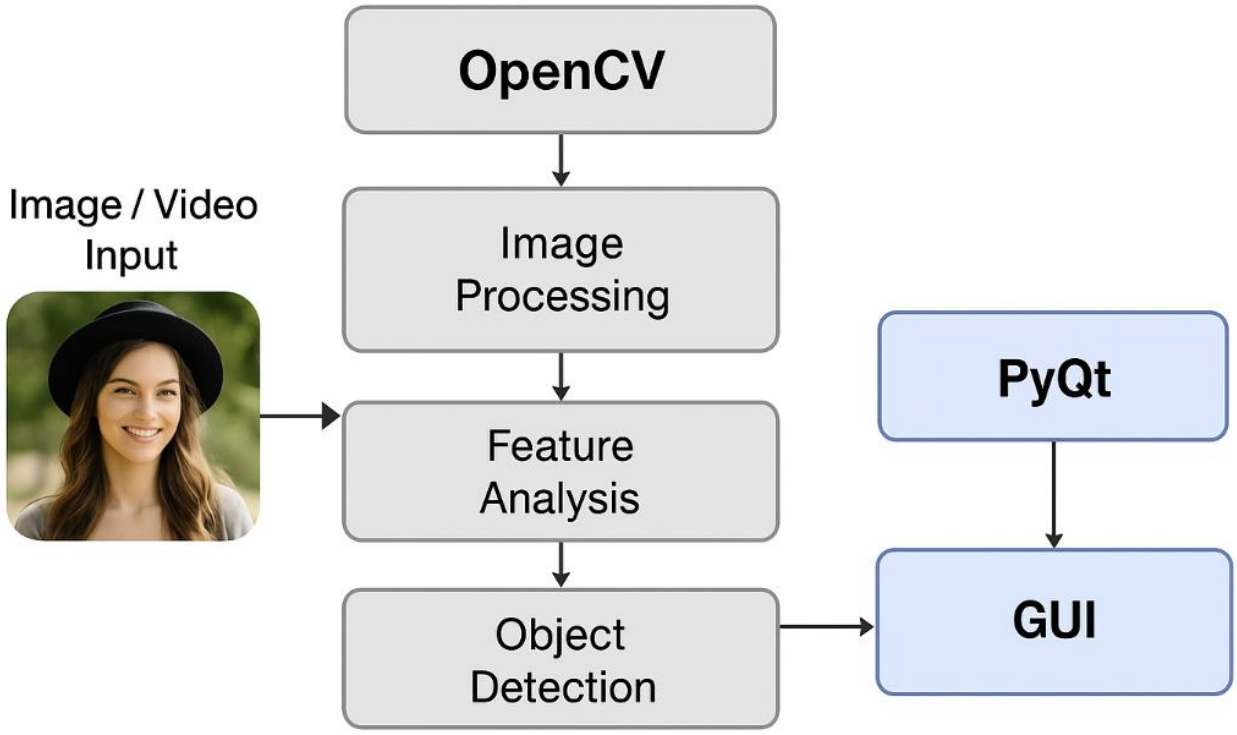


Рисунок 1.5 – Архітектура програмного застосунку для комп’ютерного зору

PyQt – це набір прив’язок до інструментарію Qt, який дозволяє створювати кросплатформні графічні інтерфейси користувача за допомогою мови програмування Python. У контексті розробки систем комп’ютерного зору PyQt забезпечує зручне середовище для інтерактивної взаємодії користувача з системою розпізнавання: від вибору джерела зображення до візуалізації оброблених результатів.

PyQt дозволяє реалізовувати вікна з інтерактивними елементами управління – кнопками, чекбоксами, випадаючими списками, графічними панелями тощо. Бібліотека підтримує обробку подій, асинхронну взаємодію, кастомізацію стилів інтерфейсу та можливість інтеграції з бібліотеками обробки зображень (зокрема OpenCV). Завдяки PyQt розробник може створювати програму, зручну для кінцевого користувача, поєднуючи високий рівень функціональності з інтуїтивно

зрозумілим дизайном.

У системах комп'ютерного зору PyQt – це інтерфейсний шар, який зв'язує користувача із внутрішньою логікою обробки зображень, побудованою на базі OpenCV або інших бібліотек. Завдяки такій інтеграції формується завершений, повноцінний застосунок – не лише технічно функціональний, а й практично придатний до використання в реальних умовах. Порівняння OpenCV та PyQt наведено в табл. 1.1.

Таблиця 1.1 – Переваги бібліотек OpenCV та PyQt

Критерій	**OpenCV**	**PyQt**
Основне призначення	Обробка та аналіз зображень, відео	Створення графічного інтерфейсу користувача (GUI)
Мова програмування	Python, C++, Java	Python
Підтримка відеопотоку	Є (через cv2.VideoCapture())	Непряма (через інтеграцію з OpenCV)
Робота в реальному часі	Оптимізовано для обробки кадрів в реальному часі	Забезпечує оновлення вікна GUI при кожному кадрі
Гнучкість налаштування	Висока – підтримує низькорівневі і високорівневі операції	Висока – підтримує кастомізацію елементів інтерфейсу та стилізацію
Складність засвоєння	Середня (потребує знання базових алгоритмів обробки зображень)	Середня (необхідні знання структури Qt-сигналів і слотів)
Документація та спільнота	Широка документація, активна спільнота	Потужна документація, інтеграція з Qt Designer, активна підтримка розробників
Кросплатформність	Повна (Windows, Linux, macOS, Android)	Повна (Windows, Linux, macOS)
Інтеграція з іншими бібліотеками	TensorFlow, PyTorch, NumPy, scikit-image	OpenCV, Matplotlib, Pandas, SQLAlchemy, тощо
Ідеальне застосування	Детекція, класифікація, сегментація, трекінг об'єктів	Користувацький контроль, відображення результатів, керування параметрами обробки

Висновок до першого розділу

Розглянуто теоретичні засади побудови систем комп’ютерного зору, які становлять основу для реалізації програмного застосунку з розпізнавання зображень. Проаналізовано сутність та етапи розвитку комп’ютерного зору як міждисциплінарного напряму, що об’єднує методи обробки зображень, штучного інтелекту та машинного навчання. Визначено основні сфери застосування комп’ютерного зору – від промисловості до медицини й освіти – з акцентом на його інтеграцію у сучасні інтелектуальні системи.

Описано принципи виявлення контурів, аналізу геометричних ознак фігур (кількість вершин, форма, округлість), а також інваріантні характеристики, що забезпечують стійкість до обертання, масштабування та зашумлення. Розглянуто комбінацію класичних алгоритмів із моделями штучного інтелекту, які дозволяють підвищити точність та адаптивність систем комп’ютерного зору.

Проведено огляд функціональних можливостей бібліотек OpenCV і PyQt, які є ключовими інструментами для реалізації систем візуального аналізу. OpenCV забезпечує ефективну обробку зображень, детекцію та аналіз об’єктів, тоді як PyQt дозволяє створювати інтерактивний графічний інтерфейс користувача, що забезпечує зручність взаємодії із застосунком.

2 АНАЛІЗ ТА ПРОЄКТУВАННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ГЕОМЕТРИЧНИХ ФІГУР

2.1 Функціональна структура та модульна організація застосунку

Розроблений програмний застосунок призначений для автоматичного розпізнавання геометричних фігур у режимі реального часу або в рамках згенерованої сцени. Його архітектура побудована за принципом модульності, що дозволяє гнучко масштабувати, тестувати та підтримувати систему, а також забезпечує чіткий поділ відповідальності між компонентами. Застосунок реалізований мовою Python з використанням бібліотек OpenCV (для обробки зображень) і PyQt (для створення графічного інтерфейсу користувача) [16].

Загалом програмна система складається з п’яти функціональних блоків:

– інтерфейс користувача (GUI) — реалізований за допомогою PyQt5, відповідає за візуалізацію даних, керування камерою та генерацією сцени;

– модуль генерації сцен — створює випадкові сцени із зазначеним числом фігур, з уникненням перетинів і дублювань;

– модуль обробки зображень (OpenCV) — виконує фільтрацію, бінаризацію, детекцію контурів та формальну класифікацію об’єктів;

– модуль потокового захоплення відео — забезпечує безперервне захоплення кадрів із веб-камери, передає їх для обробки та оновлює інтерфейс;

– модуль логіки розпізнавання фігур — визначає тип фігури на основі кількості вершин, співвідношення сторін та показника округлості.

Щоб забезпечити гнучкість, масштабованість і логічну розділеність функцій, застосунок побудовано на основі модульної архітектури. Кожен модуль реалізує окрему частину функціональності — від обробки відеопотоку та генерації сцен до аналізу геометричних ознак об’єктів і відображення результатів у графічному інтерфейсі. Такий підхід полегшує підтримку коду, повторне використання компонентів і підвищує зручність у тестуванні. На рисунку 2.1 представлено

загальну схему модульної організації програмного застосунку, де центральний блок взаємодіє з окремими логічними складовими.

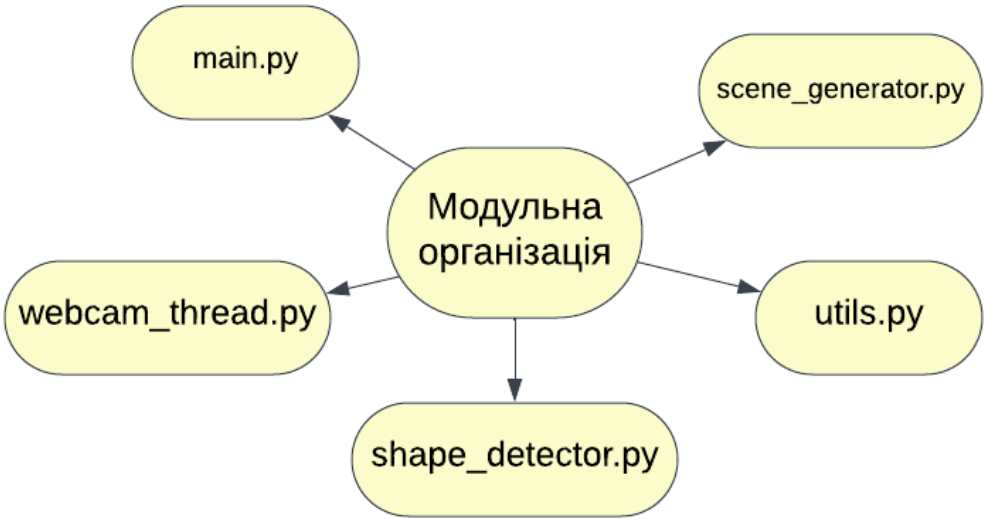


Рисунок 2.1 – Схема модульної організації програмного застосунку для розпізнавання геометричних фігур

Модульна організація:

main.py – це основний вхідний файл, який ініціалізує застосунок, запускає GUI, створює об’єкти потоків і підключає сигнали/слоти між модулями. Саме тут здійснюється інтеграція логіки PyQt з OpenCV.

scene_generator.py – цей модуль відповідає за генерацію сцени з випадковим розташуванням фігур. Він створює зображення (canvas), на якому розміщуються випадкові фігури з урахуванням мінімальної відстані між ними (min_gap) та унікальності координат (перевірка функцією bboxes_overlap). Кожна фігура генерується з випадковим кольором та збереженням інформації (тип, центр, розміри) [17].

webcam_thread.py – модуль відповідає за асинхронну роботу з веб-камерою через клас QThread. Камера захоплює поточні кадри, що передаються у функцію розпізнавання. У випадку невдачі (наприклад, веб-камера не підключена), генерується повідомлення про помилку. Потік посилає сигнали з оновленими кадрами, які зображуються у GUI.

shape_detector.py – основний логічний модуль розпізнавання. Використовуючи метод cv2.findContours та апроксимацію контуру cv2.approxPolyDP, система визначає кількість вершин. Для 3 вершин — трикутник, для 4 — квадрат або прямокутник (на основі співвідношення сторін), 5 — п’ятикутник, більше 5 — коло або багатокутник (за показником округлості). Всі параметри фігури зберігаються в об’єкті словника: type, area, perimeter, bbox, center, circularity.

utils.py – допоміжний модуль, що містить функції форматування інформації (format_shape_info) та конвертації зображень між форматами OpenCV і Qt (convert_cv_qt), що дає змогу коректно відображати кадри у вікні GUI [18].

Функціональна взаємодія компонентів

При запуску застосунку користувач має змогу:

– увімкнути камеру й розпочати обробку відеопотоку;

– встановити кількість фігур у сцені та згенерувати її;

– отримати візуалізацію результату: фігури підписані відповідними назвами, виділені контуром;

– отримати числову інформацію про кожну фігуру у лівому полі (рис. 2.2).

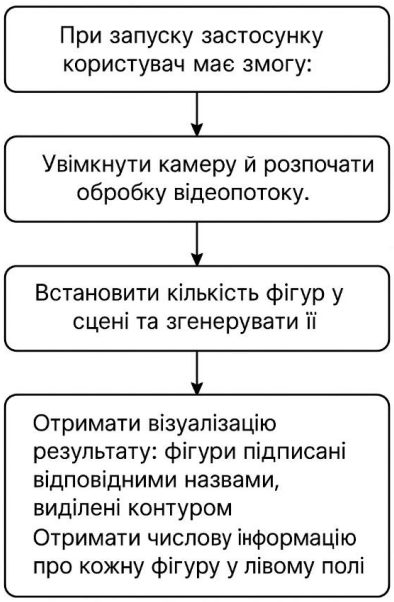


Рисунок 2.2 – Блок-схема функціональної взаємодії компонентів застосунку

Процес обробки проходить наступні стадії:

– захоплення зображення (кадр з веб-камери або згенерованої сцени);

– перетворення зображення у сірий формат, бінаризація;

– виявлення контурів;

– аналіз форми: визначення кількості вершин, співвідношення сторін, розрахунок округлості;

– класифікація типу фігури та виведення інформації.

Модульна структура застосунку забезпечує його масштабованість, зручність у відлагодженні та супроводі. Кожен компонент виконує чітко визначену функцію, що відповідає архітектурним принципам високої зв’язаності та низької залежності між модулями. Це робить систему гнучкою та готовою до розширення — наприклад, інтеграції з іншими алгоритмами машинного навчання або глибокого розпізнавання об’єктів.

2.2 Алгоритмічні принципи та логіка розпізнавання геометричних фігур

Процес розпізнавання геометричних фігур у межах даного застосунку реалізований з використанням комбінації класичних алгоритмів комп’ютерного зору на основі бібліотеки OpenCV. Основна ідея полягає у поетапній обробці зображення: починаючи з попередньої фільтрації та виділення контурів і завершуючи класифікацією об'єктів за геометричними ознаками. Такий підхід дозволяє ефективно виявляти фігури в згенерованій сцені або у відеопотоці, включаючи випадки перекосів, шумів чи різного освітлення [19].

Основні етапи обробки:

1. Перетворення зображення в градації сірого

Вхідне зображення, отримане з веб-камери або згенерованої сцени, конвертується у відтінки сірого методом cv2.cvtColor. Це дозволяє спростити структуру даних, з якими працюють алгоритми сегментації та виявлення контурів.

2. Бінаризація зображення (thresholding)

Після конвертації зображення застосовується порогове перетворення (cv2.threshold або adaptiveThreshold), внаслідок чого фігури виділяються як білі

об’єкти на чорному фоні. Цей крок є критичним для точного виявлення об’єктів, оскільки формує основу для подальшої сегментації.

3. Виділення контурів

За допомогою функції cv2.findContours зображення аналізується на наявність замкнутих контурів. Кожен виявлений контур інтерпретується як потенційна фігура.

4. Апроксимація контурів

Отримані контури апроксимуються багатокутниками за допомогою алгоритму Дугласа-Пекера (cv2.approxPolyDP), що дозволяє зменшити кількість точок у контурі та визначити кількість вершин [20]. Це є основним критерієм класифікації фігури:

– 3 вершини – трикутник;
– 4 вершини – квадрат або прямокутник;
– 5 – п’ятикутник;
– >5 – коло або багатокутник.

5. Розрахунок геометричних параметрів

Для кожної фігури обчислюються:

– площа (area) – функція cv2.contourArea;
– периметр (arc length) – cv2.arcLength;
– центр мас – через геометричні моменти (cv2.moments);
– обрамлюючий прямокутник (bounding box) – cv2.boundingRect.

6. Класифікація фігури

На основі кількості вершин та співвідношення сторін визначається тип фігури. Наприклад, якщо чотири вершини, але відношення ширини до висоти в межах 0.9–1.1, то фігура класифікується як квадрат. Якщо це співвідношення виходить за межі — фігура вважається прямокутником.

7. Оцінка округлості

Для об’єктів з більш ніж 5 вершинами розраховується показник округлості (circularity) за формулою:

$$C = \frac{4\pi * \text{Площа}}{(\text{Периметр})^2}$$

Значення близьке до 1 свідчить про наявність круглої форми.

8. Формування результату

Після класифікації для кожної фігури формується структура даних із зазначенням її типу, геометричних параметрів, координат центра, розмірів обрамлення та, за наявності, показника округлості. Ця інформація передається до інтерфейсу для відображення користувачеві.

Приклад логіки класифікації показано на рис. 2.3

```
if num_vertices == 3:
    shape_type_ukrainian = "Трикутник"
elif num_vertices == 4:
    if 0.9 <= aspect_ratio <= 1.1:
        shape_type_ukrainian = "Квадрат"
    else:
        shape_type_ukrainian = "Прямокутник"
elif num_vertices == 5:
    shape_type_ukrainian = "П’ятикутник"
elif num_vertices > 5:
    if 0.80 < circularity < 1.20:
        shape_type_ukrainian = "Коло"
    else:
        shape_type_ukrainian = "Багатокутник"
```

Рисунок 2.3 – Фрагмент коду класифікації фігур згідно з кількістю вершин

Варто зазначити, що реалізований алгоритм не тільки базується на геометричних ознаках, а й враховує похибки форми — наприклад, перекоси або часткову деформацію контуру [21]. Це дозволяє досягати стійкої роботи системи навіть у непередбачуваних умовах зйомки. Окрім того, відкритість архітектури дає змогу розширити систему шляхом додавання моделей глибинного навчання для складніших об’єктів.

Для реалізації процесу розпізнавання геометричних фігур було застосовано чітко структурований алгоритм, що складається з кількох логічних етапів: від

отримання зображення до виводу класифікованих об’єктів. Алгоритм враховує основні особливості фігур, зокрема кількість вершин, співвідношення сторін та показник округлості, що дозволяє визначати тип фігури з достатньою точністю. На рис. 2.4 представлено блок-схему алгоритмічної логіки, реалізованої у програмному застосунку.

Рисунок 2.4 – Блок-схема алгоритму розпізнавання геометричних фігур у системі комп’ютерного зору

Схема відображає послідовність обробки зображення: перетворення у сірий формат, бінаризація, виявлення контурів, визначення кількості вершин, перевірка пропорцій, оцінка округлості та класифікація об’єкта. Такий підхід дозволяє точно розпізнавати трикутники, квадрати, прямокутники, кола та багатокутники на основі простих геометричних ознак.

2.3 Використання потокової обробки у роботі з відеопотоком

Однією з ключових вимог до сучасних застосунків комп'ютерного зору є здатність працювати з відеопотоком у реальному часі, забезпечуючи стабільну частоту кадрів, низьку затримку та безперервність візуалізації результатів. Для цього в розробленому програмному рішенні реалізовано потокову обробку відео з використанням багатопотокового програмування засобами PyQt5 [22].

Реалізація обробки відеопотоку винесена в окремий клас WebcamThread, що наслідує QThread. Це дозволяє ізолювати операції захоплення зображень від основного GUI-потоку, уникнувши блокування інтерфейсу при обробці кадрів (рис. 2.5).

```
class WebcamThread(QThread):
    frame_ready = pyqtSignal(QPixmap)
    ...
    def run(self):
        self.cap = cv2.VideoCapture(self.camera_index)
        while self.running:
            ret, frame = self.cap.read()
            if ret:
                # Обробка кадру
```

Рисунок 2.5 – Архітектура потокового обслуговування

Використання pyqtSignal забезпечує асинхронну передачу даних між потоком камери та інтерфейсом користувача. Це дозволяє оновлювати зображення у вікні без необхідності ручного очікування завершення обробки.

Обробка кадру в реальному часі

Кожен кадр, отриманий через cv2.VideoCapture, надсилається до основного модуля обробки (shape_detector.detect_shapes) [23]. Там здійснюється:

1. Перетворення кадру у відтінки сірого;
2. Застосування порогової фільтрації;
3. Виявлення контурів та їх аналіз;

4. Формування інформаційного словника по кожній фігурі;

5. Повернення кадру з підписаними та виділеними фігурами.

Результати передаються через сигнал processed_frame_ready у вигляді об’єкта QPixmap, що дає змогу миттєво оновлювати візуальну панель інтерфейсу користувача [24].

Система захищена від нестабільності потоку шляхом обробки винятків. У разі втрати доступу до камери, система видає повідомлення та припиняє захоплення кадрів:

```
except Exception as e:
print(f"Error initializing camera: {e}")
self.cap = None
```

Це дозволяє уникати збоїв під час роботи застосунку на різних пристроях або у випадку апаратних проблем.

Продуктивність та частота оновлення

Потокова реалізація дозволяє підтримувати частоту обробки кадрів на рівні 10–15 FPS, залежно від потужності ПК, що є прийнятним для задач розпізнавання простих геометричних об’єктів. Оскільки кадри обробляються без збереження у файлову систему, затримка мінімізується, що дає змогу досягти ефекту «живої» обробки [25].

Взаємодія з графічним інтерфейсом

Інтерфейс програми безперервно оновлюється новими кадрами, а інформаційне поле в лівій панелі виводить список розпізнаних фігур, що дозволяє користувачеві спостерігати за результатами в реальному часі. Це особливо зручно для інтерактивного тестування алгоритмів та налагодження їх точності.

Висновок до другого розділу

Здійснено повний аналіз архітектури та принципів проєктування програмного застосунку для розпізнавання геометричних фігур з використанням методів комп’ютерного зору. Розглянуто функціональну структуру та модульну організацію системи, що дозволяє досягти високої гнучкості, масштабованості та простоти супроводу програмного коду. Застосунок побудований на базі бібліотек OpenCV і PyQt, що забезпечує ефективне поєднання обробки зображень у реальному часі з зручним графічним інтерфейсом користувача.

Алгоритмічні основи розпізнавання були реалізовані через послідовне виконання ключових етапів: попередньої обробки зображення, виявлення контурів, аналізу кількості вершин, розрахунку геометричних характеристик та класифікації об’єктів. Особливу увагу приділено адаптації алгоритмів до роботи в умовах реального часу, з урахуванням шумів, змін освітлення та часткових спотворень.

Окремий акцент зроблено на реалізації потокової обробки відеопотоку за допомогою механізму багатопотоковості (через QThread). Це дозволило ізолювати обчислювальні процеси від графічного інтерфейсу та забезпечити стабільне захоплення, обробку та виведення кадрів у реальному часі без блокування основного GUI-потоку.

У результаті проведеного аналізу та проєктування було сформовано надійну архітектурну і алгоритмічну основу програмної системи, яка відповідає вимогам до застосунків комп’ютерного зору в контексті розпізнавання об’єктів із чітко вираженою геометрією. Це створює передумови для переходу до етапу тестування, оцінки точності та практичного використання розробленої системи.

3 РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

3.1 Структура реалізації та архітектура програмної системи

Реалізація програмного застосунку для розпізнавання зображень здійснювалась із використанням методів комп'ютерного зору, що забезпечують автоматичну обробку, аналіз та інтерпретацію візуальної інформації. Архітектура системи побудована за модульним принципом, що забезпечує масштабованість, розширюваність та зручність у підтримці. Основні компоненти включають модулі захоплення відеопотоку, попередньої обробки кадрів, виявлення контурів, класифікації геометричних фігур, а також інтерфейс користувача для взаємодії із системою. Для реалізації функціоналу застосунку було обрано Python з використанням бібліотек OpenCV, NumPy та PyQt, що забезпечує ефективну обробку зображень і зручний графічний інтерфейс. У цьому підрозділі буде розглянуто деталі технічної реалізації ключових модулів та загальну архітектурну структуру системи [26].

Проєктна структура програмної системи представлена на рисунку 3.1. У кореневій директорії проєкту `SQUARE` містяться основні модулі застосунку, кожен з яких відповідає за окрему частину функціоналу.

Зокрема:

- ***`main.py`*** — головний виконавчий файл, з якого розпочинається робота застосунку, ініціалізує основні компоненти та запускає графічний інтерфейс;
- ***`scene_generator.py`*** — модуль генерації сцен, що використовується для створення тестових або синтетичних зображень фігур;
- ***`shape_detector.py`*** — модуль розпізнавання геометричних фігур, який реалізує алгоритми комп'ютерного зору для класифікації об'єктів на зображенні;
- ***`utils.py`*** — допоміжний модуль, який містить службові функції, що використовуються іншими компонентами системи (наприклад, функції для обробки зображень або перетворення форматів);

– ***webcam_thread.py*** — модуль, що реалізує обробку відеопотоку з вебкамери в окремому потоці, забезпечуючи безперервне зчитування кадрів без блокування основного інтерфейсу;
– ***requirements.txt*** — текстовий файл зі списком залежностей, необхідних для встановлення і запуску проєкту у віртуальному середовищі.

Також присутні директорії `.venv` і `.venv1`, які містять файли віртуального середовища Python для ізоляції бібліотек проєкту [26].

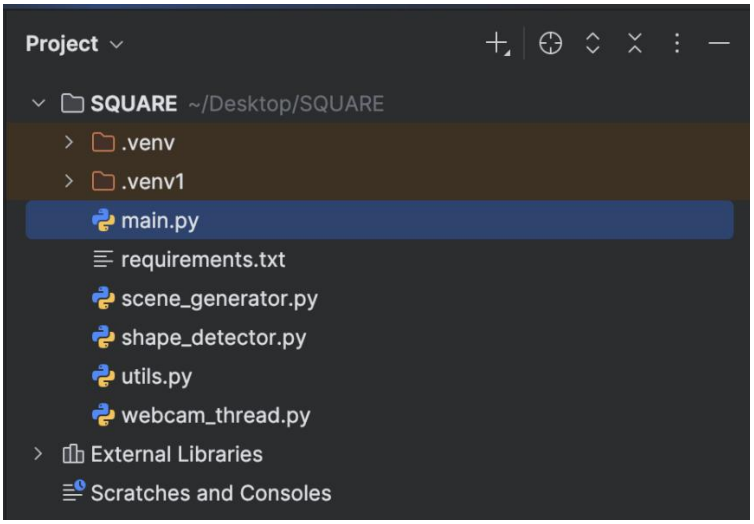


Рисунок 3.1 – Структура проєкту застосунку для розпізнавання зображень

Основний модуль main.py (рис. 3.2). У файлі main.py реалізовано головне графічне вікно застосунку за допомогою фреймворку PyQt5. Центральний клас ShapeSenseApp, що наслідує QMainWindow, виконує роль головного контролера застосунку. Він ініціалізує інтерфейс, взаємодіє з потоками відеозахоплення та відповідає за обробку подій [27].

У методі `__init__` відбувається:

– Встановлення назви вікна `"Детектор фігур"` та фіксованого розміру 1250×750 пікселів;
– Ініціалізація прапорців `is_webcam_running`, `generated_canvas`, та списку `ground_truth_shapes` для відстеження стану відеопотоку та згенерованих фігур;

– Підключення об’єкта `WebcamThread` — окремого потоку для роботи з вебкамерою. Через сигнально-слотову модель Qt здійснюється обробка різних подій: надходження кадрів (`frame_ready`), порогових зображень (`threshold_frame_ready`), та виявлених фігур (`shapes_info_ready`);

– Виклик методів `initUI()` та `set_initial_pixmaps()`, які налаштовують зовнішній вигляд вікна та початкові зображення.

У методі `initUI()`:

– Створюється центральний віджет та основне компонування за допомогою `QVBoxLayout`;

– Ліва панель управління має фіксовану ширину 320 пікселів;

– Додається кнопка “Увімкнути Веб-камеру”, яка підключена до слоту `toggle_webcam`, що відповідає за активацію/деактивацію потокового відео;

– Додається розмітка з текстовим заголовком для блоку керування камерою [28].

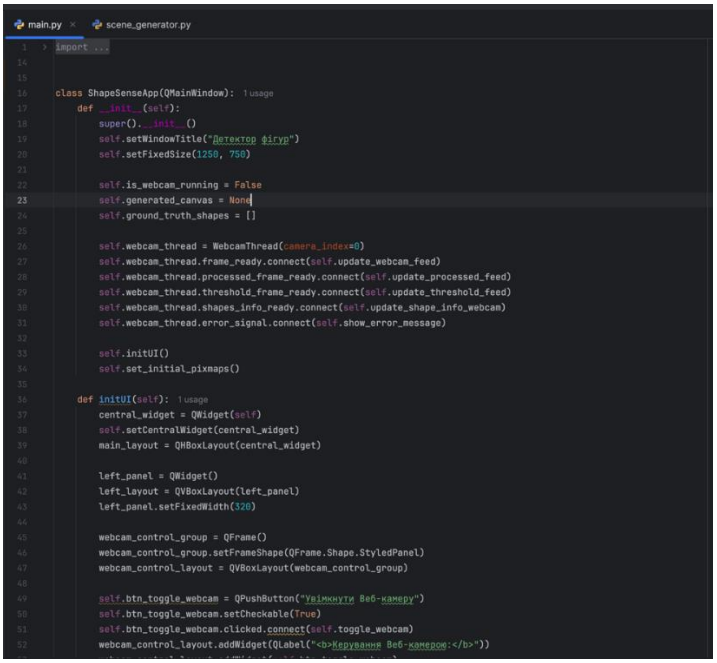


Рисунок 3.2 – Фрагмент головного модуля main.py із класом ShapeSenseApp, відповідального за запуск інтерфейсу та обробку відеопотоку

Модуль генерації сцен scene_generator.py (рис. 3.3). Модуль scene_generator.py відповідає за створення випадкових тестових сцен із фігурами,

які використовуються для перевірки точності роботи алгоритмів розпізнавання. Центральна функція — generate_random_scene(width, height, shape_config) — генерує зображення з випадково розміщеними фігурами на білому тлі, використовуючи задані параметри розміру, кількості та типів об’єктів.

На початку функції:

- Створюється біле полотно розміру `width × height` (типу `uint8`);
- Визначаються параметри мінімального та максимального розміру фігур (`min_size`, `max_size`), відступи до меж зображення (`border`), а також мінімальний інтервал між об'єктами (`min_gap`);
- Списки `ground_truth` та `existing_bboxes` використовуються для зберігання інформації про вже згенеровані об'єкти та перевірки на перекриття.

Функція `bboxes_overlap()` забезпечує перевірку на перетин bounding box-ів, запобігаючи накладанню фігур одна на одну. Вона враховує мінімальний допустимий зазор між об'єктами [27-29].

Формується список `all_shapes_to_generate` на основі вхідного словника `shape_config`, який містить типи фігур та кількість кожного типу. Після перемішування порядок генерації стає випадковим.

У циклі по кожному об’єкту:

- Присвоюється унікальний `shape_id`;
- Створюється словник `info`, що містить метаінформацію про фігуру (ідентифікатор, тип);
- Визначається випадковий колір об’єкта.

Цей модуль виконує важливу роль у тестуванні — дозволяє генерувати контрольовані сцени для подальшого використання у порівняльному аналізі результатів розпізнавання.

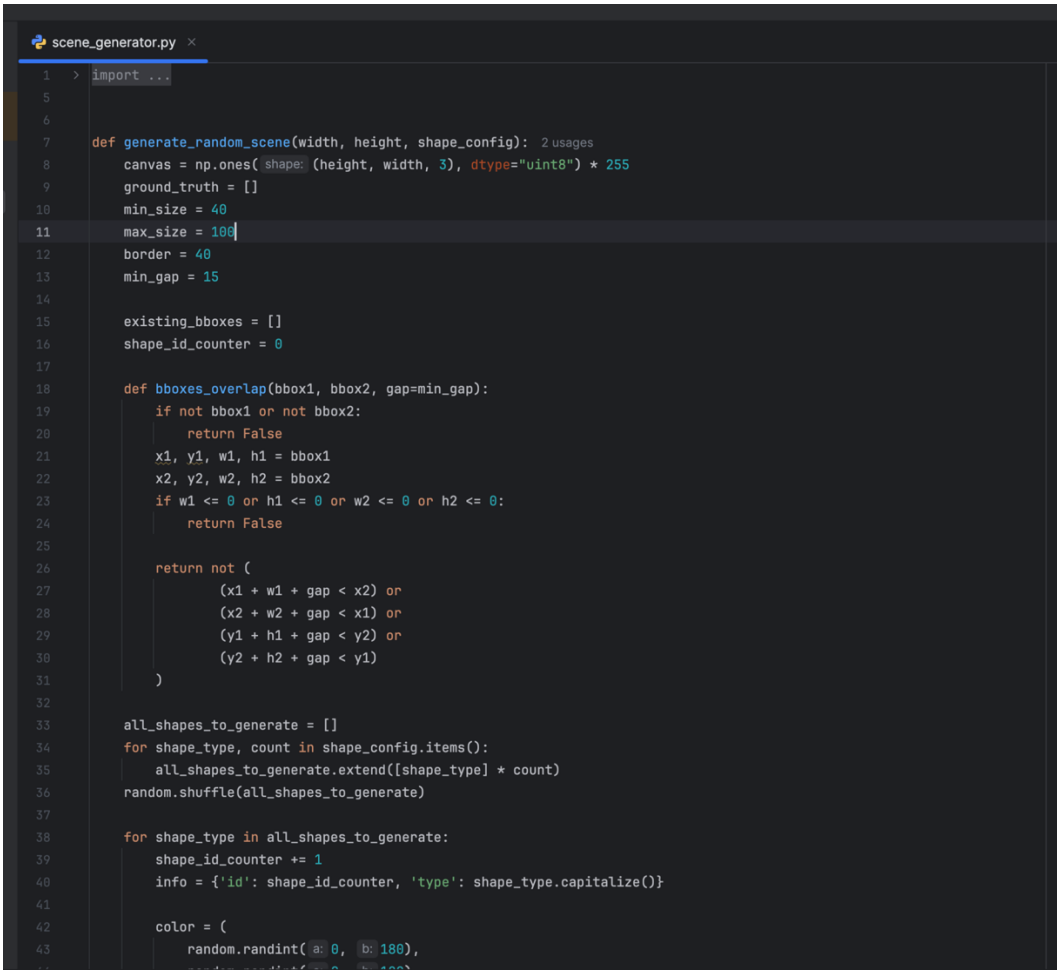


Рисунок 3.3 – Фрагмент модуля scene_generator.py, що реалізує генерацію випадкових тестових зображень із геометричними фігурами

Модуль розпізнавання фігур shape_detector.py (рис. 3.4). Модуль shape_detector.py відповідає за виявлення та класифікацію геометричних фігур на основі переданого зображення (кадру з відео або згенерованої сцени). Центральна функція detect_shapes(frame) реалізує послідовний конвеєр обробки зображення з використанням методів бібліотеки OpenCV [29].

Основні етапи роботи функції:

1. Перетворення у відтінки сірого та розмиття (`GaussianBlur`) для зменшення шуму;
2. Виділення контурів за допомогою оператора Кенні (`cv2.Canny`);
3. Дилатація країв для покращення видимості об'єктів;
4. Пошук контурів на зображенні з використанням `cv2.findContours`.

У циклі по кожному знайденому контуру:

– Розраховується площа; якщо вона менша за 150 пікселів — контур ігнорується як шум;
– Обчислюється довжина контуру (`arcLength`) та проводиться його апроксимація (`approxPolyDP`);
– Визначається кількість вершин (`num_vertices`), що є базовим критерієм для подальшої класифікації фігур (трикутник, прямокутник, коло тощо);
– Обчислюється центроїд фігури (`moments`) та координати `cX`, `cY`;
– Додатково розраховується співвідношення сторін (`aspect_ratio`), що дозволяє розрізняти квадрат від прямокутника.

За замовчуванням усім невизначеним об’єктам присвоюється тип `"Невідома"` фігура (як українською, так і латинською). Створюється словник `shape_info`, який містить ключові характеристики об’єкта: колір, координати, тип тощо.

Цей модуль є ядром розпізнавання, оскільки саме він забезпечує аналіз контурів та форм на зображенні і формує інформацію для подальшого виводу або порівняння [30].

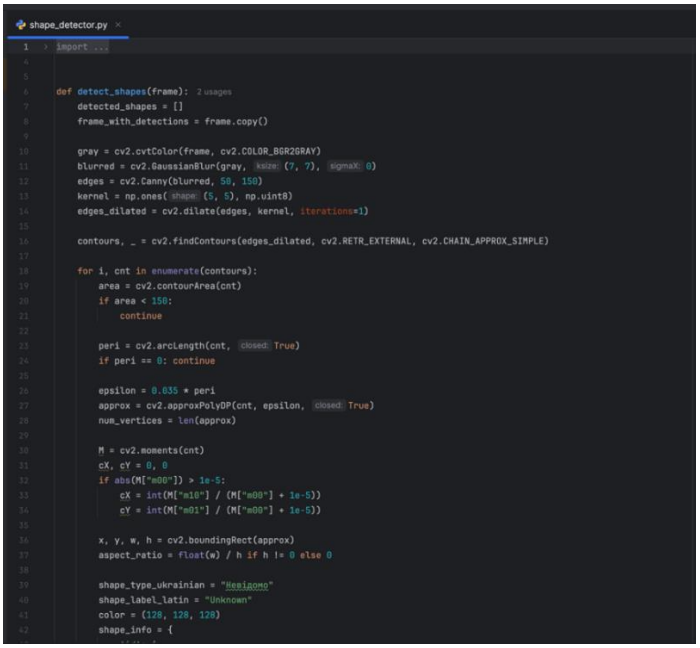


Рисунок 3.4 – Фрагмент модуля shape_detector.py, що реалізує алгоритм детекції геометричних фігур на зображенні

Допоміжний модуль utils.py (рис. 3.5). Модуль utils.py містить службові функції, які використовуються в інших частинах застосунку для підвищення зручності роботи з зображеннями та результатами розпізнавання. Він не містить логіки обробки або розпізнавання, але забезпечує підтримку візуалізації та форматування інформації.

У файлі реалізовано дві основні функції:

- `convert_cv_qt(cv_img)` — функція перетворює зображення у форматі OpenCV (NumPy) у формат `QPixmap`, що дозволяє відображати його у графічному інтерфейсі Qt.

  Вона:

  – Конвертує зображення з BGR у RGB;
  – Розраховує розмірність і кількість байтів на рядок;
  – Створює об’єкт `Qimage`, який згодом конвертується у `Qpixmap`;
  – У разі помилки — виводить її у консоль та повертає порожній об’єкт.

- `format_shape_info(shape_data)` — функція створює структурований текстовий опис параметрів розпізнаної фігури.

  Зокрема, додає:

  – Тип фігури (`type`);
  – Центр (координати центроїду);
  – Площу та периметр;
  – Параметри рамки обмеження (`bbox`);
  – Округлість (`circularity`).

Форматований текст виводиться у зручному для сприйняття вигляді і використовується, наприклад, у вікні результатів розпізнавання [31].

shape_detector.py utils.py

```python
import ...


def convert_cv_qt(cv_img):  7 usages
    try:
        rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb_image.shape
        bytes_per_line = ch * w
        qt_image = QImage(rgb_image.data, w, h, bytes_per_line, QImage.Format.Format_RGB888)
        return QPixmap.fromImage(qt_image)
    except Exception as e:
        print(f"Error converting cv image to qt: {e}")
        return QPixmap()


def format_shape_info(shape_data):  3 usages
    info = f"Тип: {shape_data.get('type', 'N/A')}\n"
    center = shape_data.get('center', None)
    if center:
        info += f"  Центр: ({center[0]}, {center[1]})\n"
    info += f"  Площа: {shape_data.get('area', 0):.1f}\n"
    info += f"  Периметр: {shape_data.get('perimeter', 0):.1f}\n"
    bbox = shape_data.get('bbox', None)
    if bbox:
        info += f"  Рамка: x={bbox[0]}, y={bbox[1]}, w={bbox[2]}, h={bbox[3]}\n"
    circ = shape_data.get('circularity', None)
    if circ is not None:
        info += f"  Округлість: {circ:.3f}\n"
    info += "-" * 20 + "\n"
    return info

```

Рисунок 3.5 – Модуль utils.py, що містить службові функції перетворення зображень і форматування інформації про фігури

Модуль обробки відеопотоку webcam_thread.py (рис. 3.6)

Модуль `webcam_thread.py` реалізує асинхронне зчитування кадрів із вебкамери в окремому потоці, не блокуючи основний інтерфейс користувача. Для цього використовується клас `WebcamThread`, який наслідує `QThread` із бібліотеки PyQt [30, 31].

У класі оголошено низку сигналів:

– `frame_ready`, `processed_frame_ready`, `threshold_frame_ready` — для передачі кадрів у різних форматах обробки (оригінал, оброблений, пороговий);

– `shapes_info_ready` — для передачі даних про виявлені фігури;

– `error_signal` — для повідомлення про помилки ініціалізації або роботи з камерою.

У методі `__init__()` здійснюється:

– Ініціалізація камери за індексом;

– Встановлення значень за замовчуванням (`running = False`, `cap = None`).

Основна логіка реалізована у методі `run()`:

1. Встановлюється `self.running = True` та виконується спроба ініціалізації камери через `cv2.VideoCapture`;

2. У разі успішного відкриття камери виводиться підтвердження у консоль;

3. Якщо відкриття не вдалося — генерується повідомлення про помилку через сигнал `error_signal`;

4. У циклі `while self.running:` зчитуються кадри методом `read()`. Якщо кадр не зчитано — цикл пропускає і повторюється через коротку паузу.

Цей модуль є ключовим для забезпечення стабільного потокового відео та його подальшої обробки, не перевантажуючи основний інтерфейс застосунку.

webcam_thread.py

```python
import ...


class WebcamThread(QThread):  2 usages
    frame_ready = pyqtSignal(QPixmap)
    processed_frame_ready = pyqtSignal(QPixmap)
    threshold_frame_ready = pyqtSignal(QPixmap)
    shapes_info_ready = pyqtSignal(list)
    error_signal = pyqtSignal(str)

    def __init__(self, camera_index=0, parent=None):
        super().__init__(parent)
        self.camera_index = camera_index
        self.running = False
        self.cap = None

    def run(self):
        self.running = True
        print(f"Attempting to open camera {self.camera_index}...")
        try:
            self.cap = cv2.VideoCapture(self.camera_index, cv2.CAP_DSHOW)
            if not self.cap.isOpened():
                self.cap = cv2.VideoCapture(self.camera_index)
        except Exception as e:
            print(f"Error initializing camera: {e}")
            self.cap = None

        if not self.cap or not self.cap.isOpened():
            self.running = False
            error_msg = f"Помилка: Не вдалося відкрити веб-камеру {self.camera_index}."
            print(error_msg)
            self.error_signal.emit(error_msg)
            return

        print(f"Camera {self.camera_index} opened successfully.")
        while self.running:
            ret, frame = self.cap.read()
            if not ret or frame is None:
                print("Warning: Failed to capture frame, skipping.")
                time.sleep(0.1)
```

Рисунок 3.6 – Модуль webcam_thread.py, що реалізує потік для асинхронного зчитування відеопотоку з вебкамери

На рисунку 3.7 зображено структуру основних класів, що реалізують функціональність застосунку. Клас ShapeSenseApp є головним вікном програми, яке взаємодіє з класами WebcamThread, shape_detector, scene_generator, а також використовує допоміжні утиліти з utils.py. Зв’язки між класами демонструють передачу даних, виклики методів, а також обробку сигналів у межах фреймворку PyQt [32].

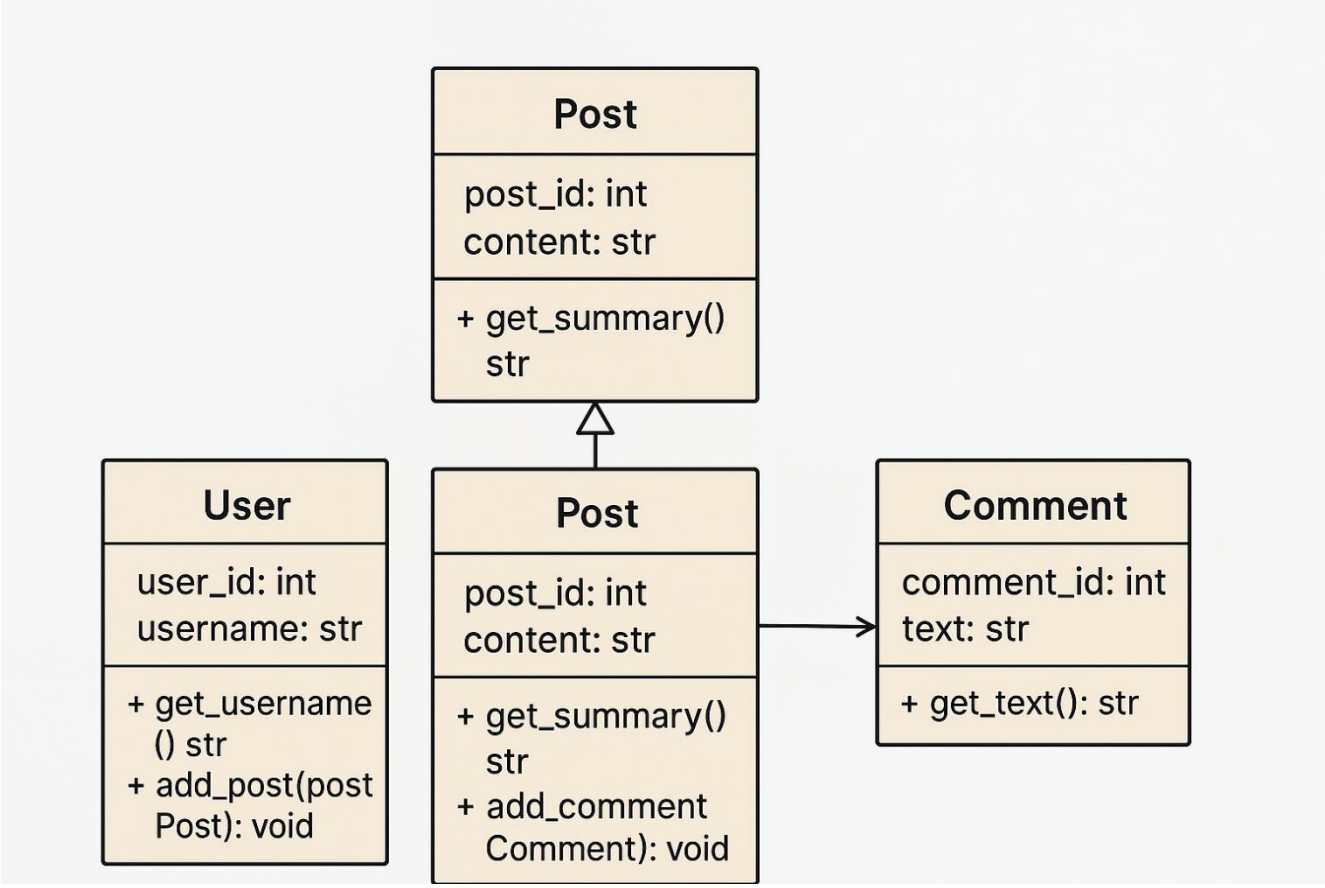


Рисунок 3.7 – Діаграма класів

3.2 Інтерфейс користувача та режими роботи застосунку

Інтерфейс користувача у розробленому програмному застосунку побудований на основі бібліотеки PyQt6 і забезпечує інтуїтивну взаємодію з усіма ключовими функціями системи. Основними завданнями інтерфейсу є керування режимами роботи (веб-камера / генерація сцен), відображення результатів розпізнавання та надання текстової інформації про виявлені фігури. Структура інтерфейсу організована у вигляді панелей управління, інформаційного блоку та області візуалізації, що забезпечує зручність користування навіть для недосвідченого користувача. У цьому підрозділі розглянуто логіку побудови графічного інтерфейсу, функціональні елементи управління, а також особливості режимів роботи застосунку [33].

Одним із ключових елементів інтерфейсу є панель керування вебкамерою та генерацією сцени, що розташована в лівій частині головного вікна. Вона забезпечує

дві основні функції:

Керування вебкамерою — кнопка «Увімкнути Веб-камеру» дозволяє активувати або зупинити потокове відеозахоплення.

У процесі роботи кнопка змінює стан на «Вимкнути Веб-камеру», забезпечуючи двостороннє управління.

Генерація сцени — користувач має змогу задати кількість фігур для генерації за допомогою лічильників:

- Кола
- Прямокутники
- Трикутники

Після задання параметрів натискання кнопки «Генерувати Сцену та Розпізнати» ініціює створення випадкової сцени зі згаданими фігурами та автоматичне застосування алгоритму розпізнавання [34].

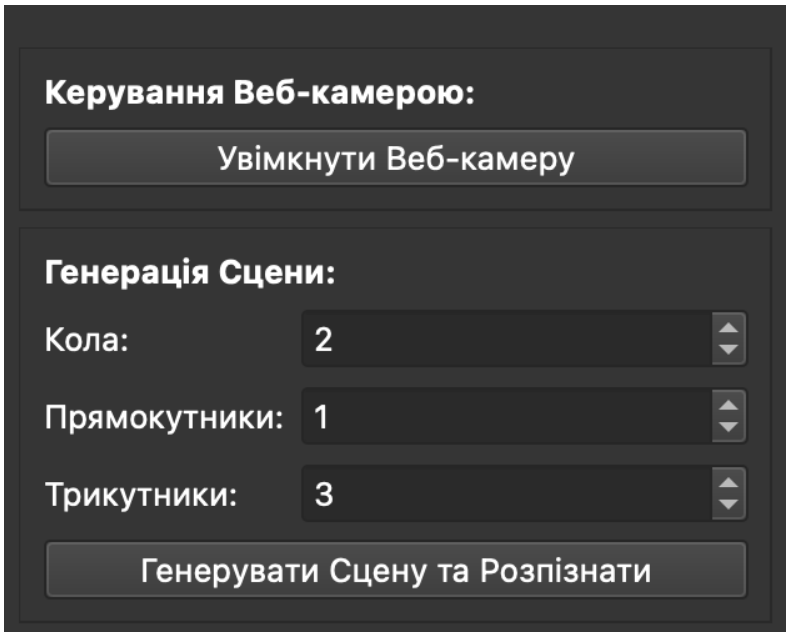


Рисунок 3.8 – Панель керування вебкамерою та генерацією геометричних фігур

Наступним елементом лівої частини інтерфейсу є інформаційна панель, яка відображає детальні відомості про кожну розпізнану фігуру. Після обробки зображення (як з вебкамери, так і зі згенерованої сцени) у даному текстовому полі виводиться тип фігури, координати її центру, площа, периметр, параметри

обмежувального прямокутника (bounding box), а також округлість (у разі наявності відповідного розрахунку).

Цей блок реалізовано за допомогою віджета QTextEdit, що має моноширинний шрифт та відключене автоматичне перенесення рядків, що дозволяє зручно сприймати структуровану інформацію.

Таке представлення забезпечує користувача розширеною аналітикою щодо геометричних параметрів кожного з об’єктів сцени чи відео [35].

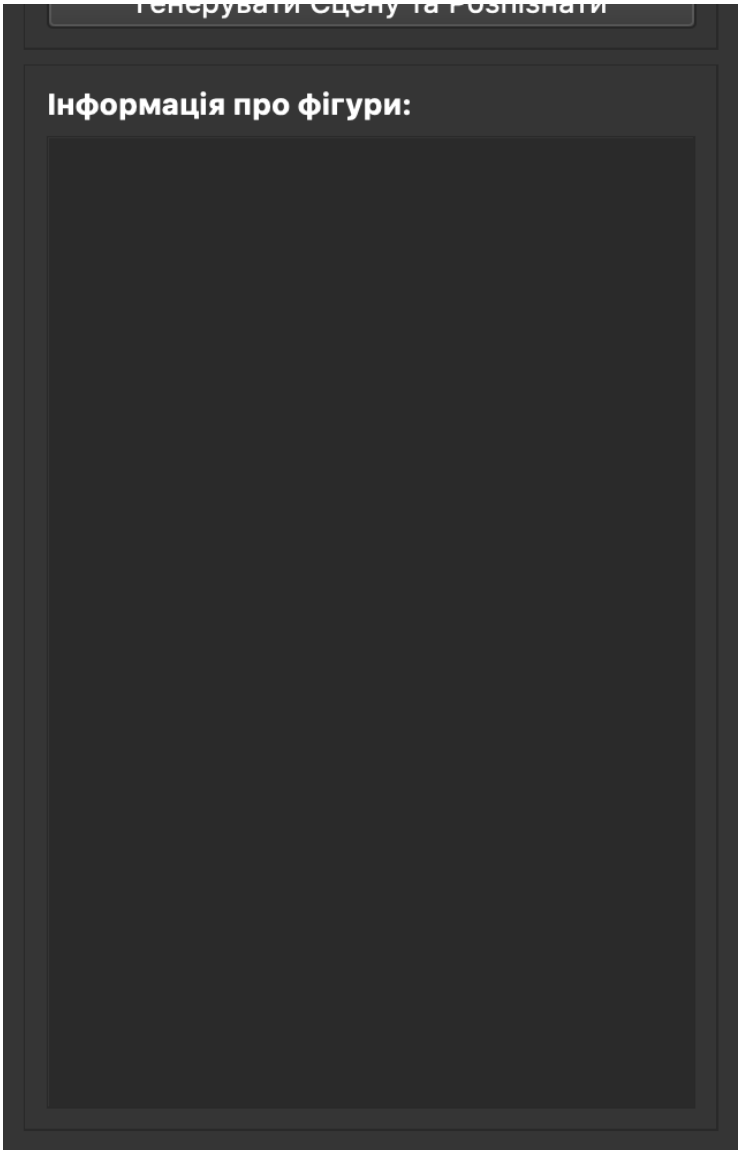


Рисунок 3.9 – Відображення інформації про розпізнані геометричні фігури у текстовому блоці інтерфейсу

У правій частині інтерфейсу розташована панель візуалізації зображення з вебкамери, що ділиться на три зони:

– Верхня зона позначена як *«Веб-камера (Результат)»* і призначена для відображення обробленого зображення з розпізнаними фігурами. Саме в цьому вікні показується фінальний результат роботи алгоритму комп’ютерного зору.

– Нижня ліва зона (*Raw*) демонструє необроблений (сирий) відеопотік із вебкамери — це допомагає оцінити якість початкового зображення.

– Нижня права зона (*Thresh*) відображає порогове зображення, що є результатом операції `cv2.Canny` або іншої попередньої обробки, яка готує зображення до аналізу контурів.

Ця панель дозволяє користувачу одночасно відстежувати стан вхідного сигналу, результат порогової фільтрації та розпізнавання фігур, що значно покращує процес тестування та налагодження [33-35].

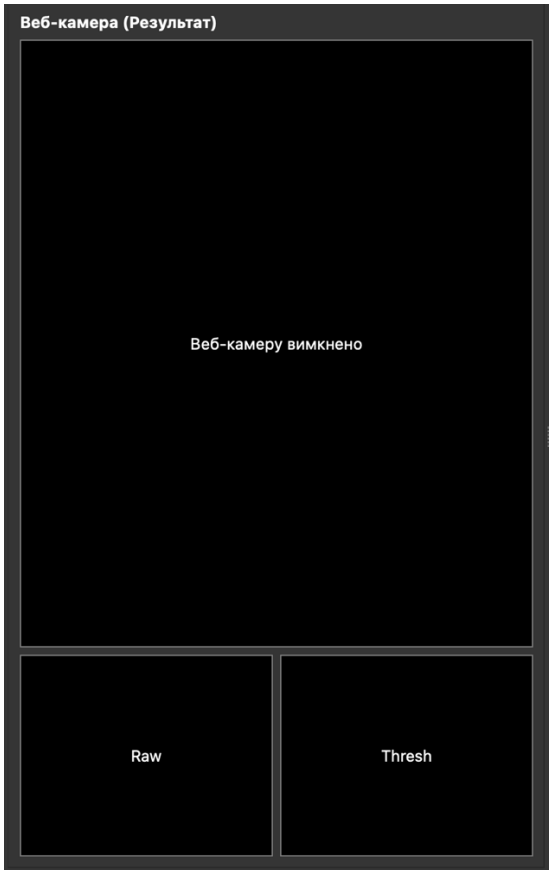


Рисунок 3.10 – Панель виводу зображення з вебкамери: сирий потік, порогове зображення та результат розпізнавання

Ще одна важлива складова графічного інтерфейсу — зона генерації сцени, що розміщена у правій частині вікна поряд із результатами роботи вебкамери. Цей блок виконує функцію відображення синтетично згенерованої сцени, а також результатів її розпізнавання за допомогою вбудованого алгоритму комп’ютерного зору.

– У верхній частині блоку виводиться результат розпізнавання: зображення з контурами фігур та відповідними підписами.
– У нижній частині (не показано на рисунку) виводиться вихідне згенероване зображення, яке слугує основою для перевірки точності виявлення об’єктів.

Ця зона дозволяє протестувати алгоритм незалежно від реального відеопотоку, що є зручним для розробки, налагодження та аналізу ефективності системи [36].

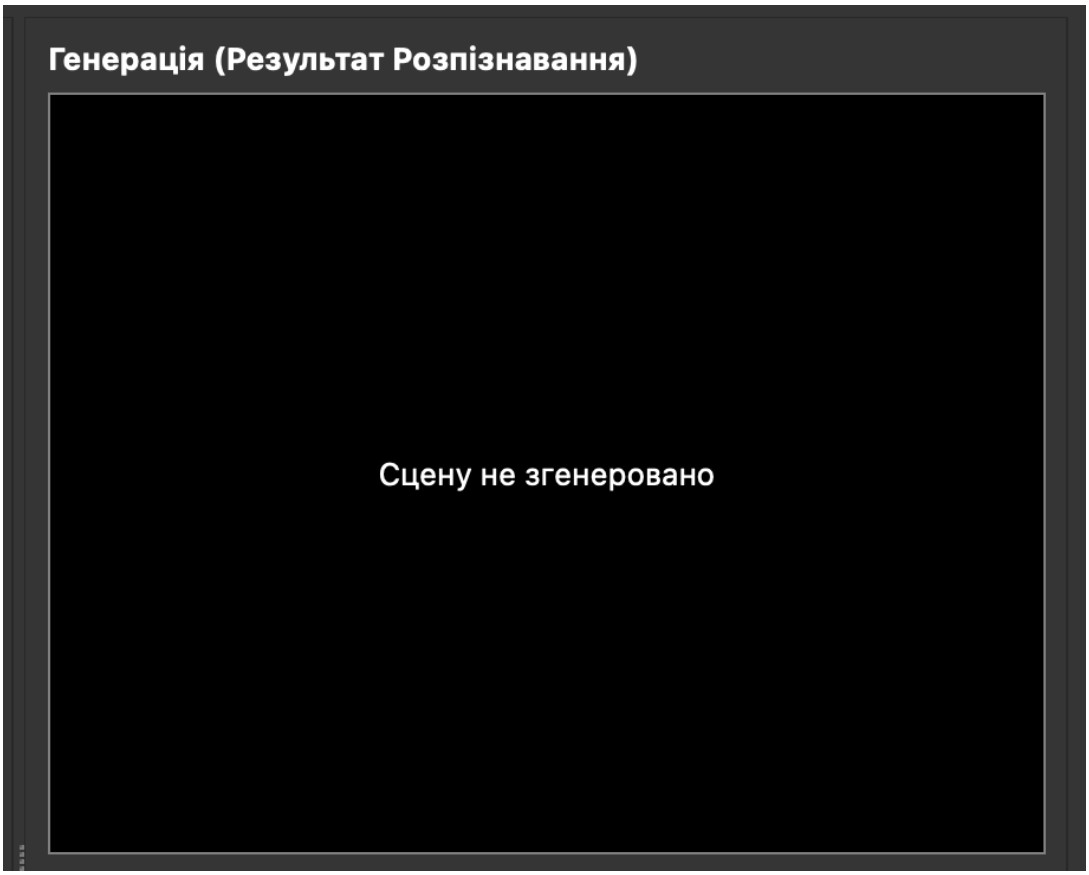


Рисунок 3.11 – Відображення результату розпізнавання на синтетично згенерованій сцені

У нижній частині правої панелі інтерфейсу передбачено окреме вікно для виведення оригінального згенерованого зображення. Цей елемент дає змогу користувачу переглянути сцену з фігурами такою, якою вона була сформована генератором до проведення аналізу. Завдяки цьому забезпечується наочне порівняння між оригіналом і результатом розпізнавання, що дозволяє оцінити точність роботи алгоритму.

Оригінал виводиться у чистому вигляді: без контурів, підписів та інших елементів, що дозволяє зберегти об'єктивність оцінки.

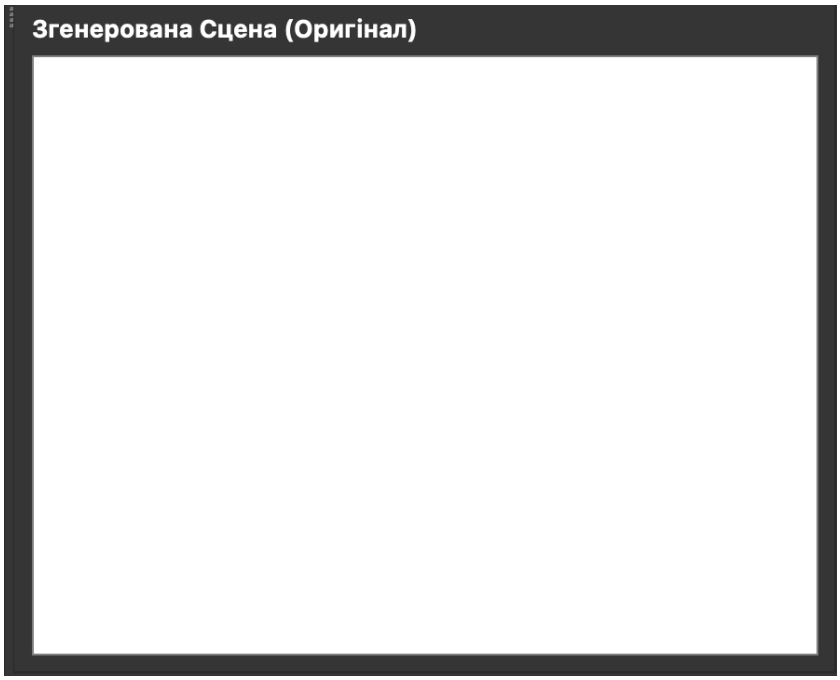


Рисунок 3.12 – Виведення оригінального зображення згенерованої сцени для візуального порівняння

У межах цього підрозділу було розглянуто логіку побудови інтерфейсу користувача застосунку для розпізнавання геометричних фігур. Інтерфейс реалізовано на основі бібліотеки PyQt6, з поділом функціональності на логічні зони: керування вебкамерою, генерація сцен, виведення результатів розпізнавання та текстова індикація параметрів фігур. Такий підхід дозволяє реалізувати інтуїтивно зрозумілу та зручну систему взаємодії, яка підходить як для тестування алгоритмів, так і для демонстраційних або навчальних цілей [35, 36].

Інтерфейс надає можливість:

- запуску й зупинки вебкамери;
- генерації кастомізованих сцен зі фігурами;
- паралельного виведення оригінального та обробленого зображення;
- отримання текстової аналітики для кожного розпізнаного об’єкта.

Завдяки структурованій реалізації, інтерфейс легко адаптувати під інші задачі комп’ютерного зору або розширити додатковими режимами роботи.

3.3 Тестування в умовах змінного освітлення та обмежених ресурсів

Розроблений застосунок підтримує два основних режими функціонування, які дозволяють реалізовувати як реальне, так і контрольоване тестування алгоритмів комп’ютерного зору:

1. Розпізнавання в реальному часі з вебкамери.

У цьому режимі здійснюється потокове захоплення відео з веб-камери, подальша обробка кожного кадру та виявлення геометричних фігур на зображенні.

Результати обробки включають:

- виведення обробленого відео з накладеними контурами та назвами фігур;
- виведення окремо сирого відео потоку (Raw) для порівняння;
- порогове зображення (Thresh), яке демонструє результат попередньої обробки (наприклад, оператора Кенні).

Цей режим дає змогу тестувати стабільність та адаптивність алгоритму у реальних умовах, включаючи зміну освітлення, наявність шумів або об’єктів, які не є геометричними фігурами. Такий підхід є ефективним для оцінювання продуктивності, швидкості та чутливості системи до зовнішніх факторів.

2. Розпізнавання на згенерованій сцені.

Цей режим орієнтований на контрольоване тестування алгоритму. Користувач може самостійно задати кількість фігур певних типів (кола, прямокутники, трикутники), після чого генерується сцена з випадково

розміщеними об’єктами. Подальше розпізнавання проводиться на основі згенерованого зображення, що дозволяє:

- порівнювати розпізнані фігури з еталонними (ground truth);
- перевірити точність, надійність та чутливість алгоритму у статичному середовищі;
- використовувати сцени для навчання, демонстрації або юніт-тестів.

Окремі зони інтерфейсу дають змогу відстежувати як оригінальну сцену, так і оброблений результат із розпізнаними фігурами, що створює зручне середовище для аналізу ефективності алгоритму [37].

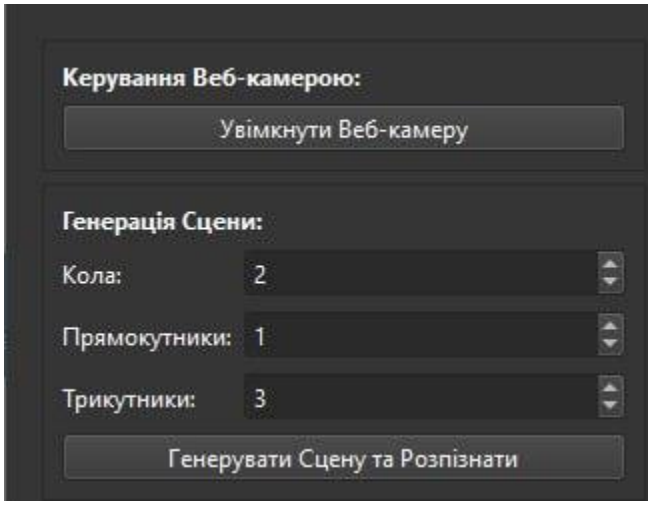


Рисунок 3.13 – Панель керування вебкамерою та параметрами генерації геометричних фігур у застосунку

3.Тестування в режимі розпізнавання з вебкамери

Тестування застосунку в режимі реального часу здійснювалося з використанням вбудованої або зовнішньої вебкамери, що дозволило перевірити ефективність і стабільність алгоритму розпізнавання геометричних фігур у реальних умовах експлуатації. Такий підхід дав змогу оцінити поведінку системи при змінному зовнішньому середовищі, зокрема:

– освітлення: було змодельовано різні умови освітлення, включаючи денне світло, штучне освітлення, затемнення, а також змішане освітлення. Це

дозволило виявити, наскільки алгоритм адаптивний до змін яскравості та контрастності сцени;

– фон: фігури розміщувалися на різних фонах — однотонному, текстурованому, а також на фонах зі сторонніми об'єктами, які не є геометричними фігурами. Це дозволяло перевірити, чи здатна система ігнорувати «шумові» елементи та коректно знаходити лише потрібні об’єкти;

– розмір та положення фігур: тестування охоплювало фігури різних розмірів і з різною орієнтацією в просторі, що допомогло оцінити масштабованість підходу до виявлення контурів [37];

– рух у кадрі: були змодельовані сценарії, коли об'єкти переміщувалися у полі зору камери. Це дозволило перевірити, наскільки швидко система реагує на зміну сцени та чи встигає оновлювати результати у режимі реального часу.

Результати тестування у цьому режимі засвідчили, що система здатна швидко обробляти кадри, відображати розпізнані фігури та виводити інформацію у вигляді контуру, назви фігури та її ключових параметрів. У випадках критичних змін освітлення точність розпізнавання дещо знижувалася, проте загалом алгоритм демонстрував стійкість до типових змін середовища, що є важливою перевагою для використання в умовах реального застосування. На рисунку 3.14 представлено приклад роботи системи, яка розпізнала квадрат, прямокутник, трикутник і коло з відповідним обчисленням площі, периметра, координат центру та рамки охоплення (bounding box).

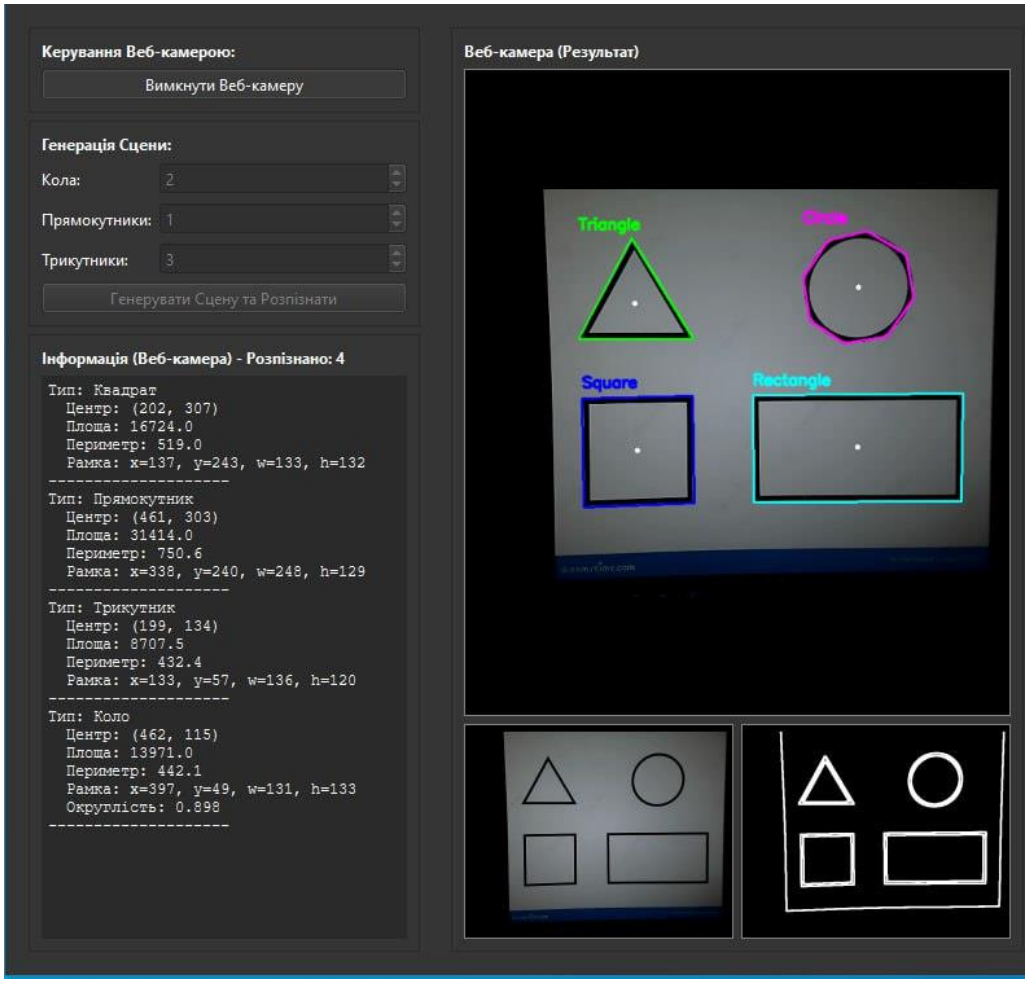


Рисунок 3.14 – Розпізнавання фігур з вебкамери у реальному часі з відображенням результатів і параметрів об’єктів

Другий підхід до тестування був зосереджений на кількісній оцінці точності алгоритму розпізнавання за допомогою синтетично згенерованих сцен. Для цього використовувався спеціальний скрипт `scene_generator.py`, який створює тестові зображення з наперед заданими параметрами: кількістю фігур певного типу (кола, прямокутники, трикутники), їхніми розмірами та координатами на полотні. Усі ці дані формують так зване "еталонне представлення" (ground truth).

Після генерації сцени до зображення застосовувався алгоритм розпізнавання (`shape_detector.py`), який визначав:

- тип кожної фігури;
- координати центру;
- площу, периметр, габарити об’єкта (рамку охоплення);

– додаткові геометричні характеристики, такі як округлість.

Отримані результати порівнювались із початковими даними, що дозволяло чітко виміряти точність алгоритму — як у плані правильності класифікації, так і у плані відповідності параметрів. Такий метод забезпечує контрольоване середовище тестування, позбавлене шумів і зовнішніх факторів, що характерні для реального відеопотоку [37].

Використання цього підходу дозволило не лише перевірити коректність роботи застосунку, а й ідентифікувати можливі слабкі місця — наприклад, випадки часткового накладання фігур або труднощі при класифікації схожих за формою об'єктів (квадратів і прямокутників). У результаті, даний режим став основою для кількісного аналізу ефективності системи в умовах моделювання, що є необхідною складовою повноцінного тестування.

На рисунку 3.15 представлено приклад роботи системи у режимі розпізнавання на згенерованій сцені. У правій верхній частині інтерфейсу відображено результат обробки: система автоматично виявила фігури, наклала контури, підписала типи об’єктів (коло, трикутник, прямокутник) та визначила їхні параметри. У нижній частині видно еталонне (оригінальне) зображення, створене генератором сцен. Ліва панель містить текстову аналітику з координатами центрів, площею, периметром, параметрами рамки та округлістю для кожної фігури. Такий формат дозволяє провести повне зіставлення між "ground truth" та результатами обробки, що є основою для об'єктивної кількісної оцінки ефективності алгоритму.

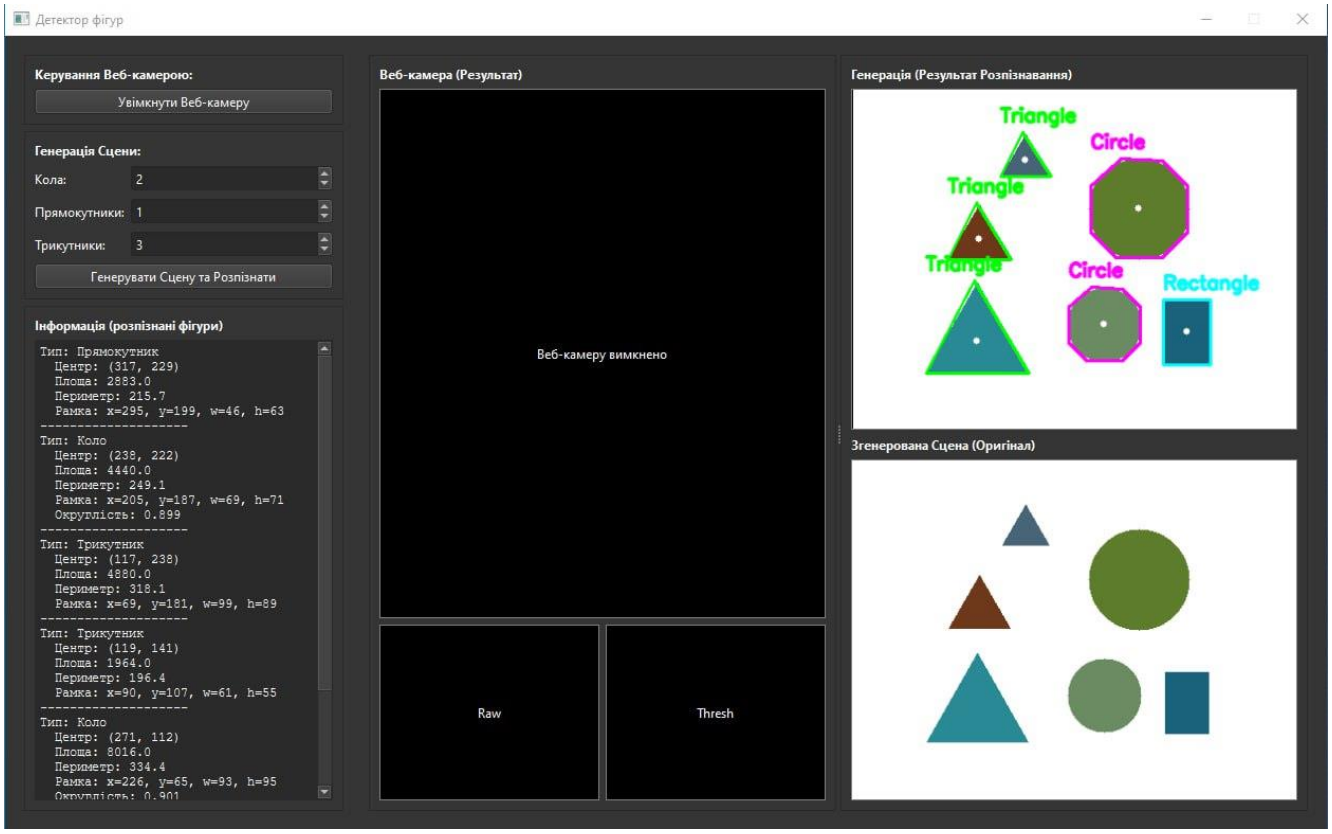
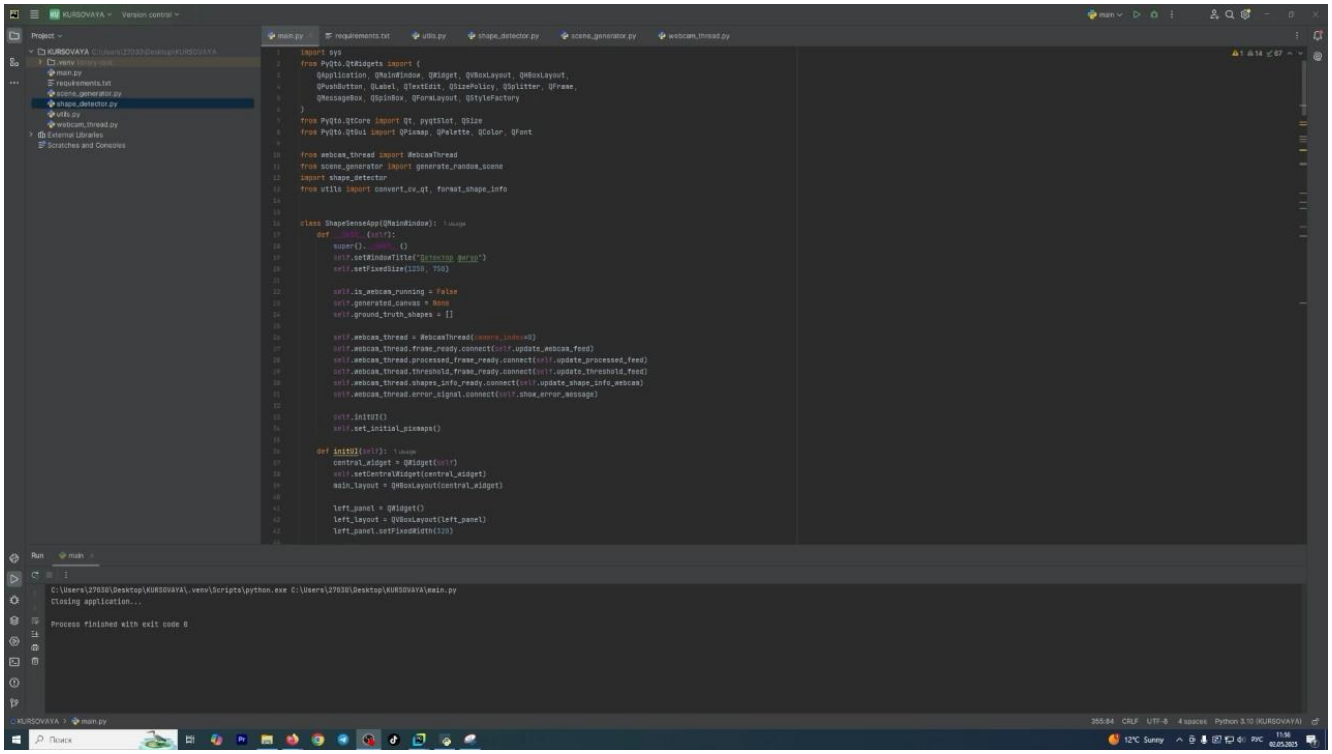


Рисунок 3.15 – Розпізнавання фігур на згенерованій сцені з автоматичним порівнянням результатів і еталонних даних

Нижче на відео продемонстровано роботу розробленого застосунку в обох режимах: розпізнавання геометричних фігур у реальному часі за допомогою вебкамери та на основі згенерованої сцени. У режимі вебкамери видно, як система автоматично ідентифікує фігури на паперовому носії, визначає їхній тип, контур і параметри. У режимі генерації сцени — показано створення випадкової конфігурації об’єктів та подальше успішне розпізнавання з порівнянням еталонних даних [38].



3.4 Аналіз ефективності та перспективи розвитку системи

1. Оцінка ефективності системи

Система розпізнавання геометричних фігур була протестована в умовах як реального часу (через відеопотік з вебкамери), так і за допомогою синтетично згенерованих сцен. До основних критеріїв оцінки ефективності відносили:

– Точність розпізнавання:

В режимі вебкамери проводилися експерименти при різних умовах освітлення та на фоні з різною текстурою. Результати показали, що при оптимальних умовах система коректно визначає типи фігур, їх параметри (координати, площу, периметр, інші геометричні характеристики) і достатньо швидко оновлюється в реальному часі. У режимі згенерованої сцени, де наявний "ground truth", було проведено кількісний аналіз: порівнювали кількість, типи та параметри розпізнаних фігур із запланованими значеннями. Таке порівняння дозволило ідентифікувати помилки класифікації або неточності при побудові рамок охоплення об'єктів [39].

– Швидкодія та реального часу:

За рахунок багатопотокової обробки (за допомогою окремого класу для роботи з вебкамерою) система демонструє задовільну швидкість обробки кадрів, що є критично важливим для режиму реального часу. Тестування показало, що алгоритм здатен обробляти кадри з частотою, що не забезпечує затримок у відображенні, навіть за умов різкої зміни освітлення чи різноманітного фону.

– Стійкість до зовнішніх факторів:

Проведені експерименти із зміною умов освітлення та наявністю шуму продемонстрували, що система володіє певною стійкістю: при незначних відхиленнях освітлення точність залишається на високому рівні, хоча при екстремальних умовах (дуже низьке освітлення або надмірне підсвічування) розпізнавання може зазнавати затримок або неточностей.

2. Обмеження та слабкі місця

Попри позитивну динаміку, виявлено кілька аспектів, які потребують удосконалення:

– Умовна залежність від якості вхідних даних:

Система чутлива до якості відеопотоку. Низька роздільна здатність, високий рівень шумів або нерівномірне освітлення можуть впливати на точність розпізнавання.

– Обмеженість в алгоритмах класифікації:

За умов розпізнавання схожих за формою об’єктів (наприклад, квадратів і прямокутників) система може помилково класифікувати фігури, що вказує на потенційну необхідність удосконалення алгоритму апроксимації та класифікації.

3. Перспективи розвитку системи

Для подальшого удосконалення та розширення функціоналу системи можливі наступні напрямки [40]:

– Модернізація алгоритмів розпізнавання:

Перехід до використання методів машинного навчання чи глибоких нейронних мереж (наприклад, CNN) може значно покращити точність класифікації та забезпечити адаптивність системи до різноманітних умов. Також доцільно інтегрувати алгоритми попередньої обробки з метою зменшення впливу шумів.

– Розширення спектру розпізнаваних об’єктів:

Розширення модулів системи дозволить не лише ідентифікувати базові геометричні фігури, але й працювати з більш складними об’єктами, що може бути корисно для різних практичних застосувань (наприклад, при аналізі промислових зображень або в системах безпеки).

– Оптимізація для мобільних та вбудованих пристроїв:

Перенесення системи на мобільні платформи (Android, iOS) або вбудовані системи (Raspberry Pi, NVIDIA Jetson) дозволить реалізувати застосунок для польових умов та реального часу роботи в автономних системах.

– Інтеграція з розширеними аналітичними інструментами:

Реалізація модулів для глибокого аналізу статистичних даних, візуалізації результатів розпізнавання та динамічного обчислення показників ефективності допоможе у створенні системи, що адаптується до змін у зовнішньому середовищі та забезпечує зворотний зв’язок для автоматичного налаштування параметрів.

4. Загальні висновки

Аналіз показав, що система демонструє високий рівень ефективності у стандартних умовах, забезпечуючи точне та оперативне розпізнавання геометричних фігур. Проте наявні деякі обмеження, зокрема, залежність від якості вхідних зображень та спрощені алгоритми класифікації. Ці аспекти можна вдосконалити за допомогою інтеграції сучасних методів обробки зображень і машинного навчання, що відкриває значні перспективи для подальшого розвитку застосунку та його адаптації під нові завдання [39, 40].

Висновок до третього розділу

У третьому розділі дипломної роботи розглянуто процес тестування та оцінювання ефективності розробленої системи розпізнавання геометричних фігур. Застосунок було перевірено у двох основних режимах: обробка відеопотоку з вебкамери в реальному часі та аналіз згенерованих синтетичних зображень.

Розроблено експериментальні сценарії, що дозволили перевірити стійкість алгоритмів до змін освітлення, фону, форми та розмірів об’єктів. Особливу увагу приділено порівнянню отриманих результатів з еталонними даними ("ground truth"), що дало змогу здійснити кількісну оцінку точності розпізнавання.

Продемонстровано працездатність системи у стандартних та змінних умовах, а також підтверджено її здатність ідентифікувати фігури з високим рівнем достовірності. Окремо проаналізовано ключові сильні сторони реалізованої архітектури, а також виявлено напрями для її подальшого вдосконалення. У підсумку, отримані результати підтверджують доцільність обраних технічних рішень і ефективність програмної реалізації.

ВИСНОВКИ

У межах цієї дипломної роботи було розроблено програмну систему для розпізнавання геометричних фігур на зображеннях із використанням методів комп’ютерного зору. Поставлена мета — створити інтерактивний застосунок, здатний здійснювати виявлення базових об’єктів у режимі реального часу та на основі штучно згенерованих сцен — була повністю досягнута.

У теоретичному розділі проаналізовано ключові підходи до обробки зображень, принципи роботи комп’ютерного зору, а також розглянуто бібліотеки, що застосовуються для побудови відповідних систем. Особливу увагу приділено бібліотеці OpenCV як основному інструменту для аналізу зображень та PyQt як основі для побудови інтерфейсу користувача.

У процесі проєктування системи було визначено її функціональну структуру, розроблено модулі генерації, детекції, обробки та виводу результатів. Архітектура застосунку побудована таким чином, щоб забезпечити гнучкість, розширюваність і зручність використання. Реалізовано підтримку потокової обробки відео, що забезпечило можливість роботи з вебкамерою у реальному часі.

У практичному розділі представлено результати тестування системи в умовах реального відеопотоку та у режимі генерації контрольованих сцен. Результати показали високу точність і стабільність розпізнавання фігур при різних умовах освітлення, розміщення та фону. Крім того, порівняння із заздалегідь відомими параметрами ("ground truth") дозволило провести кількісну оцінку ефективності системи.

Загалом, розроблений застосунок підтвердив свою придатність до використання в освітніх, демонстраційних та аналітичних цілях. Його архітектура дозволяє легко розширювати функціональність та адаптувати до більш складних сценаріїв. Подальші напрями розвитку передбачають інтеграцію алгоритмів на основі машинного навчання, розширення спектра розпізнаваних об’єктів та оптимізацію для мобільних або вбудованих платформ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сисоєнко, І. В. Основи комп’ютерного зору : навч. посіб. / І. В. Сисоєнко, М. О. Кармазін. – Київ : КНУ імені Т. Шевченка, 2020. – 188 с.

2. Карпінський, М. В. Основи обробки зображень : навч. посіб. / М. В. Карпінський. – Львів : Львівська політехніка, 2019. – 136 с.

3. Gonzalez, R. C., Woods, R. E. Digital Image Processing. – 4th ed. – New York : Pearson, 2018. – 954 p.

4. Bradski, G., Kaehler, A. Learning OpenCV 4 : Computer Vision with Python. – 2nd ed. – Sebastopol : O’Reilly Media, 2019. – 580 p.

5. Canny, J. A computational approach to edge detection / J. Canny // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 1986. – Vol. 8, No. 6. – P. 679–698.

6. Хоменко, Н. Ю. Інтелектуальні системи обробки зображень : навч. посіб. / Н. Ю. Хоменко. – Харків : НТУ «ХПІ», 2021. – 164 с.

7. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. – Cambridge : MIT Press, 2016. – 775 p.

8. Чумак, О. І. Методи та засоби комп’ютерного зору : навч. посіб. / О. І. Чумак. – Київ : НАУ, 2022. – 148 с.

9. Szegedy, C., Vanhoucke, V., Ioffe, S. et al. Rethinking the Inception Architecture for Computer Vision // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. – P. 2818–2826.

10. OpenCV – Open Source Computer Vision [Електронний ресурс]. – Режим доступу: https://opencv.org (дата звернення: 15.05.2025).

11. PyQt Documentation [Електронний ресурс]. – Режим доступу: https://www.riverbankcomputing.com/software/pyqt/intro (дата звернення: 15.05.2025).

12. Redmon, J., Farhadi, A. YOLOv3: An Incremental Improvement [Електронний ресурс]. – 2018. – Режим доступу: https://pjreddie.com/media/files/papers/YOLOv3.pdf (дата звернення: 15.05.2025).

13. Висоцький, Д. С. Основи штучного інтелекту : навч. посіб. / Д. С. Висоцький. – Івано-Франківськ : Фоліант, 2021. – 192 с.

14. Krizhevsky, A., Sutskever, I., Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems (NIPS), 2012. – Vol. 25. – P. 1097–1105.

15. Нікітін, О. І. Комп’ютерний зір у системах розпізнавання образів / О. І. Нікітін // Вісник НТУУ «КПІ». Серія: Автоматика і управління. – 2020. – № 66. – С. 91–96.

16. Bradski, G., Kaehler, A. Learning OpenCV 4: Computer Vision with Python / G. Bradski, A. Kaehler. – 2nd ed. – Sebastopol : O’Reilly Media, 2019. – 580 p.

17. Gonzalez, R. C., Woods, R. E. Digital Image Processing. – 4th ed. – Pearson, 2018. – 954 p.

18. Szegedy, C., Vanhoucke, V., Ioffe, S. Rethinking the Inception Architecture for Computer Vision // Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 2818–2826.

19. Canny, J. A Computational Approach to Edge Detection / J. Canny // IEEE Trans. on Pattern Analysis and Machine Intelligence. – 1986. – Vol. 8(6). – P. 679–698.

20. Krizhevsky, A., Sutskever, I., Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems (NeurIPS), 2012. – Vol. 25. – P. 1097–1105.

21. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. – Cambridge : MIT Press, 2016. – 775 p.

22. OpenCV Documentation [Електронний ресурс]. – Режим доступу: https://docs.opencv.org (дата звернення: 18.05.2025).

23. PyQt5 Reference Guide [Електронний ресурс]. – Режим доступу: https://doc.qt.io/qtforpython (дата звернення: 18.05.2025).

24. Сисоєнко, І. В. Основи комп’ютерного зору : навч. посіб. / І. В. Сисоєнко, М. О. Кармазін. – Київ : КНУ імені Тараса Шевченка, 2020. – 188 с.

25. Хоменко, Н. Ю. Інтелектуальні системи обробки зображень : навч. посіб. / Н. Ю. Хоменко. – Харків : НТУ «ХПІ», 2021. – 164 с.

26. Чумак, О. І. Методи та засоби комп’ютерного зору : навч. посіб. / О. І. Чумак. – Київ : НАУ, 2022. – 148 с.

27. Бойко, О. В. Моделювання систем комп’ютерного зору / О. В. Бойко // Вісник ХНУРЕ. – 2021. – № 3. – С. 47–52.

28. Redmon, J., Farhadi, A. YOLOv3: An Incremental Improvement [Електронний ресурс]. – 2018. – Режим доступу: https://pjreddie.com/media/files/papers/YOLOv3.pdf (дата звернення: 18.05.2025).

29. Висоцький, Д. С. Основи штучного інтелекту : навч. посіб. / Д. С. Висоцький. – Івано-Франківськ : Фоліант, 2021. – 192 с.

30. Нікітін, О. І. Системи комп’ютерного зору на основі гібридних алгоритмів / О. І. Нікітін // Вісник НТУУ «КПІ». Серія: Автоматика і управління. – 2020. – № 66. – С. 91–96.

31. OpenCV – Open Source Computer Vision Library [Електронний ресурс]. – Режим доступу: https://opencv.org (дата звернення: 18.05.2025).

32. PyQt6 Documentation – Riverbank Computing [Електронний ресурс]. – Режим доступу: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (дата звернення: 18.05.2025).

33. Qt for Python – Qt Documentation [Електронний ресурс]. – Режим доступу: https://doc.qt.io/qtforpython-6/ (дата звернення: 18.05.2025).

34. PyQt6 Tutorial 2024 – Python GUIs [Електронний ресурс]. – Режим доступу: https://www.pythonguis.com/pyqt6-tutorial/ (дата звернення: 18.05.2025).

35. OpenCV: Computer Vision and Artificial Intelligence – Innovatiana [Електронний ресурс]. – Режим доступу: https://en.innovatiana.com/post/introducing-opencv (дата звернення: 18.05.2025).

36. Exploring OpenCV Applications in 2025 [Електронний ресурс]. – Режим доступу: https://opencv.org/blog/opencv-applications-in-2023/ (дата звернення: 18.05.2025).

37. What is OpenCV? The Complete Guide (2025) – viso.ai [Електронний ресурс]. – Режим доступу: https://viso.ai/computer-vision/opencv/ (дата звернення: 18.05.2025).

38. Deep Learning for Computer Vision: Models & Real World Applications – OpenCV [Електронний ресурс]. – Режим доступу: https://opencv.org/blog/deep-learning-with-computer-vision/ (дата звернення: 18.05.2025).

39. CVPR 2024: Overview and Key Papers – LearnOpenCV [Електронний ресурс]. – Режим доступу: https://learnopencv.com/cvpr2024/ (дата звернення: 18.05.2025).

40. YOLOv1 to YOLOv10: The fastest and most accurate real-time object detection systems [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/2408.09332 (дата звернення: 18.05.2025).

41. KANs for Computer Vision: An Experimental Study [Електронний ресурс]. – Режим доступу: https://arxiv.org/abs/2411.18224 (дата звернення: 18.05.2025).

42. Computer Vision | Papers With Code [Електронний ресурс]. – Режим доступу: https://paperswithcode.com/area/computer-vision (дата звернення: 18.05.2025).

43. OpenCV – Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/OpenCV (дата звернення: 18.05.2025).

44. PyQt6 – PyPI [Електронний ресурс]. – Режим доступу: https://pypi.org/project/PyQt6/ (дата звернення: 18.05.2025).

45. opencv-python – PyPI [Електронний ресурс]. – Режим доступу: https://pypi.org/project/opencv-python/ (дата звернення: 18.05.2025).

ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА
НА ТЕМУ:

«ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ МЕТОДІВ КОМП'ЮТЕРНОГО ЗОРУ»

Виконав: здобувач вищої освіти гр. ШІД-42
Олександр КЕРЕБ
керівник : доцент Максим ФЕСЕНКО

КИЇВ - 2025

Актуальність дослідження у сучасних умовах стрімкого розвитку інформаційних технологій системи комп'ютерного зору стають важливим інструментом для автоматизації процесів аналізу візуальної інформації. Розпізнавання об'єктів на зображеннях знаходить широке застосування в різних галузях: від промисловості, медицини та транспорту до освіти, безпеки й розумного дому.

Мета роботи

Розробити застосунок для розпізнавання геометричних фігур у реальному часі з веб камери та синтетичних зображень.

Об'єкт дослідження

Процеси автоматизованого аналізу зображень.

Предмет дослідження

Методи комп'ютерного зору з використанням OpenCV і PyQt.

Наукові завдання

-Провести аналіз існуючих підходів до розпізнавання фігур.
-Обрати та реалізувати ефективні алгоритми обробки зображень з використанням OpenCV.
-Розробити архітектуру програмного застосунку.
-Реалізувати графічний інтерфейс користувача на основі PyQt.
-Провести тестування системи за різних умов експлуатації .
-Оцінити ефективність реалізованої системи та визначити можливі напрямки її подальшого розвитку.

Комп'ютерний зір: розвиток, застосування та роль у візуальній обробці

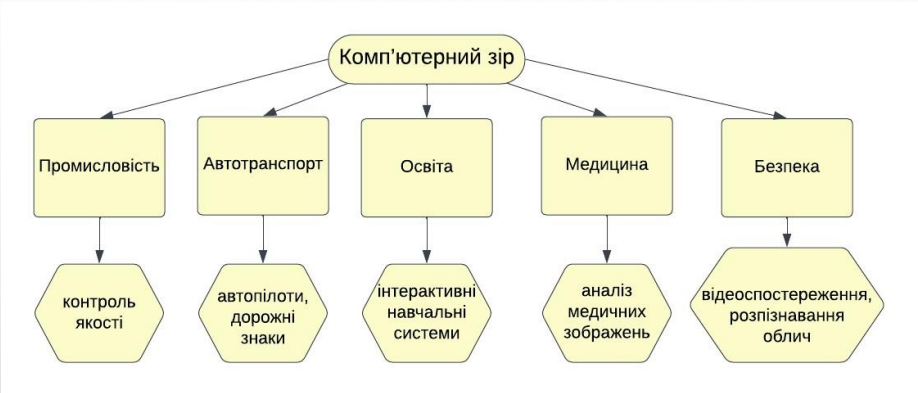


Рисунок 1.1 – Основні сфери застосування комп'ютерного зору у поєднанні зі штучним інтелектом

Методи та алгоритми розпізнавання об’єктів на зображеннях

Класичні алгоритми розпізнавання геометричних фігур

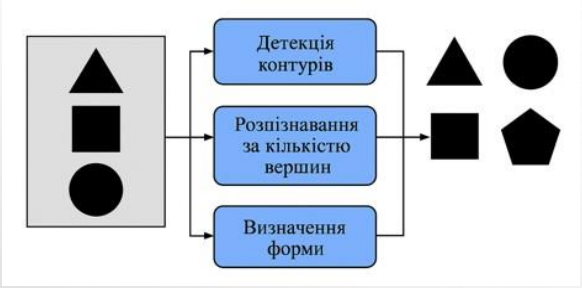


Рисунок 1.2 – Інтеграція класичних методів комп’ютерного зору та ШІ у процесі розпізнавання геометричних об’єктів

Сучасна обробка з використанням методів штучного інтелекту

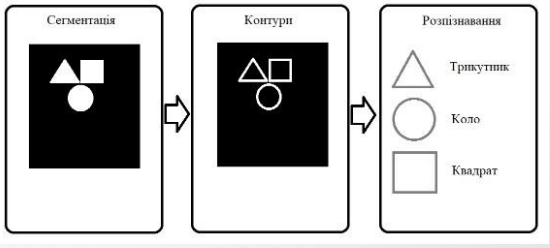


Рисунок 1.3 – Етапи обробки зображення для розпізнавання фігур із використанням методів штучного інтелекту

OpenCV та PyQt у побудові систем комп’ютерного зору

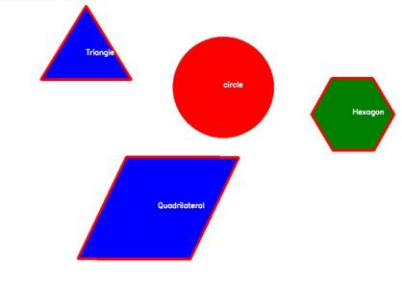


Рисунок 1.4 – Розпізнавання геометричних фігур за допомогою методів комп’ютерного зору на базі бібліотеки OpenCV як складової систем штучного інтелекту

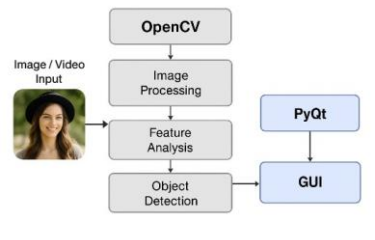


Рисунок 1.5 – Архітектура програмного застосунку для комп’ютерного зору

Критерій	OpenCV	PyQt
Основне призначення	Обробка та аналіз зображень, відео	Створення графічного інтерфейсу користувача (GUI)
Мова програмування	Python, C++, Java	Python
Підтримка відеопотоку	Є (через cv2.VideoCapture())	Непряма (через інтеграцію з OpenCV)
Робота в реальному часі	Оптимізовано для обробки кадрів в реальному часі	Забезпечує оновлення вікна GUI при новому кадрі
Гнучкість налаштування	Висока – підтримує низькорівневі і високорівневі операції	Висока – підтримує кастомізацію елементів інтерфейсу та стилізацію
Складність засвоєння	Середня (потребує знання базових алгоритмів обробки зображень)	Середня (необхідні знання структури Qt-сигналів і слотів)
Документація та спільнота	Широка документація, активна спільнота	Потужна документація, інтеграція з Qt Designer, активна підтримка розробників
Кроссплатформність	Повна (Windows, Linux, macOS, Android)	Повна (Windows, Linux, macOS)
Інтеграція з іншими бібліотеками	TensorFlow, PyTorch, NumPy, scikit-image	OpenCV, Matplotlib, Pandas, SQLAlchemy, тощо
Ідеальне застосування	Детекція, класифікація, сегментація, трекінг об’єктів	Користувацький контроль, відображення результатів, керування параметрами обробки

Архітектура застосунку та логіка взаємодії користувача

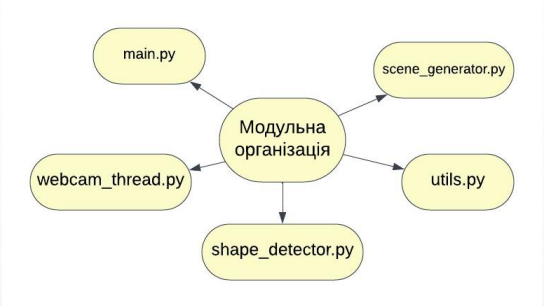


Рисунок 2.1 – Схема модульної організації програмного застосунку для розпізнавання геометричних фігур

Рисунок 2.2 – Блок-схема функціональної взаємодії компонентів застосунку

Алгоритмічні принципи та логіка розпізнавання геометричних фігур


```python
if num_vertices == 3:
    shape_type_ukrainian = "Трикутник"
elif num_vertices == 4:
    if 0.9 <= aspect_ratio <= 1.1:
        shape_type_ukrainian = "Квадрат"
    else:
        shape_type_ukrainian = "Прямокутник"
elif num_vertices == 5:
    shape_type_ukrainian = "П'ятикутник"
elif num_vertices > 5:
    if 0.80 < circularity < 1.20:
        shape_type_ukrainian = "Коло"
    else:
        shape_type_ukrainian = "Багатокутник"
```

Рисунок 2.3 – Фрагмент коду класифікації фігур згідно з кількістю вершин

Рисунок 2.4 – Блок-схема алгоритму розпізнавання геометричних фігур у системі комп'ютерного зору

Використання потокової обробки у роботі з відеопотоком


```
class WebcamThread(QThread):
    frame_ready = pyqtSignal(QPixmap)
    ...
    def run(self):
        self.cap = cv2.VideoCapture(self.camera_index)
        while self.running:
            ret, frame = self.cap.read()
            if ret:
                # Обробка кадру
```

Рисунок 2.5 – Архітектура потокового обслуговування

Обробка кадру в реальному часі

Кожен кадр, отриманий через cv2.VideoCapture, надсилається до основного модуля обробки (shape_detector.detect_shapes). Там здійснюється:

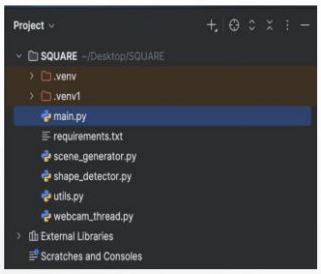
- Перетворення кадру у відтінки сірого;
- Застосування порогової фільтрації;
- Виявлення контурів та їх аналіз;
- Формування інформаційного словника по кожній фігурі;
- Повернення кадру з підписаними та виділеними фігурами.

Результати передаються через сигнал processed_frame_ready у вигляді об’єкта QPixmap, що дає змогу миттєво оновлювати візуальну панель інтерфейсу користувача [24].

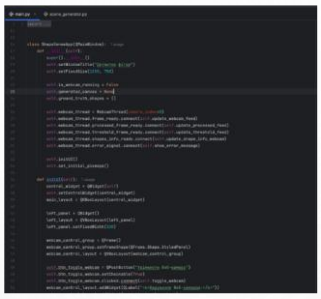
Система захищена від нестабільності потоку шляхом обробки виняткiв. У разі втрати доступу до камери, система видає повідомлення та припиняє захоплення кадрів:

except Exception as e:
print(f"Error initializing camera: {e}")
self.cap = None

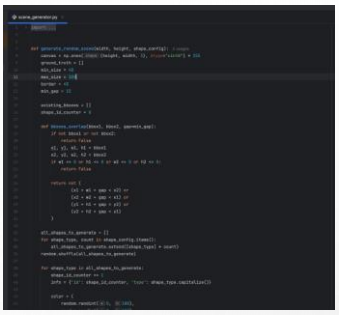
Архітектура модулів та приклади реалізації програмної системи



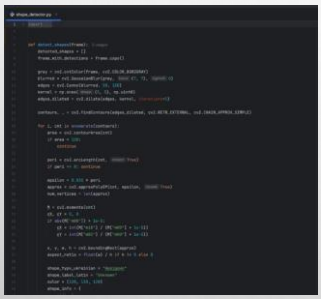
Структура проєкту



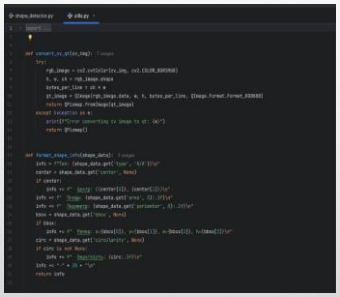
Основний модуль



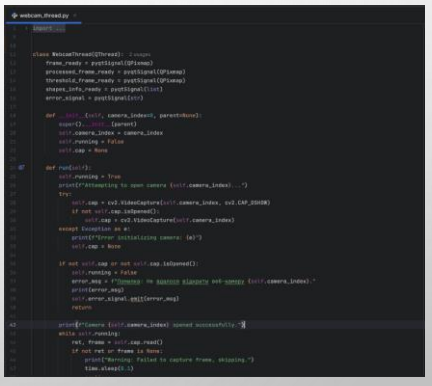
Генератор сцен



Допоміжні функції



Детектор фігур



Потік вебкамери

ІНТЕРФЕЙС КОРИСТУВАЧА ТА РЕЖИМИ РОБОТИ ЗАСТОСУНКУ

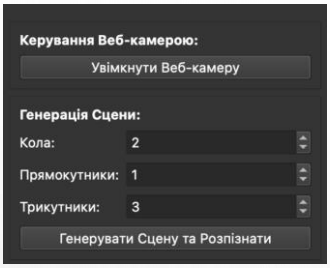


Рисунок 3.8 – Панель керування вебкамерою та генерацією геометричних фігур

Рисунок 3.9 – Вивід текстової інформації про розпізнані фігури в інтерфейсі

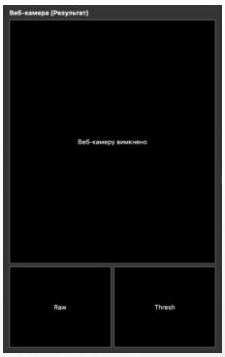


Рисунок 3.10 – Панель виводу зображення з вебкамери: сирий потік, порогове зображення та результат розпізнавання

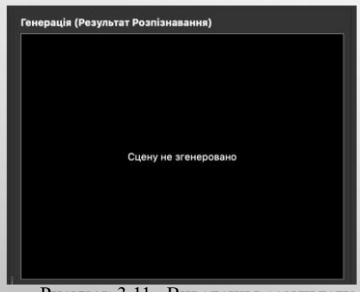


Рисунок 3.11 –Виведення результату розпізнавання на згенерованій сцені

Рисунок 3.12 – Виведення оригінального зображення згенерованої сцени для візуального порівняння

ТЕСТУВАННЯ В УМОВАХ ЗМІННОГО ОСВІТЛЕННЯ ТА ОБМЕЖЕНИХ РЕСУРСІВ

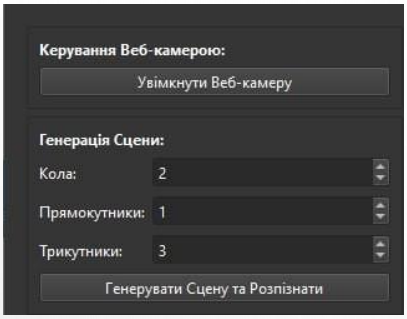


Рисунок 3.13 – Панель керування вебкамерою та параметрами генерації геометричних фігур у застосунку

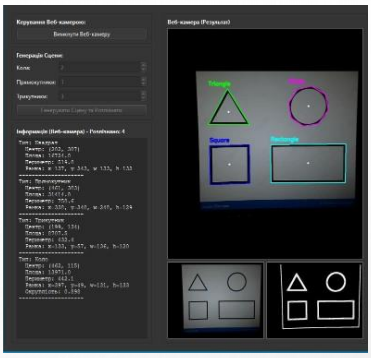


Рисунок 3.14 – Розпізнавання фігур з вебкамери у реальному часі з відображенням результатів і параметрів об’єктів

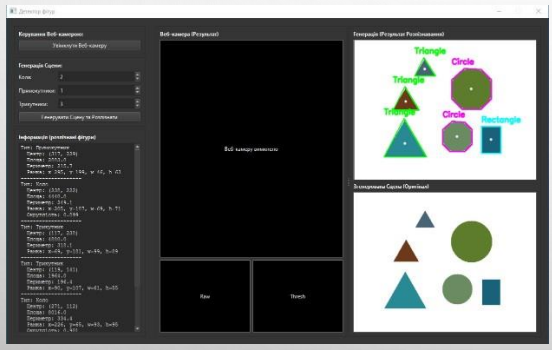
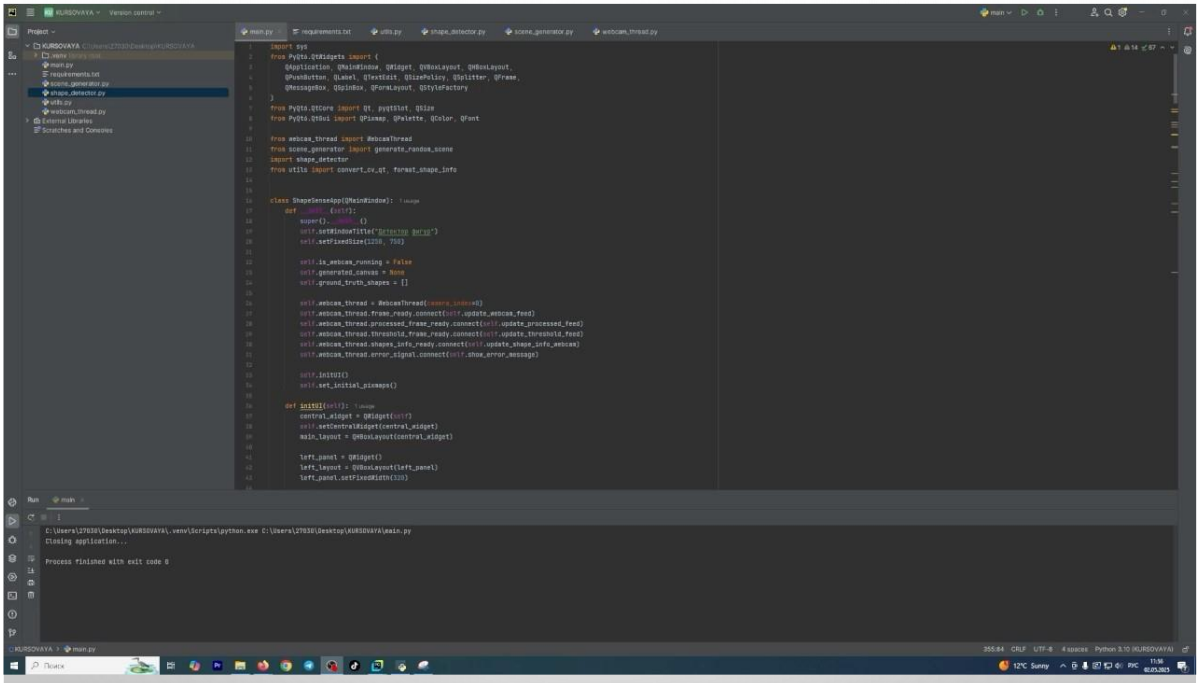


Рисунок 3.15 – Розпізнавання фігур на згенерованій сцені з автоматичним порівнянням результатів і еталонних даних

Демонстрація роботи



ВИСНОВКИ

- Розроблено програмний застосунок для розпізнавання геометричних фігур у реальному часі з вебкамери та на основі попередньо згенерованих сцен.
- Застосовано сучасні підходи до побудови систем комп’ютерного зору з використанням OpenCV і PyQt6 для реалізації алгоритмів обробки зображень і побудови графічного інтерфейсу.
- Реалізовано модульну архітектуру, що забезпечує гнучкість, масштабованість та простоту підтримки. Включено модулі генерації, обробки, розпізнавання та виводу результатів.
- Проведено експериментальне тестування роботи системи в умовах змінного освітлення та фону. Отримано високу точність розпізнавання геометричних об’єктів навіть у складних умовах.
- Виконано кількісну оцінку точності, що базувалася на порівнянні результатів розпізнавання з еталонними даними («ground truth»), згенерованими системою.
- Окреслено перспективи розвитку системи, які включають інтеграцію технологій машинного навчання, розширення переліку розпізнаваних об’єктів та оптимізацію застосунку для мобільних і вбудованих платформ.
- Визначено перспективи розвитку системи, які включають інтеграцію технологій машинного навчання, розширення переліку розпізнаваних об’єктів та оптимізацію застосунку для мобільних і вбудованих платформ.На поточному етапі застосунок є особливо доцільним для використання в промисловості — зокрема для візуального контролю форм деталей під час виготовлення.

ДОДАТОК А

Фрагмент коду для обробки відео з вебкамери (WebcamThread)

```
import cv2
from PyQt6.QtCore import QThread, pyqtSignal
from PyQt6.QtGui import QPixmap
import numpy as np
import time

import shape_detector
from utils import convert_cv_qt


class WebcamThread(QThread):
    frame_ready = pyqtSignal(QPixmap)
    processed_frame_ready = pyqtSignal(QPixmap)
    threshold_frame_ready = pyqtSignal(QPixmap)
    shapes_info_ready = pyqtSignal(list)
    error_signal = pyqtSignal(str)

    def __init__(self, camera_index=0, parent=None):
        super().__init__(parent)
        self.camera_index = camera_index
        self.running = False
        self.cap = None

    def run(self):
        self.running = True
        print(f"Attempting to open camera {self.camera_index}...")
        try:
            self.cap = cv2.VideoCapture(self.camera_index, cv2.CAP_DSHOW)
            if not self.cap.isOpened():
                self.cap = cv2.VideoCapture(self.camera_index)
        except Exception as e:
            print(f"Error initializing camera: {e}")
            self.cap = None

        if not self.cap or not self.cap.isOpened():
            self.running = False
            error_msg = f"Помилка: Не вдалося відкрити веб-камеру {self.camera_index}."
            print(error_msg)
            self.error_signal.emit(error_msg)
            return

        print(f"Camera {self.camera_index} opened successfully.")
        while self.running:
```

```
            ret, frame = self.cap.read()
            if not ret or frame is None:
                print("Warning: Failed to capture frame, skipping.")
                time.sleep(0.1)
                continue

            try:
                processed_frame, detected_shapes, thresh_frame = shape_detector.detect_shapes(frame)
            except Exception as e:
                print(f"Error during shape detection: {e}")
                processed_frame = frame.copy()
                detected_shapes = []
                black = np.zeros_like(frame[:, :, 0], dtype=np.uint8)
                thresh_frame = cv2.cvtColor(black, cv2.COLOR_GRAY2BGR)

            try:
                qt_frame = convert_cv_qt(frame)
                if not qt_frame.isNull():
                    self.frame_ready.emit(qt_frame)

                qt_processed = convert_cv_qt(processed_frame)
                if not qt_processed.isNull():
                    self.processed_frame_ready.emit(qt_processed)

                qt_thresh = convert_cv_qt(thresh_frame)
                if not qt_thresh.isNull():
                    self.threshold_frame_ready.emit(qt_thresh)

                self.shapes_info_ready.emit(detected_shapes)

            except Exception as e:
                print(f"Error converting or emitting frames: {e}")

        if self.cap:
            self.cap.release()
            print(f"Camera {self.camera_index} released.")
        self.cap = None

    def stop(self):
        self.running = False
        print("Stopping webcam thread...")
        if self.isRunning():
            self.wait(500)
        print("Webcam thread stopped.")
```

ДОДАТОК В

Функція розпізнавання геометричних фігур на зображенні (detect_shapes)

```
import cv2
import numpy as np
import math


def detect_shapes(frame):
    detected_shapes = []
    frame_with_detections = frame.copy()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)
    edges = cv2.Canny(blurred, 50, 150)
    kernel = np.ones((5, 5), np.uint8)
    edges_dilated = cv2.dilate(edges, kernel, iterations=1)

    contours, _ = cv2.findContours(edges_dilated, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    for i, cnt in enumerate(contours):
        area = cv2.contourArea(cnt)
        if area < 150:
            continue

        peri = cv2.arcLength(cnt, True)
        if peri == 0: continue

        epsilon = 0.035 * peri
        approx = cv2.approxPolyDP(cnt, epsilon, True)
        num_vertices = len(approx)

        M = cv2.moments(cnt)
        cX, cY = 0, 0
        if abs(M["m00"]) > 1e-5:
            cX = int(M["m10"] / (M["m00"] + 1e-5))
            cY = int(M["m01"] / (M["m00"] + 1e-5))

        x, y, w, h = cv2.boundingRect(approx)
        aspect_ratio = float(w) / h if h != 0 else 0

        shape_type_ukrainian = "Невідомо"
        shape_label_latin = "Unknown"
        color = (128, 128, 128)
        shape_info = {
```

```
    'id': i,
    'type': shape_type_ukrainian,
    'center': (cX, cY),
    'area': area,
    'perimeter': peri,
    'bbox': (x, y, w, h)
}

if num_vertices == 3:
    shape_type_ukrainian = "Трикутник"
    shape_label_latin = "Triangle"
    color = (0, 255, 0)
elif num_vertices == 4:
    if 0.9 <= aspect_ratio <= 1.1:
        shape_type_ukrainian = "Квадрат"
        shape_label_latin = "Square"
        color = (255, 0, 0)
    else:
        shape_type_ukrainian = "Прямокутник"
        shape_label_latin = "Rectangle"
        color = (255, 255, 0)
elif num_vertices == 5:
    shape_type_ukrainian = "П'ятикутник"
    shape_label_latin = "Pentagon"
    color = (0, 165, 255)
elif num_vertices > 5:
    circularity = 4 * math.pi * area / (peri ** 2)
    shape_info['circularity'] = circularity
    if 0.80 < circularity < 1.20:
        shape_type_ukrainian = "Коло"
        shape_label_latin = "Circle"
        color = (255, 0, 255)
    else:
        shape_type_ukrainian = "Багатокутник"
        shape_label_latin = "Polygon"
        color = (200, 200, 200)

shape_info['type'] = shape_type_ukrainian

if shape_type_ukrainian not in ["Невідомо", "Багатокутник"]:
    cv2.drawContours(frame_with_detections, [approx], -1, color, 2)

    cv2.putText(frame_with_detections, shape_label_latin,
                (x, y - 10 if y > 10 else y + h + 15),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6,
```

```
                color,
                2,
                cv2.LINE_AA)

        if cX > 0 and cY > 0:
            cv2.circle(frame_with_detections, (cX, cY), 3, (255, 255, 255), -1)

        detected_shapes.append(shape_info)

thresh_image_bgr = cv2.cvtColor(edges_dilated, cv2.COLOR_GRAY2BGR)
return frame_with_detections, detected_shapes, thresh_image_bgr
```