МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

Кафедра комп'ютерних технологій та інформаційної безпеки

Кваліфікаційна робота
на правах рукопису

КВАЛІФІКАЦІЙНА РОБОТА

Дослідження можливостей бібліотеки OpenCV для систем захисту інформації

Ступінь вищої освіти: магістр
Галузь знань: 14 Електрична інженерія
Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»
Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

Виконав: студент 6 курсу групи 6366м
Курманський Владислав Васильович _____

Кваліфікаційна робота містить результати самостійного виконання навчального завдання. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Керівник:
к.т.н., доцент кафедри КТІБ

Сірівчук Андрій Сергійович _____

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут автоматики і електротехніки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних технологій та інформаційної безпеки, к.т.н., доцент
_____ С.М. Нужний
«____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

Студент: Курманський Владислав Васильович

Курс:6

Група: 6366м

Ступінь вищої освіти: магістр

Галузь знань: 14 Електрична інженерія

Спеціальність: 141 «Електроенергетика, електротехніка та електромеханіка»

Освітньо-професійна програма: Системи технічного захисту інформації, автоматизація її обробки

1. Тема роботи: Дослідження можливостей бібліотеки OpenCV для систем захисту інформації

затверджена наказом НУК від «_14__» жовтня 2024 р. № 1075 - уч

2. Вихідні дані до роботи: об'єкт дослідження – вбудовані функції бібліотеки OpenCV; реалізація можливостей – знаходження контуру, сегментація зображення, пошук об'єктів за шаблоном та пошук обличчя на зображенні.

3. Перелік питань, які необхідно розробити: основи роботи з OpenCV; особливості роботи з зображеннями та відео; демонстрація прикладів використання OpenCV для комп'ютерного зору та системи розпізнавання образів.

4. Терміни виконання завдання:
термін видачі завдання: «_04_» вересня 2024 р.
термін подання роботи: «_19_» грудня 2024 р.

6. План-графік виконання кваліфікаційної роботи

№ п/п	Заходи	Дата		Готовність	Відмітка про виконання
		Навч. тиждень	Контрольна дата		
1.	Наукове стажування	1 - 8	25.10.2024	Звіт з наукового стажування	
2.	Організаційні збори	9	31.10.2024	Попередній варіант змісту КР	
3.	Рубіжний контроль	10	05.11.2024	Попередній варіант оглядових розділів	
4.	Рубіжний контроль	13	28-29.11.2024	1. Попередній варіант повної КР. 2. Попередній варіант презентації.	
5.	Рубіжний контроль	15	12.12.2024	Доповідь на 12-ій Всеукраїнській науково-технічній конференції з міжнародною участю «СПІБТ–2024»	
6.	Перевірка на плагіат	15	12-13.12.2024	Електронний варіант кваліфікаційної роботи	
7.	КЕ на рівень « магістра»	16	17-18.12.2024	Здавання державного екзамену зі спеціальності на ступінь бакалавра	
8.	Кафедральний захист	16	19.12.2024	1. Оформлена КР (в форматі pdf) (передається студентом на кафедрі для внутрішньої рецензії). У разі отримання позитивної внутрішньої рецензії, КР подається на зовнішню рецензію. 2. Оформлена презентація (в форматі pdf).	
9.	Подання документів на захист	16	20.12.2024	1. Подання щодо захисту КР: тема, успішність, висновок керівника та висновок кафедри про КР. 2. Зовнішня рецензія (окремо від КР). Резюме на КР. 3. Електронний варіант та роздрукована презентація в кількості 5 примірників. 4. Електронний варіант КР на диску	
10.	Захист ДР	17	24,26,27, 28.12.2024	1. Приєднання студента до засідання кваліфікаційної комісії (конференції) у встановлений час для проведення процедури дистанційного захисту 2. Процедура захисту КР.	

ПРИМІТКА: Завдання додається до виконаної роботи та подається до атестаційної комісії.

Керівник
К.т.н., доцент _____ А.С. Сірівчук

Студент _____ В.В. Курманський

АНОТАЦІЯ

Курманський В.В. Дослідження можливостей бібліотеки Opencv для систем захисту інформації

Кваліфікаційна робота: 69 с.; 21 рис.; 35 джерел,.

Актуальність роботи визначається постійно зростаючою тенденцією використання систем комп’ютерного зору в багатьо сферах діяльності. Одним з ключових засобів обробки цифрових зображень для задач компютерного зору є бібліотека OpenCV.

Метою роботи є дослідження функціональних можливостей бібліотеки OpenCV, як основи для систем комп’ютерного зору та цифрової обробки зображень.

У кваліфікаційній роботі проведено проведено аналіз задач в сфері систем захисту інформації, що вирішуються за допомогою систем комп’ютерного зору. Також описано особливості роботи бібліотеки OpenCV, та реалізовано базові алгоритми обробки зображень, що дають основу до майбутніх досліджень.

Результати роботи можуть бути використанні при підготовці студентів, а також в науковій роботі кафедри та студентів в системах обробки цифрових зображень.

Ключові слова: OpenCV, комп’ютерний зір, обробка цифрових зображень, виявлення обличчя.

ANNOTATION

Kurmansky V.V. Research of the capabilities of the Opencv library for information security systems

Qualification work: 69 *p.;* 21 *fig.;* 35 *sources,.*

The relevance of the work is determined by the constantly growing trend of using computer vision systems in many areas of activity. One of the key tools for processing digital images for computer vision tasks is the OpenCV library.

The purpose of the work is to study the functional capabilities of the OpenCV library as a basis for computer vision systems and digital image processing.

The qualification work analyzes the tasks in the field of information security systems that are solved using computer vision systems. The features of the OpenCV library are also described, and basic image processing algorithms are implemented, which provide the basis for future research.

The results of the work can be used in the training of students, as well as in the scientific work of the department and students in digital image processing systems.

Keywords: OpenCV, computer vision, digital image processing, face detection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

3*D* – тривимірний

2*D* – двовимірний

OpenCV – Open Source Computer Vision Library

SVM – алгоритми опорних векторних машин

KNN – K-Nearest Neighbors

YOLO – You Look Only Once

ВСТУП

У сучасному світі безпека та спостереження стають дедалі важливішими аспектами повсякденного життя. Оскільки технології продовжують розвиватися, комп'ютерне бачення стали потужними інструментами у сфері спостереження та безпеки.

Комп'ютерний зір – це галузь дослідження, спрямована на те, щоб дозволити машинам інтерпретувати та розуміти візуальні дані з навколишнього світу. *OpenCV*, бібліотека комп'ютерного бачення з відкритим кодом, є популярним інструментом, який використовується для розробки програм комп'ютерного бачення в реальному часі. Завдяки широкому спектру функцій і алгоритмів *OpenCV* став важливим інструментом для розробників і дослідників, які працюють у сфері комп'ютерного зору.

Одним із найважливіших застосувань комп'ютерного зору та *OpenCV* у сфері спостереження та безпеки є розпізнавання обличчя. Технологія розпізнавання обличчя стала важливим інструментом у правоохоронних органах, що дозволяє більш ефективно та точно ідентифікувати підозрюваних.

Технологія виявлення об'єктів стала важливим інструментом у безпеці аеропорту та прикордонному контролі, що дозволяє більш ефективно та точно перевіряти людей та їхні речі.

Метою роботи є дослідження функціональних можливостей *OpenCV* в сфері інформаційної безпеки для впровадження в науковий і навчальний процес кафедри.

Обєктом дослідження є принципи роботи з бібліотекою *OpenCV* та визначення основного функціоналу ядра.

Предметом дослідження є базові функції *OpenCV*, що лежать в складних алгоритмах виявлення об'єктів, детекторів руху або охороних систем.

Для виконання поставленої задачі необхідно вирішити наступні задачі:

– виділення основних задач що вирішуються при цифровій обробці зображень;

– визначення структури та принципів роботи *OpenCV*;

– реалізація та демонстрація роботи базових можливостей *OpenCV*, що використовуються в системах захисту інформації.

Результати роботи частково опубліковані в наступному джерелі:

Курманський В.В. Застосування бібліотеки *OpenCV* в охоронних системах. Матеріали: Всеукраїнська науково-технічна конференції з міжнародною участю «Сучасні проблеми інформаційної безпеки на транспорті». Миколаїв: НУК, 2024.

Результати роботи можуть бути використанні при підготовці студентів в рамках дисциплін: «Системи комп’ютерного зору», «Системи розпізнавання образів», «Кіберфізичні системи та Інтернет речей» та «Захист аудіо- та відеоінформації». А також в науковій роботі кафедри та студентів в системах обробки цифрових зображень.

1. ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ СИСТЕМ ОБРОБКИ ЗОБРАЖЕНЬ

1.1 Цифрова обробка зображень та її задачі

Цифрова обробка зображень означає обробку цифрових зображень за допомогою комп'ютера. Ми також можемо сказати, що це використання комп’ютерних алгоритмів для отримання покращеного зображення або для отримання корисної інформації.

Обробка цифрових зображень – це використання алгоритмів і математичних моделей для обробки й аналізу цифрових зображень. Метою цифрової обробки зображень є підвищення якості зображень, вилучення значущої інформації із зображень і автоматизація завдань, пов’язаних із зображеннями.

Основні етапи обробки цифрових зображень:

– отримання зображення: це передбачає захоплення зображення за допомогою цифрової камери чи сканера або імпорт існуючого зображення на комп’ютер;

– покращення зображення: це передбачає покращення візуальної якості зображення, наприклад підвищення контрастності, зменшення шуму та видалення артефактів;

– відновлення зображення: це передбачає усунення погіршення якості зображення, наприклад розмиття, шуму та спотворення;

– сегментація зображення: передбачає поділ зображення на області або сегменти, кожен з яких відповідає певному об’єкту чи функції зображення;

– репрезентація та опис зображення: це передбачає представлення зображення таким чином, щоб його можна було проаналізувати та обробити комп’ютером, а також опис характеристик зображення в стислий та змістовний спосіб;

– аналіз зображення: це передбачає використання алгоритмів і математичних моделей для отримання інформації із зображення, наприклад розпізнавання об’єктів, виявлення шаблонів і кількісного визначення характеристик;

– синтез і стиснення зображень: це передбачає створення нових зображень або стиснення існуючих зображень для зменшення вимог до зберігання та передачі.

Обробка цифрових зображень широко використовується в різноманітних програмах, включаючи медичні зображення, дистанційне зондування, комп’ютерне зір і мультимедіа.

Обробка зображення в основному включає наступні етапи:

– імпортування зображення за допомогою інструментів отримання зображення;

– аналіз та маніпулювання зображенням;

– вихід, результатом якого може бути змінене зображення або звіт, заснований на аналізі цього зображення.

Зображення в цифровому вигляді можутиь мати декіотка різновидів: бінарне, відтінки сірого, 8-бітний формат кольорів,16-бітний формат кольору.

Бінарне зображення – як випливає з назви, двійкове зображення містить лише два піксельні елементи, тобто 0 і 1, де 0 означає чорний колір, а 1 – білий. Це зображення також відоме як монохромне. Бінарне (двійкове) зображення іноді можуть називати «монохромним» і «чорно-білим», що в загальному випадку невірно і іноді веде до плутанини. Бінарне растрове зображення може бути монохромним у таких випадках:

– один вид пікселів (не важливо "0" або "1") відображається абсолютно чорним кольором, а інший будь-яким довільним (наприклад, "чорно-білий" растр, "чорно-зелений" тощо);

– обидва види (і «0», і «1») відображаються одним відтінком, але з різною яскравістю (наприклад, «темно- і світло-сірий», «яскраво-зелений та темно-зелений», «синій і сірий» тощо) .

Бінарне зображення не буде монохромним, якщо різні види пікселів відображаються різними відтінками кольору (наприклад, жовто-зелене, червоно-синє растрове зображення тощо). ;

Чорно-біле зображення (відтінки сірого) – зображення, яке складається лише з чорного та білого кольорів, називається чорно-білим зображенням.

8-бітний формат кольорів – це найвідоміший формат зображення. Він містить 256 різних відтінків кольорів і широко відомий як зображення у відтінках сірого. У цьому форматі 0 означає чорний, 255 – білий, а 127 – сірий.

16-бітний формат кольору – це формат кольорового зображення. Він містить 65 536 різних кольорів. Він також відомий як формат високого кольору. У цьому форматі розподіл кольорів не такий самий, як зображення у відтінках сірого.

Цифрова обробка зображень дозволяє використовувати набагато складніші алгоритми, а отже, може запропонувати як більш складну продуктивність у простих завданнях, так і реалізацію методів, які були б неможливі за допомогою аналогових засобів.

1.2 Область застосування цифрової обробки зображень

Зокрема, обробка цифрових зображень є конкретним застосуванням і практичною технологією. Основними завданнями є класифікація, вилучення ознак, проекція та розпізнавання образів.

Класифікація має багато застосувань. У деяких із них це використовується як процедура інтелектуального аналізу даних , тоді як в інших виконується більш детальне статистичне моделювання.

Вилучення ознак – застосування інженерії функцій є кластеризація об’єктів-об’єктів або вибіркових об’єктів у наборі даних. Зокрема, розробка ознак, заснована на декомпозиції матриці, широко використовується для кластеризації даних за обмежень невід’ємності на коефіцієнти ознак. До них належать невід’ємна матриця факторизації (*NMF*), невід’ємна матриця-

тріфакторізація (*NMTF*), невід’ємна тензорна декомпозиція/факторизація (*NTF/NTD*) тощо. Невід’ємність обмеження на коефіцієнти векторів ознак, отриманих за допомогою вищевказаних алгоритмів, дає представлення на основі частин, а різні факторні матриці демонструють природні властивості кластеризації. У літературі повідомлялося про кілька розширень вищевказаних методів розробки функцій, включаючи факторізацію з обмеженням ортогональності для жорсткої кластеризації та різноманітне навчання для подолання властивих цим алгоритмам проблем.

Розпізнавання образів – має застосування в аналізі статистичних даних , обробці сигналів , аналізі зображень, пошуку інформації , біоінформатиці , стисненні даних , комп'ютерній графіці та машинному навчанні . Розпізнавання образів бере свій початок у статистиці та інженерії; деякі сучасні підходи до розпізнавання образів включають використання машинного навчання завдяки збільшенню доступності великих даних і новій кількості процесорних можливостей .

Проекція – це техніка проектування, яка використовується для відображення тривимірного (3*D*) об’єкта на двовимірній (2*D*) поверхні. Ці проекції спираються на візуальну перспективу та аналіз аспектів, щоб спроектувати складний об’єкт для можливості перегляду на простішій площині.

Області застосування цифрової обробки практично не має меж.

Обробка супутникових зображень є важливою галуззю досліджень і розробок і складається із зображень землі та супутників, зроблених за допомогою штучних супутників. Спочатку фотографії робляться в цифровому вигляді, а потім обробляються комп’ютерами для отримання інформації. Статистичні методи застосовуються до цифрових зображень, і після обробки різні дискретні поверхні ідентифікуються шляхом аналізу значень пікселів.

Супутникові зображення широко використовуються для планування інфраструктури або для моніторингу умов навколишнього середовища або для виявлення реагування на майбутні катастрофи.

У ширшому плані ми можемо сказати, що обробка супутникових зображень є різновидом дистанційного зондування, яке працює з роздільною здатністю пікселів для збору когерентної інформації про земну поверхню.

Відстеження відео – визначення місцезнаходження рухомого об'єкта шляхом аналізу кадрів відео

Медична візуалізація: комп'ютерний зір допомагає в реконструкції МРТ, автоматичній патології, діагностиці та операціях за допомогою комп'ютера тощо. Аналіз медичного зображення та медична візуалізація – Техніка та процес створення візуальних зображень внутрішньої частини тіла

AR/VR: оклюзія об'єктів, відстеження ззовні всередину та відстеження зсередини назовні для віртуальної та доповненої реальності.

Смартфони: усі фотофільтри (включаючи фільтри анімації в соціальних мережах), сканери *QR*-кодів, створення панорам, комп'ютерна фотографія, детектори обличчя, детектори зображень, як-от (*Google Lens, Night Sight*), які ми використовуємо, є програмами комп'ютерного зору.

Інтернет: пошук зображень, картографування, підписи до фотографій, зображення Ariel для карт, категоризація відео тощо.

В сфері захисту інформації може використовуватисб для повторна ідентифікація людини — це завдання комп'ютерного зору, мета якого полягає в тому, щоб зіставити особу людини на різних камерах або місцях у послідовності відео або зображень. Це передбачає виявлення та відстеження особи, а потім використання таких ознак, як зовнішній вигляд, форма тіла та одяг, щоб зіставити її особу в різних кадрах. Мета полягає в тому, щоб пов'язати ту саму особу з кількома видами камери, які не перекриваються, надійним і ефективним способом.

Одним із найважливіших застосувань комп'ютерного зору та *OpenCV* у сфері спостереження та безпеки є розпізнавання обличчя. Аналізуючи шаблони рис обличчя та відображаючи їх у базі даних відомих людей, алгоритми комп'ютерного зору можуть швидко й точно ідентифікувати людей у реальному часі. Технологія розпізнавання обличчя стала важливим інструментом у

правоохоронних органах, що дозволяє більш ефективно та точно ідентифікувати підозрюваних.

Ще одним важливим застосуванням комп'ютерного зору та *OpenCV* у сфері спостереження та безпеки є виявлення об'єктів. Аналізуючи шаблони у візуальних даних, алгоритми комп'ютерного зору можуть швидко ідентифікувати та відстежувати цікаві об'єкти, наприклад транспортні засоби чи зброю. Технологія виявлення об'єктів стала важливим інструментом у безпеці аеропорту та прикордонному контролі, що дозволяє більш ефективно та точно перевіряти людей та їхні речі.

Виявлення рухомих об'єктів широко використовується для різноманітних додатків, починаючи від охоронного спостереження й закінчуючи моніторингом руху. Це важливий виклик у сфері комп'ютерного зору, що постійно розвивається. Бібліотека *OpenCV* із відкритим кодом, відома своїм повним набором інструментів для комп'ютерного зору, надає надійні рішення для виявлення рухомих об'єктів. У цій статті ми розглядаємо комбінацію виявлення контурів і віднімання фону , які можна використовувати для виявлення рухомих об'єктів за допомогою OpenCV .розпізнавання облич.

Система пожежної сигналізації з використанням *OpenCV* – це рішення на основі комп'ютерного зору, призначене для раннього виявлення пожеж у відеопотоках у реальному часі. Пожежі становлять суттєву загрозу як безпеці людей, так і майну, що підкреслює важливість своєчасного втручання. Ця система використовує *OpenCV*, потужну бібліотеку комп'ютерного зору, для ідентифікації моделей полум'я та диму у відеокадрах, викликаючи попередження після виявлення. Отже, ми пропонуємо метод виявлення диму та вогню на основі мережі *YOLOv*5 як рішення для вирішення цієї проблеми.

Системи підрахунку відвідувачів призначені для врахування кількості людей, які пройшли через певний прохід за певний проміжок часу. Також іноді важливо визначити напрямок руху, проте найчастіше системи обмежуються поділом людей, що проходять, на два класи: вхідні та виходять. Точність підрахунку безпосередньо залежить від досконалості використовуваної

технології. Система підрахунку, як правило, встановлюється на вході до приміщення, дозволяючи стежити за кількістю відвідувачів.

1.3 Опис бібліотеки *OpenCV*

OpenCV, скорочення від *Open Source Computer Vision Library*, є необхідним набором інструментів для тих, хто працює з комп'ютерним зором і машинним навчанням. Це відкритий вихідний код, що означає, що будь-хто може використовувати та налаштовувати його для будь-яких проектів, від великих компаній, таких як *Google*, до невеликих стартапів і наукових досліджень.

OpenCV наповнений алгоритмами, які допомагають у всьому, від розпізнавання об'єктів до відстеження руху об'єктів і навіть створення 3D-моделей. Він неймовірно популярний, його величезна спільнота налічує понад 47 000 користувачів і понад 18 мільйонів завантажень.

OpenCV підтримує *C++*, *Python*, *Java* і *MATLAB* і працює в *Windows, Linux, Android* і *MacOS*. Завдяки ефективному використанню комп'ютерної обробки, це особливо добре для проектів, які повинні працювати в режимі реального часу. Завдяки постійному розвитку таких технологій, як *CUDA* та *OpenCL, OpenCV* є основою незліченних інноваційних програм у всьому світі, від спостереження та робототехніки до інтерактивного мистецтва.

1.3.1 Архітектура *OpenCV*

Архітектура *OpenCV* (рис. 1.1) розроблена для роботи з широким спектром програм комп'ютерного зору та машинного навчання. Він побудований навколо основного компонента *CXCore*, який включає його основні функції та алгоритми. Таке налаштування мінімізує надмірність і підвищує ефективність.

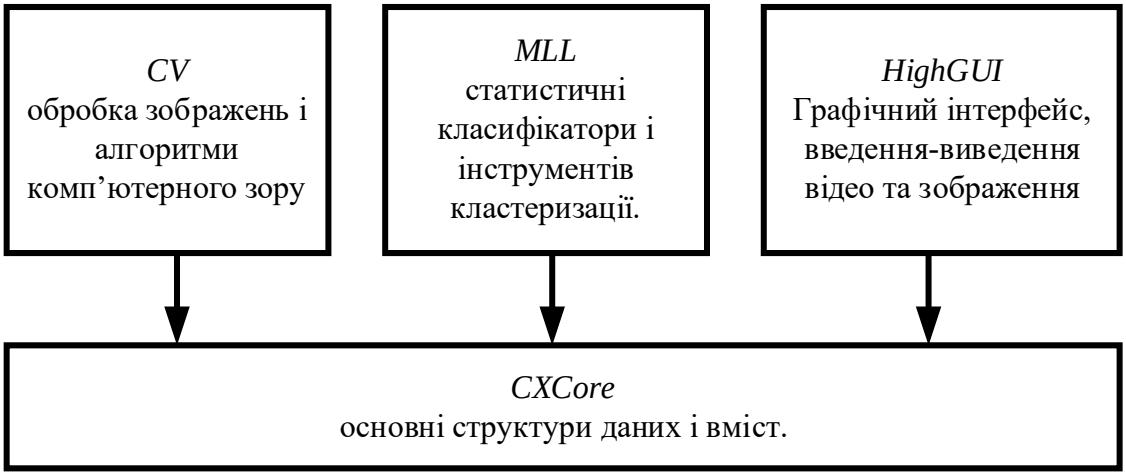


Рисунок 1.1 – Архітектура бібліотеки OpenCV

На додаток до ядра, є такі компоненти, як *CV*, який зосереджується на обробці зображень і алгоритмах зору, і *MLL*, який включає статистичні класифікатори та інструменти кластеризації. Іншим ключовим компонентом є *HighGUI*, який налаштований на функції графічного інтерфейсу користувача, операції введення/виведення зображень і відео. Цей модульний дизайн дозволяє гнучко інтегрувати різні функції машинного навчання.

Алгоритми *OpenCV*.

Бібліотека *OpenCV* містить понад 2500 оптимізованих алгоритмів. Ці алгоритми дозволяють виконувати різноманітні завдання, включаючи класифікацію зображень, виявлення та сегментацію об'єктів, виділення ознак і навіть розпізнавання людського почерку.

Алгоритми опорних векторних машин (*SVM*) і K-Nearest Neighbors (KNN) є хорошими варіантами класифікації зображень шляхом групування подібних точок даних. Цей процес можна використовувати для розрізнення таких категорій, як тварини чи предмети.

Дерева рішень зазвичай використовуються з іншими техніками машинного навчання, такими як глибоке навчання, для покращення продуктивності в таких завданнях, як виявлення об'єктів і сегментація зображень.

Аспект глибокого навчання *OpenCV* включає нейронні мережі з підтримкою фреймворків, таких як *TensorFlow* і *PyTorch*. Крім того, *OpenCV*

постійно розвивається та інтегрує нові моделі, такі як *YOLO* (*You Look Only Once*) для виявлення об’єктів і *Vision Transformers* (*ViTs*), які застосовують трансформемну архітектуру для таких завдань, як класифікація та виявлення зображень.

1.3.2 Приклади використання

OpenCV пропонує надійні набори інструментів для аналізу як 2*D*, так і 3*D* функцій у зображеннях і відео. Цю функцію можна використовувати для таких завдань, як зіставлення зображень, відстеження об’єктів і програм доповненої реальності.

Наприклад (рис. 1.2), *OpenCV* може виявляти та зіставляти ключові характеристики в зображеннях для створення панорамних зображень або створення 3*D*-моделей із кількох зображень. *OpenCV* можна навіть використовувати для реконструкції сцени, як показано нижче. На оригінальному зображенні будівля прихована за переднім планом. На вихідному зображенні сцена реконструйована, тому будівля більше не прихована.

Рисунок 1.2 – Реконструкція зовнішнього виду будівлі

Розробка систем розпізнавання жестів стає легкою за допомогою *OpenCV*. Розпізнавання жестів дозволяє користувачам взаємодіяти з комп’ютерами або пристроями за допомогою жестів і рухів (рис. 1.3).

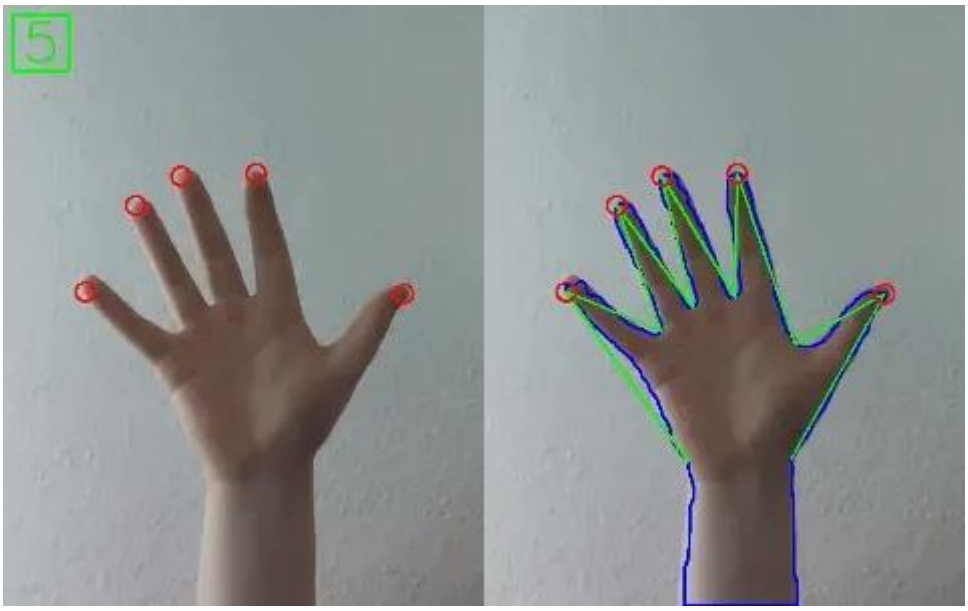


Рисунок 1.3 – Приклад захоплення руки

Розуміння руху та виявлення об'єктів. Алгоритми *OpenCV* для розуміння руху та виявлення об’єктів можна використовувати у відеоспостереженні, автономних транспортних засобах та робототехніці. Наприклад, їх можна використати для розробки системи виявлення руху, яка відстежує канали камер безпеки. Така система могла б попередити операторів про підозрілі рухи в режимі реального часу. На рисунку 1.4 показана техніка, яка називається відніманням фону, виявляє рух і, таким чином, дізнається, що об’єкт (у цьому випадку собака) потрапив на зображення.

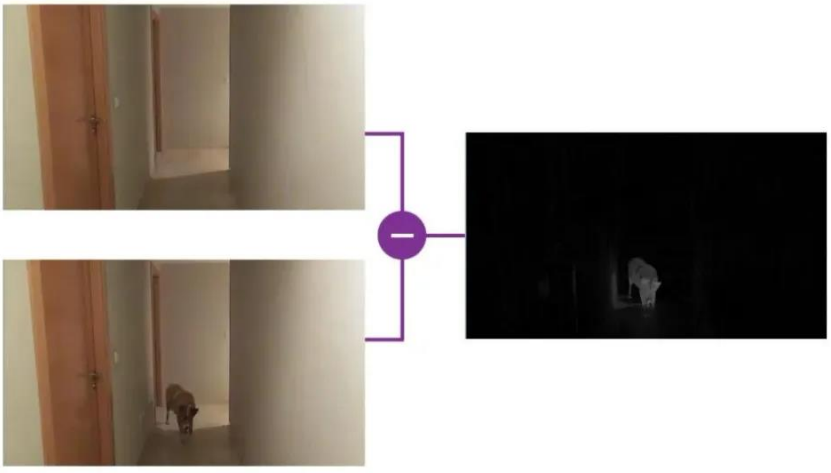


Рисунок 1.4 – Виявлення руху методом віднімання фону

Доповнена реальність. *OpenCV* відіграє важливу роль у додатках доповненої реальності, дозволяючи розробникам накладати цифровий вміст на реальний світ у режимі реального часу. Ця технологія використовується в іграх, освіті та маркетингу.

Наприклад, *OpenCV* можна використовувати для виявлення маркерів або об’єктів у фізичному середовищі та накладання на них віртуальних елементів, таких як 3*D*-моделі чи інформація. Це робить роботу користувача більш захоплюючою, поєднуючи цифровий і фізичний світи.

Висновоки до розділу

Цифрова обробка зображень може вирішити низку задач які можна використовувати не тільки для комп’ютерного зору, а й в системах захисту інформації, зокрема в відеосистемах з виявленням руху, пожежних системах тощо.

2. РОБОТА З ЗОБРАЖЕННЯМИ В *OPENCV*

2.1 Формат представлення зображення

У нас є кілька способів отримати цифрові зображення з реального світу: цифрові камери, сканери, комп'ютерна томографія та магнітно-резонансна томографія. У кожному разі ми (люди) бачимо зображення. Однак, перетворюючи це на наші цифрові пристрої, ми записуємо числові значення для кожної з точок зображення.

Цифрове зображення представлене на комп'ютері матрицею пікселів. Кожен піксель зображення зберігає ціле число. Якщо ми маємо справу із зображенням у градаціях сірого, ми використовуємо числові значення від 0 (чорні пікселі) до 255 (білі пікселі). Будь-яке число між цими двома є відтінком сірого. З іншого боку, кольорові зображення представлені трьома матрицями. Кожна з цих матриць представляє один основний колір, який також називається каналом. Найбільш поширеною моделлю кольорів є червоний, зелений, синій (*RGB*). Ці три кольори змішуються разом, щоб отримати широкий діапазон кольорів. Зауважте, що *OpenCV* завантажує кольорові зображення у зворотному порядку, так що синій канал є першим, зелений – другим, а червоний – третім (*BGR*) (рис 2.1).

Для представлення значень інтенсивності одного каналу в зображенні *RGB* ми також використовуємо значення від 0 до 255. Кожен канал створює загалом 256 дискретних значень, що відповідає загальній кількості бітів, які ви використовуєте для представлення значення каналу кольору.28= 256. Оскільки є три різні канали з 8 бітами на канал, ми називаємо це 24-бітною глибиною кольору.

	column 0			column 1			column …			column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	…	…	…	0,m	0,m	0,m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	…	…	…	1,m	1,m	1,m
Row …	...,0	...,0	...,0	...,1	...,1	...,1	…	…	…	...,m	...,m	...,m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,…	n,…	n,…	n,m	n,m	n,m

Рисунок 2.1 – Формат представлення кольорофого зображення

Після поділу воно розбивається на 3 одноканальних зображення. Кожен піксельний блок у кожному одноканальному зображенні потребує лише одного матричного елемента для збереження, а значення пікселя зазвичай становить 0–255, як показано на малюнку 2, це єдина матрична форма зображення каналу. Нарешті виберіть відповідний одноканальне зображення відповідно до потреб (рис. 2.2).

	column 0	column 1	column …	column m
Row 0	0,0	0,1	…	0,m
Row 1	1,0	1,1	…	1,m
Row …	...,0	...,1	…	...,m
Row n	n,0	n,1	n,…	n,m

Рисунок 2.2 – Формат представлення чорно-білого зображення

2.2 Концепції *API*

2.2.1 Базові модулі

OpenCV має модульну структуру, що означає, що пакет включає кілька спільних або статичних бібліотек. Доступні такі модулі:

– основна функціональність – Цей модуль охоплює основні структури даних, такі як *Scalar, Point, Range* тощо, які використовуються для створення програм *OpenCV*. На додаток до них, він також включає багатовимірний масив *Mat* , який використовується для зберігання зображень.

– обробка зображень – включає лінійну та нелінійну фільтрацію зображень, геометричні перетворення зображення (зміна розміру, афінне та перспективне викривлення, загальне перевідображення на основі таблиці), перетворення простору кольорів, гістограми тощо.

– *video* – цей модуль охоплює концепції аналізу відео, такі як оцінка руху, віднімання фону та відстеження об’єктів.

– *calib3d* – базові алгоритми багаторакурсної геометрії, калібрування одиночної та стереокамери, оцінка пози об'єкта, алгоритми стереовідповідності та елементи 3D-реконструкції.

– *Features2d* – детектори, дескриптори та відповідники дескрипторів помітних ознак.

– *objdetect* – виявлення об'єктів і екземплярів попередньо визначених класів (наприклад, обличчя, очі, чашки, люди, машини і так далі).

– *highgui* – простий у використанні інтерфейс для простих можливостей інтерфейсу користувача.

– *Video I/O* – простий у використанні інтерфейс для захоплення відео та відеокодеків.

– *gpu* – *GPU*-прискорені алгоритми з різних модулів *OpenCV*.

2.2.2 Простір імен cv

Усі класи та функції *OpenCV* розміщено в просторі імен *cv*. Тому, щоб отримати доступ до цієї функції з вашого коду, використовуйте специфікатор *cv*:: або використання простору імен *cv*; директива:

```
#include "opencv2/core.hpp"
...
cv :: Mat H = cv :: findHomography ( points1 , points2 , CV_RANSAC , 5 );
...
```

або:

```
#include "opencv2/core.hpp"
using namespace cv;
...
Mat H = findHomography(points1, points2, CV_RANSAC, 5 );
...
```

Деякі з поточних або майбутніх зовнішніх імен *OpenCV* можуть конфліктувати з *STL* або іншими бібліотеками. У цьому випадку використовуйте явні специфікатори простору імен, щоб вирішити конфлікти імен:

```
Mat  a ( 100 ,  100 ,  CV_32F );
randu ( a ,  Scalar :: all ( 1 ),  Scalar :: all ( std :: rand ()));
cv :: log ( a ,  a );
a  /=  std :: log ( 2. );
```

2.2.3 Автоматичне керування пам'яттю

OpenCV обробляє всю пам'ять автоматично.

Перш за все, *std*::*vector*, Mat та інші структури даних, які використовуються функціями та методами, мають деструктори, які за потреби звільняють базові буфери пам’яті. Це означає, що деструктори не завжди звільняють буфери, як у випадку Mat. Вони враховують можливий обмін даними. Деструктор зменшує лічильник посилань, пов'язаний з буфером даних матриці. Буфер звільняється тоді і тільки тоді, коли лічильник посилань досягає нуля, тобто коли жодна інша структура не посилається на той самий буфер. Подібним чином, коли екземпляр *Mat* копіюється, фактичні дані насправді не копіюються. Натомість лічильник посилань збільшується, щоб запам’ятати, що є інший власник тих самих даних. Існує також метод *Mat*::*clone*, який створює повну копію даних матриці. Дивіться приклад нижче:

```
// create a big 8Mb matrix
Mat A(1000, 1000, CV_64F);
// create another header for the same matrix;
// this is an instant operation, regardless of the matrix size.
Mat B = A;
// create another header for the 3-rd row of A; no data is copied either
Mat C = B.row(3);
// now create a separate copy of the matrix
Mat D = B.clone();
// copy the 5-th row of B to C, that is, copy the 5-th row of A
// to the 3-rd row of A.
B.row(5).copyTo(C);
// now let A and D share the data; after that the modified version
// of A is still referenced by B and C.
A = D;
// now make B an empty matrix (which references no memory buffers),
// but the modified version of A will still be referenced by C,
// despite that C is just a single row of the original A
B.release();
// finally, make a full copy of C. As a result, the big modified
// matrix will be deallocated, since it is not referenced by anyone
C = C.clone();
```

Ви бачите, що використання *Mat* та інших базових структур просте. Але як щодо класів високого рівня або навіть типів даних користувача, створених без урахування автоматичного керування пам’яттю? Для них *OpenCV* пропонує клас шаблону *Ptr*, подібний до *std*::*shared_ptr* із *C*++11. Отже, замість використання простих покажчиків:

```
T* ptr = new T(...);
```

ви можете використовувати:

```
Ptr<T> ptr(new T(...));
```

або:

```
Ptr<T> ptr = makePtr<T>(...);
```

Ptr<T> інкапсулює вказівник на екземпляр *T* і лічильник посилань, пов'язаний із цим вказівником. Дивіться опис *Ptr* для деталей.

2.2.4 Автоматичний розподіл вихідних даних

OpenCV автоматично звільняє пам'ять, а також автоматично виділяє пам'ять для параметрів функції виведення більшу частину часу. Отже, якщо функція має один або більше вхідних масивів (екземпляри *cv*::*Mat*) і кілька вихідних масивів, вихідні масиви автоматично розподіляються або перерозподіляються. Розмір і тип вихідних масивів визначаються розміром і типом вхідних масивів. Якщо потрібно, функції приймають додаткові параметри, які допомагають визначити властивості вихідного масиву.

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;
int main(int, char**)
{
   VideoCapture cap(0);
   if(!cap.isOpened()) return -1;
```

```
    Mat frame, edges;
    namedWindow("edges",1);
    for(;;)
    {
      cap >> frame;
      cvtColor(frame, edges, COLOR_BGR2GRAY);
      GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
      Canny(edges, edges, 0, 30, 3);
      imshow("edges", edges);
      if(waitKey(30) >= 0) break;
    }
    return 0;
}
```

Кадр масиву автоматично виділяється оператором >>, оскільки роздільна здатність відеокадру та бітова глибина відомі модулю захоплення відео. Краї масиву автоматично виділяються функцією cvtColor. Він має той самий розмір і бітову глибину, що й вхідний масив. Кількість каналів дорівнює 1, оскільки передається код перетворення кольору *COLOR_BGR2GRAY*, що означає перетворення кольору в градації сірого. Зауважте, що кадр і краї виділяються лише один раз під час першого виконання тіла циклу, оскільки всі наступні відеокадри мають однакову роздільну здатність. Якщо якось змінити роздільну здатність відео, масиви автоматично перерозподіляються.

Ключовим компонентом цієї технології є метод *Mat*::*create*. Він приймає бажаний розмір і тип масиву. Якщо масив уже має вказаний розмір і тип, метод нічого не робить. В іншому випадку він звільняє раніше виділені дані, якщо такі є (ця частина передбачає зменшення лічильника посилань і порівняння його з нулем), а потім виділяє новий буфер необхідного розміру. Більшість функцій

викликає метод *Mat*::*create* для кожного вихідного масиву, тому реалізується автоматичний розподіл вихідних даних.

Деякими помітними винятками з цієї схеми є *cv*::*mixChannels* , *cv*::*RNG*::*fill* і кілька інших функцій і методів. Вони не можуть виділити вихідний масив, тому ви повинні зробити це заздалегідь.

2.2.5 Арифметика насичення

Як бібліотека комп'ютерного зору, *OpenCV* багато має справу з пікселями зображення, які часто кодуються в компактній формі, 8- або 16-біт на канал, і тому мають обмежений діапазон значень. Крім того, певні операції над зображеннями, як-от перетворення колірного простору, налаштування яскравості/контрастності, підвищення різкості, складна інтерполяція (бі-кубік, *Lanczos*), можуть давати значення поза доступним діапазоном. Якщо ви просто збережете наймолодші 8 (16) біт результату, це призведе до візуальних артефактів і може вплинути на подальший аналіз зображення. Для вирішення цієї задачі використовується так звана арифметика насичення . Наприклад, щоб зберегти r, результат операції, у 8-бітне зображення, ви знайдете найближче значення в діапазоні 0..255:

$$I(x,y)=min(max(round(r),0),255)$$

Подібні правила застосовуються до 8-бітних знакових, 16-бітних знакових і беззнакових типів. Ця семантика використовується всюди в бібліотеці. У коді *C++* це робиться за допомогою функцій saturate_*cast*<>, які нагадують стандартні операції приведення *C++*. Дивіться нижче реалізацію наведеної вище формули:

```
I.at<uchar>(y, x) = saturate_cast<uchar>(r);
```

де *cv*::*uchar* — 8-розрядний тип цілого числа без знаку *OpenCV*. В оптимізованому коді *SIMD* використовуються такі інструкції *SSE2*, як *paddusb*,

packuswb і так далі. Вони допомагають досягти точно такої ж поведінки, як і в коді *C++*.

Насиченість не застосовується, якщо результат є 32-розрядним цілим числом.

2.2.6 Фіксовані типи пікселів. Обмежене використання шаблонів

Шаблони – це чудова функція C++, яка дозволяє реалізувати дуже потужні, ефективні та водночас безпечні структури даних і алгоритми. Однак широке використання шаблонів може значно збільшити час компіляції та розмір коду. Крім того, важко розділити інтерфейс і реалізацію, коли використовуються виключно шаблони. Це може бути добре для базових алгоритмів, але не добре для бібліотек комп'ютерного зору, де один алгоритм може охоплювати тисячі рядків коду. Через це, а також для спрощення розробки прив'язок для інших мов, таких як *Python, Java, Matlab,* які взагалі не мають шаблонів або мають обмежені можливості шаблонів, поточна реалізація *OpenCV* базується на поліморфізмі та диспетчеризації часу виконання через шаблони. У тих місцях, де диспетчеризація під час виконання буде надто повільною (наприклад, оператори доступу до пікселів), неможливою (загальна реалізація *Ptr<>*) або просто дуже незручною (*saturate_cast<>*()), поточна реалізація вводить невеликі шаблонні класи, методи та функції. Скрізь у поточній версії *OpenCV* використання шаблонів обмежене.

Отже, існує обмежений фіксований набір простих типів даних, з якими бібліотека може працювати. Тобто елементи масиву повинні мати один із наступних типів:

– 8-розрядне ціле число без знака (*uchar*);

– 8-розрядне ціле число зі знаком (*schar*);

– 16-розрядне ціле число без знаку (*ushort*);

– 16-розрядне ціле число зі знаком (*short*);

– 32-розрядне ціле число зі знаком (*int*);

– 32-розрядне число з плаваючою комою (*float*);

– 64-розрядне число з плаваючою комою (*double*);

– кортеж з кількох елементів, де всі елементи мають однаковий тип (один із зазначених вище). Масив, елементами якого є такі кортежі, називають багатоканальними масивами, на відміну від одноканальних масивів, елементами яких є скалярні значення. Максимально можлива кількість каналів визначається константою *CV_CN_MAX*, яка на даний момент має значення 512.

Для цих основних типів застосовується наступний перелік:

```
enum { CV_8U = 0 , CV_8S = 1 , CV_16U = 2 , CV_16S = 3 , CV_32S = 4 , CV_32F = 5 , CV_64F = 6};
```

Багатоканальні (n-канальні) типи можна вказати за допомогою таких параметрів:

– *CV_8UC1* … *CV_64FC4* константи (для кількості каналів від 1 до 4);

– *CV_8UC(n)* ... *CV_64FC(n)* або *CV_MAKETYPE*(*CV_8U*, *n*) … *CV_MAKETYPE*(*CV_64F*, *n*) макроси, коли кількість каналів більше 4 або невідома під час компіляції.

Приклади:

```
Mat mtx(3, 3, CV_32F); // make a 3x3 floating-point matrix
Mat cmtx(10, 1, CV_64FC2); // make a 10x1 2-channel floating-point
                           // matrix (10-element complex vector)
Mat img(Size(1920, 1080), CV_8UC3); // make a 3-channel (color) image
                                    // of 1920 columns and 1080 rows.
Mat grayscale(image.size(), CV_MAKETYPE(image.depth(), 1)); // make a 1-channel image of
                                                            // the same size and same
                                                            // channel type as img
```

Масиви зі складнішими елементами неможливо створити або обробити за допомогою OpenCV. Крім того, кожна функція або метод може обробляти лише підмножину всіх можливих типів масивів. Зазвичай, чим складніший алгоритм, тим менша підмножина підтримуваних форматів. Нижче наведено типові приклади таких обмежень:

– алгоритм розпізнавання обличчя працює лише з 8-бітними зображеннями у градаціях сірого або кольоровими зображеннями;

– функції лінійної алгебри та більшість алгоритмів машинного навчання працюють лише з масивами з плаваючою комою;

– базові функції, такі як cv::add , підтримують усі типи;

– функції перетворення простору кольорів підтримують 8-розрядні беззнакові, 16-розрядні беззнакові та 32-розрядні типи з плаваючою комою.

Підмножина підтримуваних типів для кожної функції була визначена з практичних потреб і може бути розширена в майбутньому на основі запитів користувачів.

2.2.7 *InputArray* і *OutputArray*

Багато функцій *OpenCV* обробляють щільні двовимірні чи багатовимірні числові масиви. Зазвичай такі функції приймають *cppMat* як параметри, але в деяких випадках зручніше використовувати *std::vector*<> (наприклад, для набору точок) або *Matx*<> (для гомографічної матриці 3x3 тощо). Щоб уникнути багатьох дублікатів в *API*, були введені спеціальні «проксі» класи. Базовим класом «проксі» є *InputArray*. Він використовується для передачі масивів лише для читання на вхід функції. Похідний від *InputArray* клас *OutputArray* використовується для визначення вихідного масиву для функції. Зазвичай ви не повинні піклуватися про ці проміжні типи (і ви не повинні оголошувати змінні цих типів явно) - все це буде працювати автоматично. Ви можете припустити, що замість InputArray/OutputArray ви завжди можете використовувати *Mat*, *std::vector*< >, *Matx*< >, *Vec*< > або *Scalar*. Якщо функція має додатковий вхідний або вихідний масив, а ви його не маєте або не хочете, передайте *cv::noArray*() .

2.2.8 Обробка помилок

OpenCV використовує винятки для сигналізації про критичні помилки. Якщо вхідні дані мають правильний формат і належать до вказаного діапазону значень, але алгоритм не може бути успішним з певної причини (наприклад, алгоритм оптимізації не сходиться), він повертає спеціальний код помилки (зазвичай, просто логічну змінну) .

Винятками можуть бути екземпляри класу *cv*::*Exception* або його похідні. У свою чергу, *cv*::*Exception* є похідною від *std*::*exception*. Таким чином, це може бути витончено оброблено в коді за допомогою інших стандартних компонентів бібліотеки *C++*.

Виняток зазвичай створюється або за допомогою макросу *CV_Error*(*errcode, description*), або його printf-подібного варіанту *CV_Error_*(*errcode, printf-spec,* (*printf-args*)), або за допомогою макросу *CV_Assert*(*condition*), який перевіряє умову та генерує виняток, коли він не задоволений. Для критично важливого для продуктивності коду існує *CV_DbgAssert*(умова) , який зберігається лише в конфігурації *Debug*. Завдяки автоматичному управлінню пам'яттю всі проміжні буфери автоматично звільняються у разі раптової помилки. Вам потрібно лише додати оператор *try* для перехоплення винятків, якщо потрібно: :

```
try
{
   ... // call OpenCV
}
catch( cv::Exception& e )
{
   const char* err_msg = e.what();
   std::cout << "exception caught: " << err_msg << std::endl;
}
```

2.3 Робота з зображенням та відео

2.3.1 Завантаження зображення

Для читання та відображення зображення в *OpenCV* потрібні такі функції *imread*(), *WaitKey*() та *imshow*().

imread() – ця функція використовується для читання зображень і приймає такі 2 аргументи:

– *filename*: повна адреса зображення, яке потрібно завантажити, має тип string. Наприклад: «*C:\users\downloads\sample.jpg*»

– *flag*: це необов'язковий аргумент, який визначає режим читання зображення та може приймати кілька значень:

1) *IMREAD_COLOR*: режим за замовчуванням, у якому завантажується зображення, якщо не надано аргументів. Він завантажує зображення у форматі *BGR*.
2) *IMREAD_UNCHANGED*: завантажує зображення в оригінальній формі. Він також включає альфа-канал, якщо він присутній усередині зображення.
3) *IMREAD_GRAYSCALE*: завантажує зображення як зображення у відтінках сірого.

Вихід: повертає зображення, як об'єкт типу *Mat*.

imshow(): ця функція використовується для відображення зображень і приймає такі два аргументи:

– *winname* або *window name*: це заголовок вікна, у якому відображається зображення, тип значення string.

– *image*: це зображення, яке буде показано. Його тип – *Mat* , контейнер зображень *C++*.

Результат: показується окреме вікно із зображенням.

Mat::*empty*(): це допомагає нам у обробці помилок, якщо функція *imread*() не може завантажити зображення або зображення не існує за вказаним шляхом, і повідомляє нам, чи контейнер *Mat* порожній чи ні.

WaitKey(): ця функція допомагає відображати зображення протягом більш тривалого часу, утримуючи вікно відкритим, доки користувач не натисне клавішу.

2.3.2 Відеовхід із *OpenCV* і вимірювання подібності

Відео можуть бути двох типів: зображення в реальному часі (у випадку веб-камери) або попередньо записані та збережені на жорсткому диску файли.

По суті, усі функціональні можливості, необхідні для обробки відео, інтегровані в клас *cv*::*VideoCapture C++*. Це саме по собі базується на бібліотеці з відкритим кодом *FFmpeg*. Це базова залежність *OpenCV*. Відео складається з послідовності зображень, які називаються кадрами. У випадку відеофайлу існує частота кадрів, яка визначає проміжок часу між двома кадрами. Хоча для відеокамер зазвичай існує обмеження кількості кадрів, які вони можуть оцифрувати за секунду, ця властивість менш важлива, оскільки в будь-який час камера бачить поточний знімок світу.

Перше завдання, яке вам потрібно зробити, це призначити класу *cv*::*VideoCapture* його джерело. Це можна зробити за допомогою функції *cv*::*VideoCapture*::*VideoCapture* або її функції *cv*::*VideoCapture*::*open* . Якщо цей аргумент є цілим числом, тоді ви прив’яжете клас до камери, пристрою. Переданий тут номер є ідентифікатором пристрою, призначеним операційною системою. Якщо у вас є одна камера, підключена до вашої системи, її ідентифікатор, ймовірно, дорівнюватиме нулю, а інші будуть збільшуватися. Якщо параметр, переданий до цих, є рядком, він посилатиметься на відеофайл, а рядок вказуватиме на розташування та назву файлу. Нариклад

```
VideoCapture captRefrnc(2);
// or
VideoCapture captUndTst;
captUndTst.open(“C:\Test.avi ”);
```

Щоб перевірити, чи прив’язка класу до джерела відео була успішною чи ні, використовуйте функцію *cv::VideoCapture::isOpened* :

```
if ( !captRefrnc.isOpened())
 {
 cout  << "Could not open reference " << sourceReference << endl;
 return -1;
 }
```

Закриття відео відбувається автоматично під час виклику деструктора об’єктів. Однак, якщо ви хочете закрити його до цього, вам потрібно викликати його функцію *cv::VideoCapture::release*. Кадри відео – це лише прості зображення. Тому нам просто потрібно витягти їх з об’єкта *cv::VideoCapture* і помістити в *Mat* . Відеопотоки є послідовними. Ви можете отримати кадри один за одним за допомогою *cv::VideoCapture::read* або перевантаженого оператора >>:

```
Mat frameReference, frameUnderTest;
captRefrnc >> frameReference;
captUndTst.open(frameUnderTest);
```

Верхні операції читання повернуть порожні’ об’єкти *Mat* , якщо не вдалося отримати кадр (через те, що відеопотік було закрито, або ви дійшли до кінця відеофайлу). Ми можемо перевірити це простим способом, якщо:

```
if (frameReference . empty () )
{
 // вихід з програми
}
```

Метод читання складається із захоплення кадру та застосованого до нього декодування. Ви можете явно викликати ці дві функції, використовуючи функції *cv::VideoCapture::grab* , а потім *cv::VideoCapture::retrieve* .

Окрім вмісту кадрів, до відео додається багато-багато інформації. Зазвичай це числа, однак у деяких випадках це можуть бути короткі послідовності символів (4 байти або менше). Завдяки цьому для отримання цієї інформації існує загальна функція під назвою *cv::VideoCapture::get* , яка повертає подвійні значення, що містять ці властивості. Використовуйте порозрядні операції для декодування символів типу double і перетворень, де дійсними значеннями є лише цілі числа. Його єдиним аргументом є ідентифікатор запитуваної властивості. Наприклад, тут ми отримуємо розмір кадрів відеофайлі та плюс кількість кадрів усередині посилання.

```
Size refS = Size((int) captRefrnc.get(CAP_PROP_FRAME_WIDTH),
                 (int) captRefrnc.get(CAP_PROP_FRAME_HEIGHT)),

cout << "Reference frame resolution: Width=" << refS.width << "  Height=" << refS.height
     << " of nr#: " << captRefrnc.get(CAP_PROP_FRAME_COUNT) << endl;
```

Коли ви працюєте з відео, ви часто можете самостійно контролювати ці значення. Для цього існує функція *cv::VideoCapture::set* . Його перший аргумент залишається назвою властивості, яку потрібно змінити, а другий аргумент типу *double* містить значення, яке потрібно встановити. Він поверне *true*, якщо це вдасться, і *false* в іншому випадку. Гарними прикладами цього є пошук у відеофайлі заданого часу або кадру:

```
captRefrnc.set(CAP_PROP_POS_MSEC, 1.2);  // go to the 1.2 second
captRefrnc.set(CAP_PROP_POS_FRAMES, 10); // go to the 10th frame
```

Для простих відеовиходів ви можете використовувати вбудований клас *OpenCV cv::VideoWriter* , призначений для цього.

Для початку ви повинні мати уявлення про те, як виглядає відеофайл. Кожен відеофайл сам по собі є контейнером. Тип контейнера виражається розширенням файлу (наприклад, *avi, mov* або *mkv*). Він містить кілька елементів, як-от: канали відео, канали аудіо або інші доріжки (наприклад, субтитри). Спосіб зберігання цих каналів визначається кодеком, який використовується для кожного з них. Для звукових доріжок зазвичай використовуються кодеки *mp*3 або *aac* . Для відеофайлів список дещо довший і включає такі назви, як *XVID* , *DIVX* , *H264* або *LAGS* (*Lagarith Lossless Codec*). Повний список кодеків, які ви можете використовувати в системі, залежить від того, який кодек ви встановили.

Однак *OpenCV* – це в основному бібліотека комп'ютерного зору, а не відеопотік, кодек і запис. Тому розробники постаралися зробити цю частину максимально простою. Через це *OpenCV* для відеоконтейнерів підтримує лише розширення avi, його першу версію. Прямим обмеженням цього є те, що ви не можете зберегти відеофайл розміром більше 2 ГБ. Крім того, ви можете створювати та розгортати лише одну відеодоріжку всередині контейнера. Тут немає підтримки аудіо чи інших доріжок. Тим не менш, будь-який відеокодек, наявний у вашій системі, може працювати. Якщо ви зіткнетеся з деякими з цих обмежень, вам доведеться шукати більш спеціалізовані бібліотеки для запису відео, такі як *FFMpeg* або кодеки, такі як *HuffYUV* , *CorePNG* і *LCL*.

Щоб створити відеофайл, вам просто потрібно створити екземпляр класу *cv::VideoWriter* . Ви можете вказати його властивості за допомогою параметрів у конструкторі або пізніше за допомогою функції *cv::VideoWriter::open* . У будь-якому випадку параметри однакові: 1. Ім'я результату, який містить тип контейнера у своєму розширенні. На даний момент підтримується тільки *avi* . Ми створюємо це з вхідного файлу, додаємо до нього ім'я каналу для використання та завершуємо це розширенням контейнера.

```
const string source      = "file_name";          // the source file name
```

```
string::size_type pAt = source.find_last_of('.');   // Find extension point
const string NAME = source.substr(0, pAt) + argv[2][0] + ".avi";    // Form the new name with container
```

Параметри які додатково додаються до файлу є:

1) кодек для відеодоріжки. Тепер усі відеокодеки мають унікальні короткі назви, що містять максимум чотири символи. Ви також можете запитати це у вхідного відео за допомогою функції отримання. Оскільки функція get є загальною функцією, вона завжди повертає подвійні значення. Подвійне значення зберігається на 64 бітах. Чотири символи - це чотири байти, тобто 32 біти. Ці чотири символи закодовані в молодших 32 бітах double . Простим способом викинути старші 32 біти було б просто перетворити це значення на int:

```
VideoCapture inputVideo(source);                    // Open input
int ex = static_cast<int>(inputVideo.get(CAP_PROP_FOURCC));     // Get Codec Type- Int form
```

OpenCV внутрішньо працює з цим цілим типом і очікує його як другий параметр. Тепер для перетворення з цілочисельної форми в рядкову ми можемо використовувати два методи: побітовий оператор і метод об’єднання. Перший, який витягує символи з int, виглядає так (операція «і», деякий зсув і додавання 0 у кінці, щоб закрити рядок):

```
char  EXT [] = { ex  &  0 XFF  , ( ex  &  0 XFF00 ) >> 8 , ( ex  &  0 XFF0000 ) >> 16 , ( ex  &  0 XFF000000 ) >> 24 ,  0 };
```

Ви можете зробити те саме з об’єднанням , як:

```
union { int v; char c[5];} uEx ;
```

```
uEx.v = ex;                    // From Int to char via union
uEx.c[4]='\0';
```

Перевагою цього є те, що перетворення виконується автоматично після призначення, тоді як для побітового оператора вам потрібно виконувати операції щоразу, коли ви змінюєте тип кодека. Якщо ви заздалегідь знаєте чотирисимвольний код кодеків, ви можете використати макрос *CV_FOURCC* для створення цілого числа:

```
CV_FOURCC( 'P' , 'I' , 'M,' 1 ') // це кодек MPEG1 від символів до цілого числа
```

Якщо ви передасте цьому аргументу мінус один, під час виконання з’явиться вікно, яке містить усі кодеки, встановлені у вашій системі, і попросить вас вибрати той, який використовуватиметься (рис. 2.3):

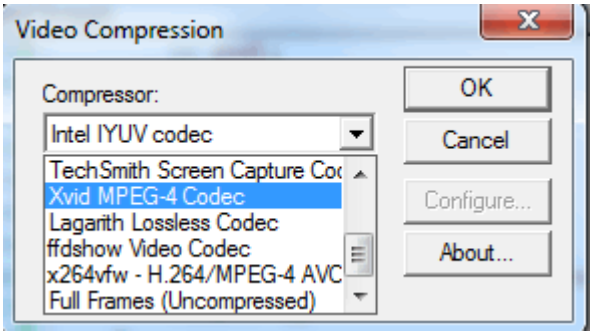


Рисунок 2.3 – Вікно відображення вибору кодеку

2) Кількість кадрів за секунду для вихідного відео. Знову ж таки, тут я зберігаю кадри в секунду вхідного відео за допомогою функції *get* .

3) Розмір кадрів для вихідного відео. Тут також я зберігаю розмір кадрів вхідного відео за секунду за допомогою функції *get* .

4) Остаточний аргумент необов’язковий. За замовчуванням має значення true і означає, що вихідні дані будуть кольоровими (тому для запису ви надішлете

три зображення каналу). Щоб створити відео в градаціях сірого, введіть тут параметр *false*.

Приклад початку запису відео:

```
VideoWriter outputVideo;
Size S = Size((int) inputVideo.get(CAP_PROP_FRAME_WIDTH),   //Acquire input size
 (int) inputVideo.get(CAP_PROP_FRAME_HEIGHT));
outputVideo.open(NAME , ex, inputVideo.get(CAP_PROP_FPS),S, true);
```

Після цього ви використовуєте функцію *cv::VideoWriter::isOpened*() , щоб дізнатися, чи вдалося відкрити операцію чи ні. Відеофайл автоматично закривається, коли об'єкт *VideoWriter* знищується. Після успішного відкриття об'єкта ви можете надіслати кадри відео в послідовному порядку за допомогою функції *cv::VideoWriter::write* класу. Крім того, ви можете використовувати його перевантажений оператор <<:

```
outputVideo.write(res);  //or
outputVideo << res;
```

Висновки до розділу

OpenCV це відкрита бібліотека компю'терного зору, що базаово написана на мові програмування C++. Дана бібліотека маємодульну структуру, що відповідає за різні алгоритми обробки зображень. Головноим модулем є core, що забезпечує основний функціонал бібліотеки. В залежності від завантажених модулів система має змогу працювати не тільки з зображеннями, а з відео.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ МОЖЛИВОСТЕЙ

3.1 Використання контурів

Контури визначаються як лінія, що з'єднує всі точки вздовж межі зображення, які мають однакову інтенсивність. Контури стають у пригоді для аналізу форми, визначення розміру цікавого об'єкта та виявлення об'єкта.

У *OpenCV* пошук контурів подібний до пошуку білого об'єкта на чорному тлі. Тому пам'ятайте, об'єкт, який потрібно знайти, має бути білим, а фон – чорним.

Для виявлення контуру використовується функція *findContours*(). Дана функція приймає три аргументи : перший – вихідне зображення, другий – режим пошуку контуру, третій – метод апроксимації контуру. І він виводить контури та ієрархію. Контури – це список усіх контурів на зображенні. Кожен окремий контур є масивом *Numpy* (*x*,*y*) координат граничних точок об'єкта.

Контури - це межі форми з однаковою інтенсивністю. Він зберігає (*x*,*y*) координати межі форми. Але чи зберігає він усі координати це визначається цим методом контурної апроксимації.

*Я*кщо ви передаєте *cv.CHAIN_APPROX_NONE* , усі граничні точки зберігаються. Ні, нам потрібні лише дві кінцеві точки цієї лінії. Ось що робить *cv.CHAIN_APPROX_SIMPLE* . Він видаляє всі зайві точки і стискає контур, тим самим економлячи пам'ять.

Для малювання контурів використовується функція *cv.drawContours* . Його також можна використовувати для малювання будь-якої фігури, якщо у вас є її граничні точки. Його першим аргументом є вихідне зображення, другим аргументом є контури, третім аргументом є індекс контурів (корисно під час малювання окремого контуру. Щоб намалювати всі контури, передайте -1), а інші аргументи – колір, товщина тощо. Наприклад:

```
cv.drawContours (img, contours, -1, (0,255,0), 3)
```

Крім знаходження контуру є додаткові можливості, це:

– моменти зображення допомагають обчислити деякі характеристики, такі як центр маси об’єкта, площа об’єкта тощо. Функція *cv.moments*();

– площа контуру задається функцією cv.contourArea() або з моментів, *M*['*m*00'];

– контурний периметр її також називають довжиною дуги. Це можна дізнатися за допомогою функції *cv.arcLength*() . Другий аргумент визначає, чи є фігура замкнутим контуром (якщо задано *True*), чи просто кривою;

– контурна апроксимація. Він наближає форму контуру до іншої форми з меншою кількістю вершин залежно від точності, яку ми вказуємо. Це реалізація алгоритму Дугласа-Пюкера . Функція *cv.approxPolyDP* ();

– обмежувальний прямокутник. Існує два типи обмежувальних прямокутників: прямий обмежувальний прямокутник (*cv.boundingRect*()) та повернутий прямокутник *cv.boxPoints*();

– мінімальне охоплююче коло. Ми знаходимо описане коло об’єкта за допомогою функції *cv.minEnclosingCircle*() . Це коло, яке повністю покриває об'єкт з мінімальною площею.

Приклад використання представлено в лістингу 3.1:

Лістинг 3.1

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;
Mat src_gray;
```

```
int thresh = 100;
void thresh_callback(int, void* );
void thresh_callback(int, void* );
int main( int argc, char** argv )
{
   Mat src = imread( "nuos.jpg" );
   if( src.empty() )
   {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
   }
   cvtColor( src, src_gray, COLOR_BGR2GRAY );
   blur( src_gray, src_gray, Size(5,5) );
   const char* source_window = "Source";
   namedWindow( source_window );
   imshow( source_window, src );
   const int max_thresh = 255;
   createTrackbar( "Canny thresh:", source_window, &thresh, max_thresh,
thresh_callback );
   thresh_callback( 0, 0 );
   waitKey();
   return 0;
}
void thresh_callback(int, void* )
{
   Mat canny_output;
   Canny( src_gray, canny_output, thresh, thresh*2 );
```

```
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

findContours( canny_output, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE );
Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );

for( size_t i = 0; i< contours.size(); i++ )
{
   Scalar color = Scalar( 255, 255, 255 );
   drawContours( drawing, contours, (int)i, color, 2, LINE_8, hierarchy, 0 );
}

imshow( "Contours", drawing );
}
```

Результат роботи пердставлено на рисунку 3.1.

а) б)

Рисунок 3.1 – Результат пошуку контуру

3.2 Сегментація зображення за допомогою алгоритму вододілу

Сегментація зображення — це фундаментальне завдання комп’ютерного зору , яке передбачає поділ зображення на значущі та семантично однорідні області. Мета полягає в тому, щоб спростити представлення зображення або зробити його більш значущим для подальшого аналізу. Ці сегменти зазвичай відповідають об’єктам або областям інтересу на зображенні.

Алгоритм вододілу — це класична техніка сегментації зображення, яка базується на концепції перетворення вододілу. Процес сегментації розглядатиме подібність із сусідніми пікселями зображення як важливий орієнтир для з’єднання пікселів із подібними просторовими положеннями та значеннями сірого.

Алгоритм вододілу використовується під час сегментації зображень із об’єктами, що торкаються або перекриваються. Він чудово підходить у сценаріях із неправильною формою об’єктів, вимогами до сегментації на основі градієнта та коли можлива сегментація за допомогою маркерів.

Алгоритм вододілу ділить зображення на сегменти за допомогою топографічної інформації. Він розглядає зображення як топографічну поверхню, ідентифікуючи водозбірні басейни на основі інтенсивності пікселів. Місцеві мінімуми позначаються як вихідні точки, а заливка кольорами заповнює водозбірні басейни до досягнення меж об’єкта. Отримана сегментація призначає унікальні кольори регіонам, сприяючи розпізнаванню об’єктів і аналізу зображень.

Весь процес алгоритму вододілу можна підсумувати в наступних кроках:

Розміщення маркерів: Першим кроком є розміщення маркерів на локальних мінімумах або найнижчих точках зображення. Ці маркери служать відправними точками для процесу затоплення.

Заливка: потім алгоритм заливає зображення різними кольорами, починаючи з маркерів. Коли колір поширюється, він заповнює водозбірні басейни, поки не досягне меж об’єктів або областей на зображенні.

Формування водозбірного басейну : у міру поширення кольору водозбірні басейни поступово заповнюються, створюючи сегментацію зображення. Отриманим сегментам або областям призначаються унікальні кольори, які потім можна використовувати для ідентифікації різних об’єктів або особливостей на зображенні.

Ідентифікація меж: алгоритм вододілу використовує межі між різними кольоровими областями для ідентифікації об’єктів або областей на зображенні. Отриману сегментацію можна використовувати для завдань розпізнавання об’єктів, аналізу зображень і виділення ознак.

Приклад роботи:

– підключення бібліотек та завантаження зображення (рис. 3.2);

```
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>

using namespace std;
using namespace cv;

int main(int argc, char *argv[])
{
  Mat src = imread( "coin.jpg" );
  if( src.empty() )
  {
    cout << "Could not open or find the image!\n" << endl;
    return -1;
  }
```

```
imshow("Source Image", src);
```

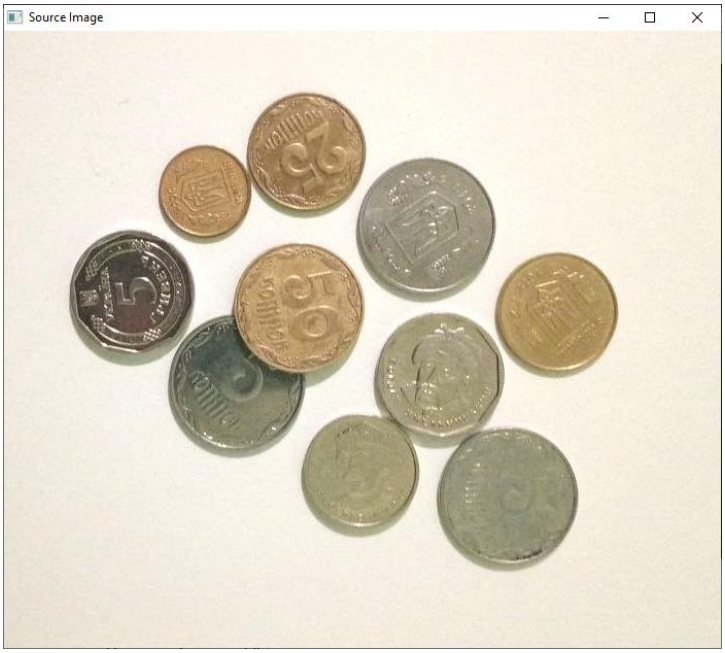


Рисунок 3.2 – Вхідне зображення

– після цього ми підвищимо різкість нашого зображення, щоб загострити краї об’єктів переднього плану. Ми застосуємо фільтр Лапласа з досить сильним фільтром (наближення другої похідної) (рис. 3.3):

```
Mat kernel = (Mat_<float>(3,3) <<
                1,  1, 1,
                1, -8, 1,
                1,  1, 1); // an approximation of second derivative, a quite strong kernel

Mat imgLaplacian;
filter2D(src, imgLaplacian, CV_32F, kernel);
Mat sharp;
src.convertTo(sharp, CV_32F);
```

```
Mat imgResult = sharp - imgLaplacian;

// convert back to 8bits gray scale
imgResult.convertTo(imgResult, CV_8UC3);
imgLaplacian.convertTo(imgLaplacian, CV_8UC3);

imshow( "New Sharped Image", imgResult );
```

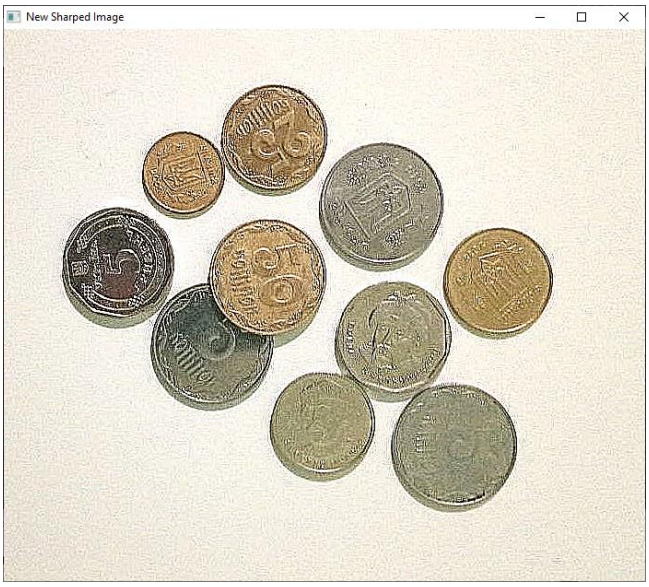


Рисунок 3.3 – Зображення підвищеної різкості

– тепер ми перетворюємо наше нове вихідне зображення з різкістю на градації сірого та бінарне відповідно рис. 3.4 та додаємо невелике розмиття:

```
Mat bw;
   cvtColor(imgResult, bw, COLOR_BGR2GRAY);
   threshold(bw, bw, 40, 255, THRESH_BINARY_INV | THRESH_OTSU);
   blur(bw, bw, Size(5 ,5));
   imshow("Binary Image", bw);
```

Рисунок 3.4 – Отримане бінарне зображення

– тепер ми готові застосувати перетворення відстані до бінарного зображення. Крім того, ми нормалізуємо вихідне зображення, щоб мати можливість візуалізувати та визначити порогове значення результату (рис. 3.5):

```
Mat dist;
distanceTransform(bw, dist, DIST_L2, 3);
normalize(dist, dist, 0, 1.0, NORM_MINMAX);
imshow("Distance Transform Image", dist);
```

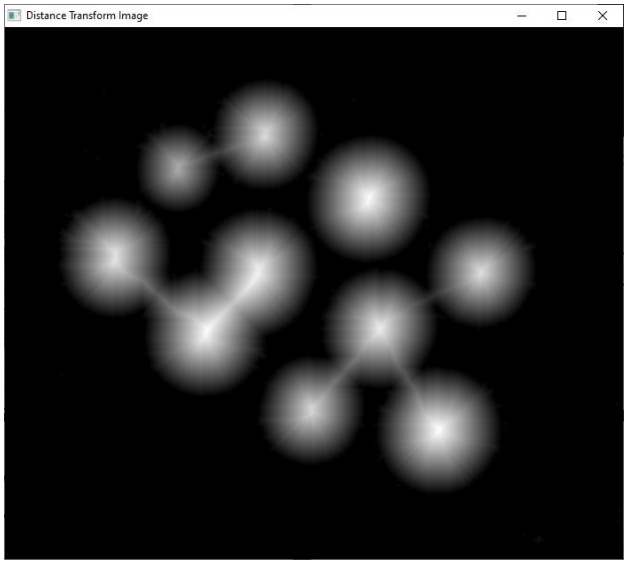


Рисунок 3.5 – Нормалізоване бінарне зображення

– ми обмежуємо порогове значення дист -зображення, а потім виконуємо певну морфологічну операцію (тобто розширення), щоб витягнути піки із зображення вище (рис. 3.6):

```
threshold(dist, dist, 0.4, 1.0, THRESH_BINARY);
   Mat kernel1 = Mat::ones(3, 3, CV_8U);
   dilate(dist, dist, kernel1);
   imshow("Peaks", dist);
```

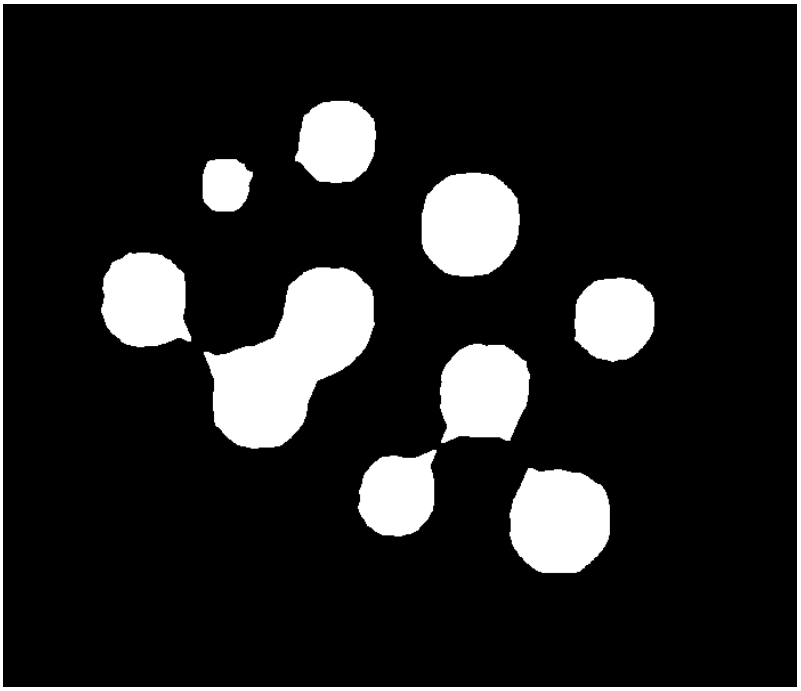


Рисунок 3.6 – Пікові області зображення

– потім з кожної області ми створюємо вихідний елемент/маркер для алгоритму вододілу за допомогою функції *cv::findContours* (рис. 3.7):

```
Mat dist_8u;
   dist.convertTo(dist_8u, CV_8U);
   vector<vector<Point> > contours;
   findContours(dist_8u, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

```
Mat markers = Mat::zeros(dist.size(), CV_32S);
for (size_t i = 0; i < contours.size(); i++)
{
   drawContours(markers, contours, static_cast<int>(i),
Scalar(static_cast<int>(i)+1), -1);
}
circle(markers, Point(5,5), 3, Scalar(255), -1);
Mat markers8u;
markers.convertTo(markers8u, CV_8U, 10);
imshow("Markers", markers8u);
```

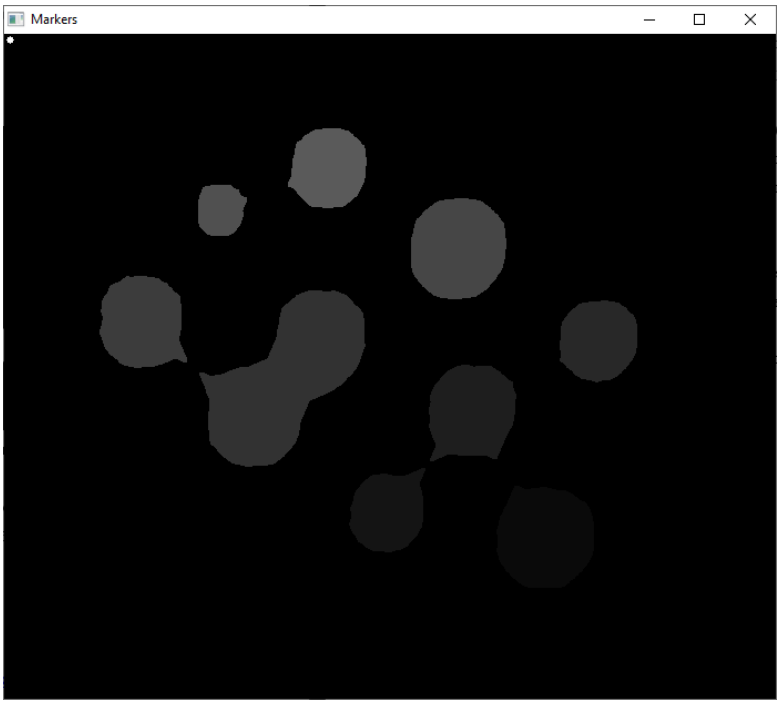


Рисунок 3.7 – Створення маркерів

– ми застосовуємо алгоритм вододілу та виводимо результат (рис. 3.8):

```
watershed(imgResult, markers);
Mat mark;
markers.convertTo(mark, CV_8U);
```

```
bitwise_not(mark, mark);
vector<Vec3b> colors;
for (size_t i = 0; i < contours.size(); i++)
{
   int b = theRNG().uniform(0, 256);
   int g = theRNG().uniform(0, 256);
   int r = theRNG().uniform(0, 256);

   colors.push_back(Vec3b((uchar)b, (uchar)g, (uchar)r));
}
// Create the result image
Mat dst = Mat::zeros(markers.size(), CV_8UC3);

// Fill labeled objects with random colors

for (int i = 0; i < markers.rows; i++)
{
   for (int j = 0; j < markers.cols; j++)
   {
      int index = markers.at<int>(i,j);
      if (index > 0 && index <= static_cast<int>(contours.size()))
      {
         dst.at<Vec3b>(i,j) = colors[index-1];
      }
   }
}
imshow("Result", dst);
```

Рисунок 3.8 – Виявленні об’єкти

– співставляємо з вхідним зображенням та завершуємо програму (рис 3.9):

```
Mat imgBin;
   Mat src2 = imread( "coin.jpg" );
   imgBin = src2 | dst;
   imshow("Final Result", imgBin);
   waitKey();
   return 0;
}
```

Результат виявлення об’єктів в нашому випадку ускладнюється наступними факторами:

– неоднородність об’єктів;
– різний колір монет;
– навність дотику та накладання монет одна на одну;

Результати можна покращити шляхом виявлення об’єктів окремо по кольоровому признаку при накладанні відповідної маски.

Рисунок 3.9 – Фінальний результат сегментації

3.3 Виявлення об’єктів за допомогою узагальненого перетворення Балларда та Гіла Хафа

Узагальнене перетворення Хафа, представлене Даною Х. Баллард у 1981 році, є модифікацією перетворення Хафа з використанням принципу відповідності шаблонів. Перетворення Хафа спочатку було розроблено для виявлення аналітично визначених форм (наприклад, лінії , кола , еліпса тощо). У цих випадках ми маємо знання про форму та прагнемо з’ясувати її розташування та орієнтацію на зображенні. Ця модифікація дозволяє використовувати перетворення Хафа для виявлення довільного об’єкта, описаного його моделлю.

Задачу пошуку об’єкта (описаного моделлю) на зображенні можна вирішити, знайшовши положення моделі на зображенні. За допомогою узагальненого перетворення Хафа задача знаходження положення моделі перетворюється на задачу знаходження параметра перетворення, який

відображає модель на зображенні. За значенням параметра перетворення можна визначити положення моделі на зображенні.

Оригінальна реалізація алгоритму використовувала інформацію про границі для визначення відображення від орієнтації крайової точки до опорної точки фігури. У випадку бінарного зображення, де пікселі можуть бути або чорними, або білими, кожен чорний піксель зображення може бути чорним пікселем бажаного шаблону, таким чином створюючи локус опорних точок у просторі Хафа. Кожен піксель зображення відповідає за відповідні опорні точки. Максимальні точки простору Хафа вказують на можливі опорні точки візерунка на зображенні. Цей максимум можна знайти, скануючи простір Хафа або розв'язуючи розслаблений набір рівнянь, кожне з яких відповідає чорному пікселю.

Щоб узагальнити алгоритм Хафа на неаналітичні криві, Баллард визначає наступні параметри для узагальненої форми: $a=\{y,s,\theta\}$, де y — вихідна точка форми, θ — її орієнтація, а $s = (s_x, s_y)$ описує два ортогональні масштабні коефіцієнти. Алгоритм може обчислити найкращий набір параметрів для заданої форми на основі піксельних даних країв. Ці параметри не мають рівноправного статусу. Розташування опорного джерела, y , описується в термінах шаблонної таблиці, яка називається *R*-таблицею можливих орієнтацій крайових пікселів. Обчислення додаткових параметрів s і θ потім виконується шляхом прямого перетворення цієї таблиці. Ключовим узагальненням для довільних форм є використання інформації про напрямок. За будь-якої форми та фіксованої контрольної точки на ній замість параметричної кривої інформація, що надається граничними пікселями, зберігається у формі *R*-таблиці на етапі перетворення. Для кожної крайової точки на тестовому зображенні властивості точки шукаються в *R*-таблиці, отримується опорна точка та збільшується відповідна клітинка в матриці, яка називається матрицею накопичувача. Комірка з максимальною кількістю «голосів» у матриці накопичувача може бути можливою точкою існування фіксованого посилання об'єкта на тестовому зображенні.

Побудова *R*-таблиці.

Виберіть опорну точку у для фігури (зазвичай вибирається всередині фігури). Для кожної граничної точки x обчисліть ϕ(*x*) , напрям градієнта та $r = y - x$. Зберігайте r як функцію ϕ (табл. 3.1). Зауважте, що кожен індекс ϕ може мати багато значень r . Можна зберігати різницю координат між фіксованою опорною точкою та крайовою точкою (($x_c - x_{ij}$), ($y_c - y_{ij}$)) або як радіальну відстань і кут між ними (r_{ij} , α_{ij}) . Зробивши це для кожної точки, *R*-таблиця буде повністю представляти об’єкт шаблону. Крім того, оскільки фаза генерації є оборотною, ми можемо використовувати її для локалізації входжень об’єктів в інших місцях зображення.

Таблиця 3.1 – Представлення шаблонної таблиці

i	ϕ_i	$R_{\phi i}$
1	0	$(r_{11}, \alpha_{11})(r_{12}, \alpha_{12})...(r_{1n}, \alpha_{1n})$
2	$\Delta\phi$	$(r_{21}, \alpha_{21})(r_{22}, \alpha_{22})...(r_{2m}, \alpha_{2m})$
3	$2\Delta\phi$	$(r_{31}, \alpha_{31})(r_{32}, \alpha_{32})...(r_{3k}, \alpha_{3k})$
...	...	...

Локалізація об'єкта. Для кожного крайового пікселя *x* на зображенні знайдіть градієнт ϕ і збільште всі відповідні точки $x+r$ у накопичувальному масиві *A* (ініціалізований до максимального розміру зображення), де r — запис таблиці з індексом ϕ , тобто $r\ (\phi)$. Ці точки входу дають нам кожну можливу позицію для контрольної точки. Незважаючи на те, що деякі фіктивні точки можуть бути обчислені, враховуючи, що об’єкт існує на зображенні, максимум буде мати місце в контрольній точці. Максимуми в *A* відповідають можливим екземплярам форми.

Узагальнення масштабу та орієнтації. Для фіксованої орієнтації форми накопичувальний масив був двовимірним у координатах опорної точки. Для

пошуку форм довільної орієнтації θ та масштабу s ці два параметри додаються до опису форми. Накопичувальний масив тепер складається з чотирьох вимірів, що відповідають параметрам (y, s, θ) . R-таблиця також може бути використана для збільшення цього більшого розмірного простору, оскільки різні орієнтації та масштаби відповідають легко обчислюваним перетворенням таблиці. Позначимо конкретну R-таблицю для форми S через $R(\phi)$. Прості трансформації цієї таблиці дозволять їй виявляти масштабовані або повернуті екземпляри однакової форми. Наприклад, якщо фігуру масштабовано на s і це перетворення позначено T_s . тоді $T_s[R(\phi)] = sR(\phi)$, тобто всі вектори масштабуються на s . Крім того, якщо об'єкт повертається на θ і це перетворення позначається T_θ , тоді $T_\theta[R(\phi)] = Rot\{R[(\phi-\theta) mod 2\pi], \theta\}$, тобто всі індекси збільшуються на $-\theta$ по модулю 2π, знаходять відповідні вектори r , а потім повертають їх на θ . Ще одна властивість, яка буде корисною при описі композиції узагальнених перетворень Хафа, це зміна опорної точки. Якщо ми хочемо вибрати нову опорну точку $\tilde{y}$ так, що $y-\tilde{y} = z$, тоді модифікація R-таблиці задається $R(\phi)+ z$, тобто z додається до кожного вектора в таблиці.

Альтернативний спосіб з використанням пар країв. Пара крайових пікселів може бути використана для зменшення простору параметрів. Використовуючи R-таблицю та описані вище властивості, кожен крайовий піксель визначає поверхню в чотиривимірному накопичувальному просторі $a = (y, s, \theta)$. Два краєві пікселі в різних орієнтаціях описують ту саму поверхню, повернуту на однакову величину відносно θ . Якщо ці дві поверхні перетинаються, точки, де вони перетинаються, відповідатимуть можливим параметрам a для форми. Таким чином, теоретично можливо використовувати дві точки в просторі зображень, щоб зменшити геометричне місце в просторі параметрів до однієї точки. Однак труднощі пошуку точок перетину двох поверхонь у просторі параметрів зроблять цей підхід неможливим для більшості випадків.

Композитні форми. Якщо фігура S має складну структуру, що складається з частин S_1 , S_2 , .. S_N , а опорними точками для форм S , S_1 , S_2 ,

.. $S_N \in y, y_1, y_2, \ldots y_n$, відповідно, тоді для коефіцієнта масштабування s орієнтаці θ узагальнене перетворення Хафа $R_s(\phi)$ визначається як

$$R_\phi = T_s \left\{ T_\theta \left[\bigcup_{k=1}^{N} R_{s_k}(\phi) \right] \right\}.$$

Проблема з цим перетворенням полягає в тому, що вибір посилання може сильно вплинути на точність. Щоб подолати це, Баллард запропонував згладжувати отриманий акумулятор за допомогою композитного шаблону згладжування. Складений шаблон згладжування $H(y)$ надається як складена згортка окремих шаблонів згладжування підформ.

$$H(y) = \sum_{i=1}^{N} h_i(y - y_i).$$

Тоді вдосконалений накопичувач задається як $A_s = A*H$, а максимуми в A_s відповідають можливим екземплярам форми.

Просторова декомпозиція. Зауваживши, що глобальне перетворення Хафа можна отримати підсумовуванням локальних перетворень Хафа непересічної підобласті, Хізер і Янг запропонували метод, який передбачає рекурсивний поділ зображення на підзображення, кожне зі своїм власним простором параметрів , і організований у структурі квадратного дерева . Це призводить до покращеної ефективності пошуку кінцевих точок сегментів лінії та покращеної стійкості та надійності при вилученні ліній у шумних ситуаціях за дещо збільшеної вартості пам’яті.

Приклад роботи в *OpenCV* побудовано на основі визначення положення на зображенні (рис. 3.10.а) об’єктів з шаблоном (рис. 3.10.б).

а) б)

Рисунок 3.10 – Вхідні зображення для експерименту

а) зображення де буде проводитися пошук;

б) зображення шаблону

Завантажте зображення, шаблон і змінні налаштування:

```
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
 using namespace cv;
using namespace std;
 int main() {
   Mat image = imread("key.jpg"));
   Mat templ = imread("keys.jpg", IMREAD_GRAYSCALE);

   Mat grayImage;
   cvtColor(image, grayImage, COLOR_RGB2GRAY);

   vector<Vec4f> positionBallard, positionGuil;
   int w = templ.cols;
   int h = templ.rows;
```

Параметри налаштування. Знаходження оптимальних значень може завершуватися методом проб і помилок і залежить від багатьох факторів, наприклад від роздільної здатності зображення:

```
//  create ballard and set options
   Ptr<GeneralizedHoughBallard> ballard = createGeneralizedHoughBallard();
   ballard->setMinDist(10);
   ballard->setLevels(360);
   ballard->setDp(2);
   ballard->setMaxBufferSize(1000);
   ballard->setVotesThreshold(40);
    ballard->setCannyLowThresh(30);
   ballard->setCannyHighThresh(110);
   ballard->setTemplate(templ);

//  create guil and set options
   Ptr<GeneralizedHoughGuil> guil = createGeneralizedHoughGuil();
   guil->setMinDist(10);
   guil->setLevels(360);
   guil->setDp(3);
   guil->setMaxBufferSize(1000);
    guil->setMinAngle(0);
   guil->setMaxAngle(360);
   guil->setAngleStep(1);
   guil->setAngleThresh(1500);
    guil->setMinScale(0.5);
   guil->setMaxScale(2.0);
   guil->setScaleStep(0.05);
```

```
guil->setScaleThresh(50);

guil->setPosThresh(10);
 guil->setCannyLowThresh(30);
guil->setCannyHighThresh(110);
 guil->setTemplate(templ);
```

Запуск системи виявлення:

```
//  execute ballard detection
   ballard->detect(grayImage, positionBallard);
//  execute guil detection
   guil->detect(grayImage, positionGuil);
```

Відображення результату (рис. 3.11):

```
//  draw ballard
   for (vector<Vec4f>::iterator iter = positionBallard.begin(); iter != positionBallard.end(); ++iter) {
      RotatedRect rRect = RotatedRect(Point2f((*iter)[0], (*iter)[1]),
                                      Size2f(w * (*iter)[2], h * (*iter)[2]),
                                      (*iter)[3]);
      Point2f vertices[4];
      rRect.points(vertices);
      for (int i = 0; i < 4; i++)
         line(image, vertices[i], vertices[(i + 1) % 4], Scalar(255, 0, 0), 6);
   }
```

```
// draw guil
    for (vector<Vec4f>::iterator iter = positionGuil.begin(); iter != positionGuil.end(); ++iter) {
        RotatedRect rRect = RotatedRect(Point2f((*iter)[0], (*iter)[1]),
                                        Size2f(w * (*iter)[2], h * (*iter)[2]),
                                        (*iter)[3]);
        Point2f vertices[4];
        rRect.points(vertices);
        for (int i = 0; i < 4; i++)
            line(image, vertices[i], vertices[(i + 1) % 4], Scalar(0, 255, 0), 2);
    }
    imshow("result_img", image);
    waitKey();
}
```

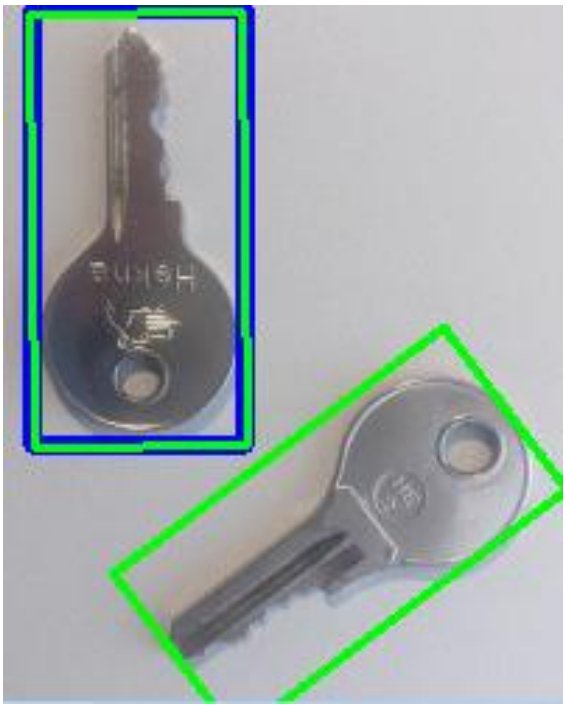


Рисунок 3.11 – Результат виявлення об’єктів

3.4 Розпізнавання обличчя за допомогою каскадів Хаара

Розпізнавання обличчя має велике значення в різних сферах сучасного світу. Це важливий крок у кількох додатках, розпізнаванні обличчя (також використовується як біометрія), фотографії (для автофокусування на обличчі), аналізі обличчя (вік, стать, розпізнавання емоцій), відеоспостереження тощо.

Одним із популярних алгоритмів визначення обличчя є «хааркаскад». Це менш дорогий обчислювальний алгоритм, швидкий алгоритм і висока точність.

Він працює в чотири етапи:

Вибір об’єкта Хаара : Об’єкт, подібний до Хаара, складається з темних і світлих областей. Він створює одне значення, беручи різницю суми інтенсивностей темних областей і суми інтенсивностей світлих областей. Це робиться для вилучення корисних елементів, необхідних для ідентифікації об'єкта. Віола та Джонс пропонують функції, що зображені на рисунку 3.12 :

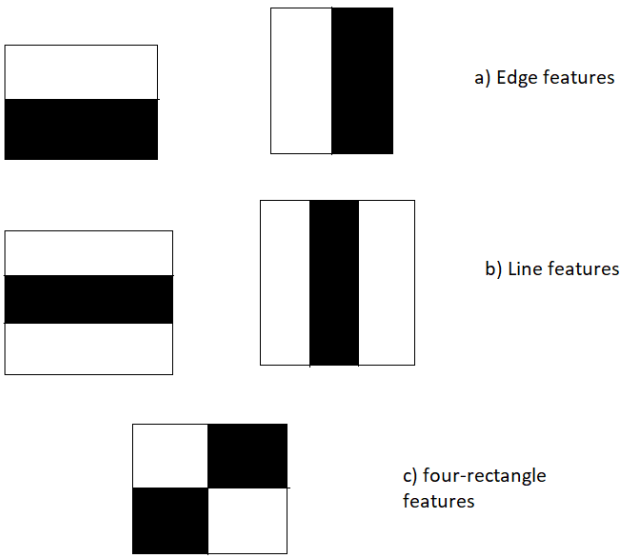


Рисунок 3.12 – Функції для ідентифікації об’єкта

Створення цілісних зображень : певний піксель цілісного зображення є сумою всіх пікселів ліворуч і всіх пікселів над ним. Оскільки процес виділення подібних до Хаара особливостей передбачає обчислення різниці між темними та світлими прямокутними областями, впровадження цілісного зображення (рис. 3.13) значно скорочує час, необхідний для виконання цього завдання.

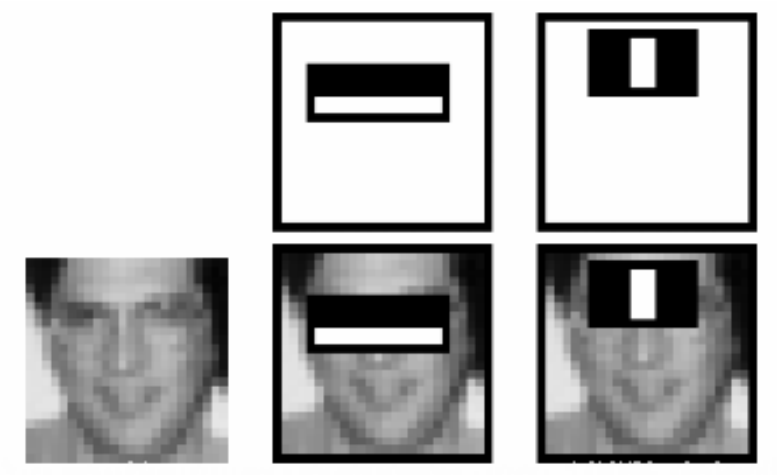


Рисунок 3.12 – Ствроення цілісних зображень

Навчання *AdaBoost* : цей алгоритм вибирає найкращі функції з усіх функцій. Він об’єднує кілька «слабких класифікаторів» (найкращі характеристики) в один «сильний класифікатор». Створений «сильний класифікатор» в основному є лінійною комбінацією всіх «слабких класифікаторів».

Каскадний класифікатор : це метод для об’єднання дедалі складніших класифікаторів, таких як *AdaBoost*, у каскад, який дозволяє швидко відкидати негативні дані (не обличчя), витрачаючи більше обчислень на багатообіцяючі або позитивні регіони, схожі на обличчя. Це значно скорочує час обчислень і робить процес більш ефективним.

OpenCV поставляється з великою кількістю попередньо навчених класифікаторів. Ці файли *XML* можна завантажити за допомогою методу *cascadeClassifier* модуля *cv*2. Тут ми збираємося використовувати haarcascade_frontalface_default.xml для виявлення облич.

Поетапне впровадження:

Крок 1: Завантаження зображення.

Крок 2: Перетворення зображення в градації сірого. Спочатку зображення є тришаровим (тобто *RGB*), тому воно перетворюється на одношарове зображення (тобто у градаціях сірого).

Крок 3: Завантаження необхідного файлу класифікатора *XML haar-cascade*. Метод *CascadeClassifier* у модулі *cv*2 підтримує завантаження *XML*-файлів *haar-cascade*. Тут нам потрібен «*haarcascade_frontalface_default.xml*» для визначення обличчя.

Крок 4: Застосування методу виявлення обличчя до зображення у градаціях сірого. Це робиться за допомогою методу *cv2::CascadeClassifier::detectMultiScale*, який повертає граничні прямокутники для виявлених облич (тобто *x, y, w, h*). Він приймає два параметри, а саме *scaleFactor і minNeighbors. ScaleFactor* визначає коефіцієнт збільшення розміру вікна, який спочатку починається з розміру «*minSize*», а після перевірки всіх вікон такого розміру вікно збільшується за допомогою «scaleFactor», і розмір вікна зростає до «*maxSize*». Якщо «*scaleFactor*» великий (наприклад, 2,0), буде менше кроків, тому виявлення буде швидшим, але ми можемо пропустити об’єкти, розмір яких знаходиться між двома перевіреними масштабами. (масштабний коефіцієнт за замовчуванням 1,3). Чим вище значення «minNeighbors», тим менше буде кількість хибних спрацьовувань, і менше помилок буде з точки зору хибного виявлення облич. Однак є ймовірність пропустити деякі нечіткі сліди обличчя.

Крок 5: Перегляд прямокутників виявлених облич

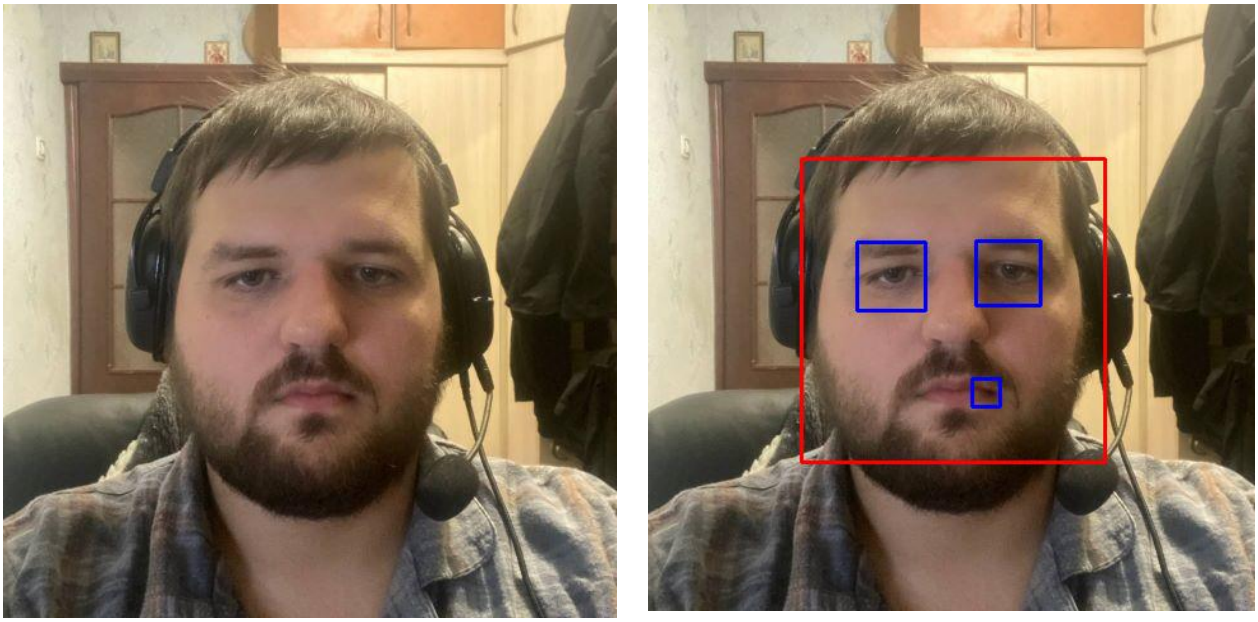
Прямокутники малюються навколо виявлених облич за допомогою методу прямокутника модуля *cv*2 шляхом повторення всіх виявлених облич.

Нижче приведений код реалізації та результати виявлення обличчя та очей (рис. 3.14). На рисунку червоним виділено знайдений контур обличчя а синім виділено очі.

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

int main( int argc, char** argv )
{
let src = cv.imread('canvasInput');
let gray = new cv.Mat();
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
let faces = new cv.RectVector();
let eyes = new cv.RectVector();
let faceCascade = new cv.CascadeClassifier();
let eyeCascade = new cv.CascadeClassifier();
// load pre-trained classifiers
faceCascade.load('haarcascade_frontalface_default.xml');
eyeCascade.load('haarcascade_eye.xml');
// detect faces
let msize = new cv.Size(0, 0);
faceCascade.detectMultiScale(gray, faces, 1.1, 3, 0, msize, msize);
for (let i = 0; i < faces.size(); ++i) {
   let roiGray = gray.roi(faces.get(i));
   let roiSrc = src.roi(faces.get(i));
   let point1 = new cv.Point(faces.get(i).x, faces.get(i).y);
   let point2 = new cv.Point(faces.get(i).x + faces.get(i).width,
                             faces.get(i).y + faces.get(i).height);
   cv.rectangle(src, point1, point2, [255, 0, 0, 255]);
   // detect eyes in face ROI
```

```
    eyeCascade.detectMultiScale(roiGray, eyes);
    for (let j = 0; j < eyes.size(); ++j) {
        let point1 = new cv.Point(eyes.get(j).x, eyes.get(j).y);
        let point2 = new cv.Point(eyes.get(j).x + eyes.get(j).width,
                                  eyes.get(j).y + eyes.get(j).height);
        cv.rectangle(roiSrc, point1, point2, [0, 0, 255, 255]);
    }
    roiGray.delete(); roiSrc.delete();
}
cv.imshow('canvasOutput', src);
src.delete(); gray.delete(); faceCascade.delete();
eyeCascade.delete(); faces.delete(); eyes.delete();
}
```



а) б)

Рисунок 3.14 – Приклад виявлення обличчя

а) вхідне зображення; б) результат роботи програми

Помилки роботи визначені великим розширенням вхідного зображення

Висновки до розділу

Реалізована демострація можливостей *OpenCV* надає базу для складних алгоритмів систем розпізнавання обраців. Алгоритми виділення контуру широко використовуються в системах кластеризації зображень для виявлення положень об’єктів. Системи сегментації та виявлення об’єктів мають широкий спектр застосування в системах компютерного зору для виявлення підозрілих об’єктів, а система виявлення обличчя є першим етапом для алгоритмів розпізнавання облиць, що використовуються в охоронних системах.

.

ВИСНОВОК

У ході виконання кваліфікаційної роботи було проведено аналіз сфери застосувань цифрової обробки зображень та її основні можливості. Представленні можливості використання показують що бібліотека *OpenCV* має широке застосування, як в охоронних системах так і в системах моніторингу.

В рамках кваліфікаційної роботи було представлено реалізації можливостей до яких відноситься виділення контуру, пошуку об’єкта на зображенні, сегментація зображення та виявлення обличчя на фото.

Робота, що має практичне значення для розвитку навчального процесу кафедри в напрямку комп’ютерного зору, який може використовуватися як в наукових цілях так і в рамках основних та вибіркових дисциплін.

ПЕРЕЛІК ПОСИЛАНЬ

1. Творошенко І. С. Конспект лекцій з дисципліни «Цифрова обробка зображень» Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків :ХНУМГ ім. О. М. Бекетова, 2015. – 75 с.

2. A. A. *Kassim, T. Tan, K. H. Tan, "A comparative study of efficient generalised Hough transform techniques", Image and Vision Computing, Volume 17, Issue 10, Pages 737-748, August 1999.*

3. *David Lowe, Object recognition from local scale-invariant features, in: International Conference on Computer Vision, Vol.2, 1999.*

4. *R.Szeliski, Computer Vision: Algorithms and Applications. URL: http://szeliski.org/Book/ .*

5. *James F Peters Foundations of Computer Vision. Publisher: Springer International Publishing Switzerland. 2017. 431 p. ISBN 978-3-319-30260-7, ISBN 978-3-319-30262-1 (eBook) DOI: 10.1007/978-3-319-52483-2 4.*

6. *David Marr Vision: A computational investigation into the human representation and processing of visual information. The MIT Press. 2010. URL: https://direct.mit.edu/books/monograph/3299/VisionA-Computational-Investigation-into-the-Human .*

7. *Face detection using Cascade Classifier using OpenCV-Python. URL: https://www.geeksforgeeks.org/face-detection-using-cascade-classifier-using-opencv-python/.*

8. *Moving Object Detection with OpenCV using Contour Detection and Background Subtraction. URL: https://learnopencv.com/moving-object-detection-with-opencv/.*

9. *Object Detection with OpenCV URL: https://www.scaler.com/topics/ opencv-object-detection/.*

10. *Reading and Displaying an image in OpenCV using C++. URL: https://www.geeksforgeeks.org/reading-and-displaying-an-image-in-opencv-using-c/*

11. *Gloria Bueno García Oscar Deniz Suarez at all Learning Image Processing with OpenCV. Packt Publishing 2015. 232 p.*

12. *The OpenCV Tutorials Release 2.4.8.0. URL: https://www.academia.edu/8827002/The_OpenCV_Tutorials.*

13. *Orešković, Marko. (2009). Implementation of computer vision using OpenCV library. URL: https://www.researchgate.net/publication/312042036_Implementation_of_computer_vision_using_OpenCV_library.*

14. *OpenCV Documentation. Introduction. URL: ttps://vovkos.github.io /doxyrest-showcase/opencv/sphinx_rtd_theme/index.html#doxid-d1-dfb-intro.*

15. *Joseph Howse, Joe Minichino Learning OpenCV 4 Computer Vision with Python 3, 3rd Edition 2020 372p.*

16. *Generalised Hough transform URL: https://en.wikipedia.org/wiki/ Generalised_Hough_transform.*

17. *What is OpenCV? A Guide for Beginners. URL: https://blog.roboflow.com/what-is-opencv/.*

18. *OpenCV – understanding the IplImage data type. URL: https://atharvai.wordpress.com/2013/03/31/opencv-understanding-the-iplimage-data-t/.*

19. *OpenCV 4.0.0 new Graph API (G-API) URL: https://blog.conan.io/ 2018/12/19/New-OpenCV-release-4-0.html*

20. *OpenCV | Saving an Image. URL: https://www.geeksforgeeks.org/opencv-saving-an-image/*

21. *What is OpenCV Library?. URL: https://www.geeksforgeeks.org/opencv-overview/*

22. *Open Source Computer Vision: веб сайт URL: https://docs.opencv.org/ 4.x/index.html*

23. *G. Bradski and A. Kaehler, Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media, 2008.*

24. *Chetana Patil , Vaishnavi Andhalkar, Janhavi Dandavate, Pranjal Pokharkar, Suresh Lora A Survey: Fire Detection Alarm System Using OpenCV.*

International Journal for Research in Applied Science & Engineering Technology. Volume 11, Nov 2023- Available at www.ijraset.com

25. *Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In Computer vision-eccv 2004, pages 469–481. Springer, 2004.*

26. *Cuneyt Akinlar and Cihan Topal. Edlines: A real-time line segment detector with a false detection control. Pattern Recognition Letters, 32(13):1633–1642, 2011.*

27. *Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pages 510–517. IEEE, 2012.*

28. *Josef Bigun. Vision with direction. Springer, 2006.*

29. *John Canny. A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (6):679–698, 1986.*

30. *Christoph Zauner. Implementation and benchmarking of perceptual image hash functions. 2010.*

31. *Simon AJ Winder and Matthew Brown. Learning local image descriptors. In Computer Vision and Pattern Recognition, pages 1–8, 2007.*

32. *Gunilla Borgefors. Distance transformations in digital images. Computer vision, graphics, and image processing, 34(3):344–371, 1986.*

33. *Getting Started with OpenCV URL: https://learnopencv.com/getting-started-with-opencv/*

34. *MATLAB OpenCV Interface URL: https://www.mathworks.com/ discovery/matlab-opencv.html*

35. *Image Processing Using OpenCV – With Practical Examples URL: https://www.analyticsvidhya.com/blog/2021/05/image-processing-using-opencv-with-practical-examples/*