Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ
В. о. завідувач кафедри,
к.т.н., доц.
_____ Я. М. Крайник
« __ » _____ 2022 р.

БАКАЛАВРСЬКА РОБОТА

Галузь знань: 12 Інформаційні технології

Спеціальність: 123 Комп'ютерна інженерія

Тема: **Програмно-апаратний комплекс розпізнавання райдужної оболонки на базі Raspberry Pi**

Шифр: 123 – ПП – 405.21810517

Виконав:
студент 4 курсу, групи 405,
спеціальності
123 Комп'ютерна інженерія
_____ Д.А. Мельниченко

Керівник: ст. викл.

_____ І. С. Бурлаченко

Миколаїв 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп’ютерних наук
Кафедра комп’ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри,
канд. техн. наук, доцент
_____ Я. М. Крайник
« __ » _____ 2022 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Видано студенту групи 405 факультету комп’ютерних наук

Мельниченко Данило Андрыйович

(прізвище, ім’я, по батькові студента)

1. Тема кваліфікаційної роботи: Програмно-апаратний комплекс розпізнавання райдужної оболонки на базі Raspberry Pi._____

Затверджена наказом по ЧНУ від «__» _____ 2022 р. No _____

2. Строк представлення кваліфікаційної роботи «____» _____ 2022 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Прототип має робити фотографію, оперувати інформацією отриманою з датчиків, для подальшої нормалізації та сегментації зображення, та ідентифіковувати особу.

_____

___

4. Перелік питань, що підлягають розробці

Аналіз апаратних платформ. Аналіз алгоритмів та методів нормалізації зображення, сегментації та збереження даних. Моделювання прототипу розпізнавання райдужної оболонки, макетної та принципової схем. Збирання прототипу. Налаштування апаратної

платформи. Розробка програмного забезпечення приладу. Спеціальна частина з охорони праці. Висновки. Перелік джерел посилання._____

5. Перелік графічних матеріалів

Зображення будови ока. Зображення системи розпізнавання райдужної оболонки ока. Зображення апаратних аналогів та таблиць їх характеристик. Зображення моделей алгоритмів Даугмана. Зображення схем порівняння достовірного зображення.

6. Завдання до спеціальної частини

Розглянути загальні положення щодо норм та законів України, щодо захисту праці та організації робочого простору комп’ютерного інженера. Дослідити правила пожежної безпеки у приміщення з ЕОМ. Вивчити ергономічні вимоги щодо робочого місця інженера-розробника.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи	
ст. викладач А. О. Алексєєва	кафедра екології Медичного інституту ЧНУ ім. Петра Могили	спеціальна частина з охорони праці	

Керівник роботи ___ст. викл. І. С. Бурлаченко_____

(посада, прізвище, ім’я, по батькові)

(підпис)

Завдання прийнято до виконання Мельниченко Данило Андрійович_____

(прізвище, ім’я, по батькові студента)

(підпис)

Дата видачі завдання «____» _____ 20_____ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Програмно-апаратний комплекс розпізнавання райдужної оболонки на базі Raspberry Pi

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КР	20.12.2021	05.01.2022	Виконано
2	Огляд літератури за темою роботи	10.01.2022	01.02.2022	Виконано
3	Складання календарного плану КР	03.02.2022	14.02.2022	Виконано
4	Аналіз предметної області	15.02.2022	23.02.2022	Виконано
5	Розробка проектних рішень	24.01.2022	10.03.2022	Виконано
6	Моделювання та конструювання прототипу	24.02.2022	15.03.2022	Виконано
7	Перевірка працездатності, тестування розробленого прототипу, аналіз результатів тестування	16.03.2022	25.03.2022	Виконано
8	Відгук керівника КР	26.05.2022	01.06.2022	Виконано
9	Оформлення КР та презентації	02.06.2022	09.06.2022	Виконано
10	Попередній захист	10.06.2022	11.06.2022	Виконано
11	Рецензування	14.06.2022	18.06.2022	Виконано
12	Завершення оформлення КР та презентації	18.06.2022	25.06.2022	Виконано
13	Захист кваліфікаційної роботи	27.06.2022	28.06.2022	Виконано

Розробив студент_____Мельниченко Данило Андрійович_____

(прізвище, ім’я, по батькові)

(підпис)

«__» _____ 2022 р.

Керівник роботи _____ст. викл. І. С. Бурлаченко_____

(посада, прізвище, ім’я, по батькові)

(підпис)

«__» _____2022 р.

АНОТАЦІЯ

бакалаврської роботи

«Програмно-апаратний комплекс розпізнавання райдужної оболонки на базі Raspberry Pi»

Студент: Мельниченко Данило Андрійович

Керівник: ст. викл. Бурлаченко І. С.

Бакалаврська робота присвячена розробці апаратно-програмного комплексу розпізнавання райдужної оболонки ока на базі Raspberry Pi. Розглянуто методи та алгоритми локалізації, нормалізації та сегментації зображення райдужної оболонки ока. Практичне значення результатів дослідження та розроблення полягає в тому що, прототип може бути використаний в різних прикладних програмах, де потрібно підтвердити особистість людини.

Пояснювальна записка бакалаврської роботи складається зі вступу, трьох розділів, висновків, одного додатку та спеціальної частини з охорони праці та безпеки життєдіяльності . У вступі визначається актуальність теми, сформульовані мета, об’єкт, предмет та завдання дослідження та розроблення бакалаврської роботи. У першому розділі досліджуються загальна архітектура систем розпізнавання, яка може бути використана у прототипі. У другому розділі наведені методи локалізації, сегментації та нормалізації зображення які будуть використовуватися у прототипі, та налаштування апаратної частини системи. У третьому розділі наведені дані про програмну частину прототипу та всі етапи обробки зображення. У висновках наведено аналіз виконаної роботи та отриманих результатів дослідження та розроблення.

У додатку А наведено програмний код, що використовувався в проекті. В цілому, бакалаврська робота без додатків містить 52 сторінки, 30 рисунків, 3 таблиці, 28 джерел посилання.

Ключові слова: локалізація, сегментація, нормалізація, алгоритм Даугмана, відстань Хеммінга, Raspberry Pi.

ABSTRACT

of the Bachelor’s Thesis

"Software and hardware complex of iris recognition based on Raspberry Pi "

Student: Melnychenko Danylo

Consultant: Senior Lecturer Ivan Burlachenko

The bachelor's thesis is devoted to the development of hardware and software complex for iris recognition based on Raspberry Pi. Methods and algorithms of localization, normalization and segmentation of the image of the iris were considered. The practical significance of the results of research and development is that the prototype can be used in various applications where you want to confirm a person's identity.

The explanatory note of the bachelor's thesis consists of an introduction, three sections, conclusions, one appendix and a special part on labor protection and life safety. The introduction determines the relevance of the topic, formulates the purpose, object, subject and objectives of research and development of the bachelor's thesis. The first section examines the general architecture of recognition systems that can be used in the prototype, and provides an analysis of existing analogues of the prototype. The second section presents the methods of localization, segmentation and normalization of the image to be used in the prototype, and the configuration of the hardware. The third section provides data on the software part of the prototype and all stages of image processing. The conclusions provide an analysis of the work performed and the results of research and development.

Appendix A lists the program code used in the project. In total, the bachelor's thesis without appendices contains 52 pages, 30 figures, 3 tables, 28 reference sources.

Keywords: localization, segmentation, normalization, Daugman algorithm, Hamming distance, Raspberry Pi.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FFT	–	Fast Fourier Transform – Швидке перетворення Фур'є
HID	–	Human Interface Device – пристрій взаємодії з людиною
ID	–	Identity Document – ідентифікатор
IEEE	–	Institute of Electrical and Electronics Engineers – інститут інженерів з електротехніки та електроніки
INC	–	Incorporated – інкорпорація
SDK	–	Software Development Kit – комплект для розробки програмного забезпечення
ОС	–	Operating System – операційна система
ДНК	–	Дезоксирибонуклеїнова кислота
ІЧ	–	Інфрачервоне світло
ОАЕ	–	Об'єднані Арабські Емірати
США	–	Сполучені Штати Америки

ВСТУП

В сучасному світі кожного дня зростає чисельність різноманітної інформації, програм, сайтів, інтернет гаманців, і т.д.. Деяка з них є приватною, захищеною від загального використання, обмеженою в доступі або повинна стати такою. Існує явна потреба в посилених заходах безпеки для багатьох програм.

Традиційні, прості методи безпеки, такі як пін-коди або паролі, стають дедалі більш вразливими. Фактори безпеки, засновані на володінні, такі як ключі, були завжди схильні до зловмисних крадіжок. Все більш популярним рішенням безпеки є використання біометричних даних і багатофакторна аутентифікація. Біометричні фактори, як правило, не піддаються крадіжкам і їх не так легко скомпрометувати. Недоліки будь-якої однієї аутентифікації можна мінімізувати, комбінуючи кілька факторів аутентифікації.

Райдужна оболонка ока людини часто описується як одна з найбільш підходящих ознак для біометричної аутентифікації через кілька бажаних характеристик:

- Райдужка має складну текстуру;
- Є унікальним для кожної людини;
- Як внутрішній орган він добре захищений від травм і досить стабільний протягом тривалого часу на відміну від інших біометричних ознак, які часто відчувають значні вікові зміни, наприклад, обличчя.

Незважаючи на безліч переваг райдужної оболонки як біометричної риса, вона все ще не використовується на практиці так широко, як відбитки пальців або зображення обличчя.

Мета: дослідження методів обробки зображень райдужної оболонки ока, розробка апаратно-програмного комплексу пристроїв для розпізнавання райдужної оболонки для визначення ідентичності людини через порівняння подібності між ознаками зображення райдужної оболонки.

Об’єкт: – методи, засоби та технології систем розпізнавання райдужної оболонки.

Предмет: система розпізнавання райдужної оболонки ока на базі апаратної платформи Raspberry PI та алгоритму Даугмана.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- З аналітичного огляду джерел інформації обґрунтувати методи, засоби моделей отримання образу райдужної оболонки найбільш оптимальними;
- Отримати образ райдужної оболонки та нормалізувати зображення райдужної оболонки;
- Розробити схеми, діаграми класів, блок-схеми алгоритмів автоматичної автентифікації;
- Розробити питання охорони праці та безпеки життєдіяльності.

Практичне значення: використання біометрії в різних прикладних програмах, де потрібно підтвердити особистість людини, чи розпізнати її, стало можливим сьогодні. Області застосування такі як: контроль пасажирів в аеропортах, контроль доступу в зонах обмеженого доступу, прикордонний контроль, доступ до баз даних і фінансових послуг, доступ до мобільних приладів чи програм в них є прикладами застосування біометричних технологій для більш надійної ідентифікації і перевірки.

Апробація результатів бакалаврської роботи відбулася під час:

XXIV Всеукр. наук.-практ. конф. Могилянські читання–2021 : досвід та тенденції розвитку суспільства в Україні : глобальний, національний та регіональний аспекти. 8–12 листоп. 2021 р., м. Миколаїв.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ ТА ПАТЕНТНОЇ ІНФОРМАЦІЇ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Що таке біометрична автентифікація в загалі? Це процес безпеки, який посилається на унікальні біологічні характеристики людей за для підтвердження того, що вони є тими, за кого себе видають. Такі системи можуть та порівнюють поведінкові ознаки чи фізичні зі збереженими та підтвердженими в базі даних. Якщо ці два зразки біометричних даних збігаються, автентифікація підтверджується. Дуже часто такий вид безпеки як біометрична автентифікація використовується для керування доступом до різних захищених цифрових або фізичних ресурсів.

Біометрична ідентифікація використовує біометричні дані, а саме сканування сітківки чи відбитки пальців, для ідентифікації особи, з іншої сторони біометрична автентифікація – використання біометричних даних для доказу чи підтвердження того, що люди є тими, за кого себе видають.

Такі технології можна використати для цифрової ідентифікації людей або надання їм дозволу на доступ до системи: хімічно-біометричні прилади, що узгоджують ДНК використовує генетичний матеріал для ідентифікації людини.

Візуальні біометричні прилади – сканування сітківки ока ідентифікує суб’єкти, аналізуючи малюнок кровоносних судин у задній частині очей. Розпізнавання райдужної оболонки ока використовує зображення райдужки для ідентифікації людей. Сканування відбитків пальців ідентифікує людей за відбитками пальців. Розпізнавання геометрії рук перевіряє ідентичність або авторизує транзакції за допомогою математичного представлення унікальних характеристик рук людей. Це робиться шляхом вимірювання відстані між різними частинами кисті, включаючи довжину пальців, ширину пальців і форму западин між кісточками пальців. Розпізнавання обличчя спирається на

унікальні характеристики та візерунки обличчя людей для підтвердження їхньої особистості. Система визначає 80 вузлових точок на обличчі людини, які утворюють числові коди, які називаються відбитками обличчя. Автентифікація вуха підтверджує особу на основі унікальної форми вуха користувача. Розпізнавання підпису використовує розпізнавання образів для ідентифікації осіб на основі їх рукописного підпису.

Сканери вен або судин – ідентифікатор вени пальця визначає людей на основі візерунка вен на їхньому пальці.

Поведінкові ідентифікатори – хода аналізує те, як люди ходять. Розпізнавання набору тексту встановлює ідентичність людей на основі їхніх унікальних характеристик друку, зокрема швидкості введення тексту.

Слухові біометричні прилади – voice ID ідентифікує людей за їх голосом і спирається на характеристики, створені формою рота та горла.

1.1.1 Зовнішня будова ока

Райдужка — це кругла кольорова структура, яка розташована перед кришталиком у корональній площині до передньої частини ока. Приклад зовнішньої будови ока показано на рис.1.1.

Для розуміння усієї цінності райдужної оболонки, треба розглянути її структуру більш детально. Незв’язана в середині, щоб дозволити зіниці змінювати розмір, ця структура з’єднана з циліарним тілом — частиною ока, яка виробляє очну рідину (водянисту вологи) і регулює скорочення і звуження райдужної оболонки.

Коли світло потрапляє в око, воно впливає на певні нервові клітини в задній частині очей. Ці клітини посилають сигнал у наш мозок, який потім повідомляє райдужній оболонці стискатися або розширюватися, залежно від того, скільки світла потрапляє всередину.

Колір райдужки визначається пігментом у клітинах райдужної оболонки, і цей пігмент може бути одним із кількох кольорів, включаючи коричневий, зелений, блакитний або горіховий. Щоб уточнити, малюнок

райдужної оболонки визначається формою та розташуванням гранул меланіну у вашій райдужній оболонці.

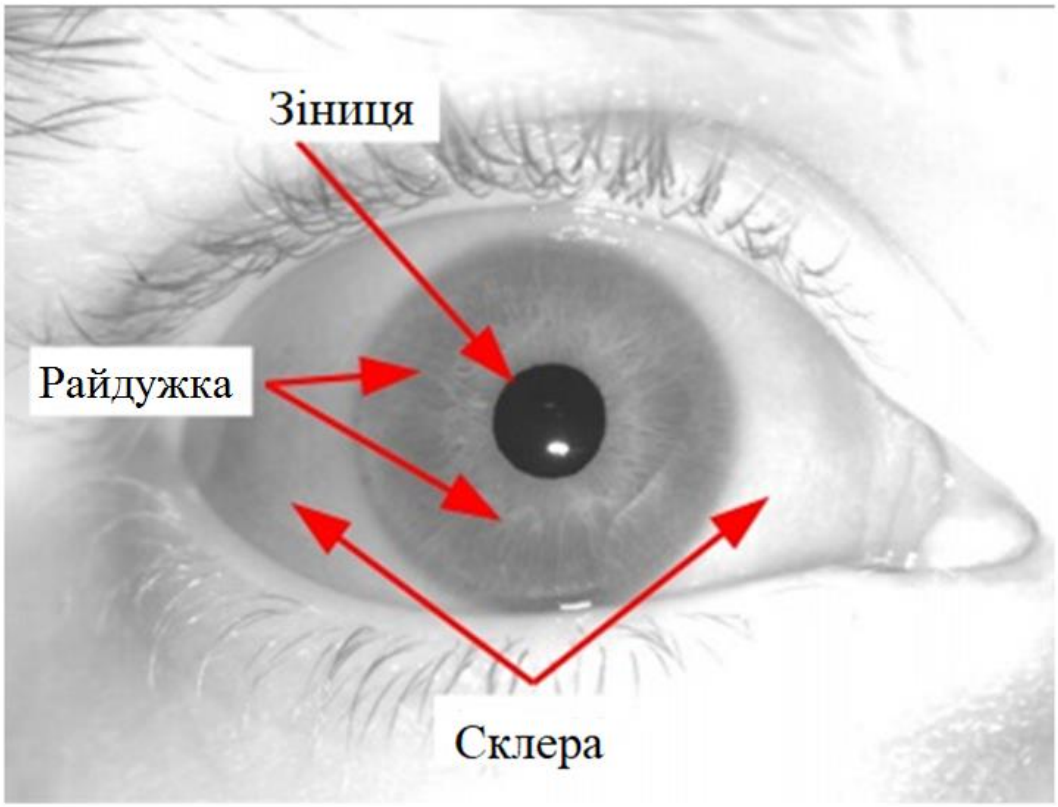


Рисунок 1.1 – Приклад зовнішньої будови ока

Фізіологічна складність цього органу призводить до випадкових закономірностей в малюнку райдужної оболонки, які є статистично унікальними, що дозволяє використовувати у біометричних вимірювань.

Як внутрішній орган він добре захищений від травм і досить стабільний протягом тривалого часу на відміну від інших біометричних ознак, які часто відчувають значні вікові зміни, наприклад, обличчя.

Склера — це частина ока, яка утворює опорну стінку очного яблука і суцільний з прозорою рогівкою. Склера покрита кон’юнктивою, прозорою слизовою оболонкою, яка допомагає змащувати око. Віна найбільш товста в області, що оточує зоровий нерв. Склера складається з трьох відділів: епісклери, пухкої сполучної тканини, безпосередньо під кон'юнктивою.

Власне склера , щільна біла тканина, яка надає області колір, внутрішня зона, що складається з еластичних волокон.

Також зіниця ока – чорне коло в центрі райдужної оболонки. Вона пропускає і регулює потік світла до сітківки. Це частина процесу, який дозволяє нам сприймати зображення. Зіниця відкривається і закривається, щоб контролювати кількість світла, яке може потрапити в око. Із зовнішнього боку ока світло проходить через прозору лінзу, потім через зіницю. Потім це світло фокусується на сітківці, яка є шаром світлочутливих клітин у задній частині ока.

1.2 Огляд досліджень

Ідею використання візерунків райдужної оболонки для ідентифікації особистості спочатку запропонував офтальмолог Френк Берч у 1936 році. У 1980-х ідея з’явилася у фільмах, але все ще залишалася науковою фантастикою та здогадами. У 1987 році два інших офтальмолога, Аран Сафір і Леонард Флом, запатентували цю ідею, а в 1989 році вони попросили Джона Даугмана (тоді викладав у Гарвардському університеті) створити реально застосовні алгоритми для розпізнавання райдужної оболонки ока.

Основна теоретична ідея в алгоритмах Даугмана полягає в тому, що провал тесту статистичної незалежності може бути дуже сильною основою для розпізнавання образів, якщо існує достатньо висока ентропія (достатня кількість ступенів свободи випадкових змін) серед вибірок з різних класів. У 1994 році він запатентував цю основу для розпізнавання райдужної оболонки ока та лежать в її основі алгоритми Computer Vision для обробки зображень, виділення ознак і зіставлення, а також опублікував статтю «Висока достовірність візуального розпізнавання осіб за допомогою тесту статистичної незалежності» в IEEE Transactions on Pattern Analysis і Машинний інтелект.

Аран Сафір і Леонард Флом поєднали сферу класичного розпізнавання образів із сучасним комп’ютерним зором, математичним статистика та

дослідження людино-машинного інтерфейсу. Це міждисциплінарна галузь. Патенти належать IridianTechnologies і є основою для всіх сучасних систем і продуктів розпізнавання райдужної оболонки ока. Ця система ідентифікації була використовувався в Об’єднаних Арабських Еміратах для контролю перетину кордону. Однак цю технологію зараз можна вважати глобальною, і кількість країн, які використовують її для визначення заявників на візи значно збільшилася і зараз включає Канаду, Нідерланди, Сполучені Штати Америки та інші. В США також використовують технологію розпізнавання по райдужці для ідентифікації усіх військовослужбовців Сполучених Штатів для підвищення безпеки та дозволу на доступ до військових об'єктів.

Загальна ефективність системи розпізнавання райдужної оболонки залежить від продуктивності її підсистем. Якість отриманих зображень, сегментація, нормалізація і витяг ознак, в цілому, визначають продуктивність системи.

Наприклад, Лім [17] повідомляє про досягнення успіху лише в 88,2% на більш ніж 6000 зображеннях райдужної оболонки на етапі попередньої обробки через небажані фактори, такі як закриття райдужки повіками, тінь від вій і шумів в області зіниць.

Таке погіршення етапу попередньої обробки можна розглядати як погіршення загальної продуктивності системи.

1.2.1 Загальна архітектура системи розпізнавання

Повна система розпізнавання райдужної оболонки ока складається з чотирьох частин:

- Отримання зображення;
- Райдужна локалізація;
- Нормалізація райдужної оболонки;
- Виділення ознак та відповідність.

Розпізнавання райдужної оболонки системи у вигляді двох плечей, як показано на рис. 1.2 Це – реєстраційне плече та плече верифікації, коли

шаблон зображення райдужної оболонки, що підлягає перевірці, порівнюється з шаблонами райдужної оболонки, що зберігаються в базі даних райдужки. Для отримання зображення, інфрачервоний знімок ока слід робити при освітленні інфрачервоним (ІЧ) світлом. Отримане зображення фіксує райдужну оболонку ока зображення. Крок локалізації райдужної оболонки локалізує область райдужки на зображенні. Райдужні кордони моделюються як два кола, які не обов'язково є концентричними. Внутрішнє коло є папілярною межею або внутрішньою межею райдужки. Зовнішнє коло - це лімбічна межа або зовнішня межа райдужки. Шум обробки часто включені в етап сегментації системи розпізнавання. Можливими джерелами шуму є оклюзії повік, вії оклюзії та дзеркальні відбиття. Більшість алгоритмів локалізації використовували градієнтні методи для того, щоб знайти краї між зіницею та райдужною оболонкою та склерою райдужки. Етап вилучення ознак кодує ознаки зображення райдужки бітовий векторний код, такий як відображений у рівнянні 1.2. Зручний пороговий рівень вибирається як граничне значення для функції кодування.

$$I_{x,y} = \{1, 0, \qquad (1.2)$$

У більшості алгоритмів для отримання інформації про текстуру райдужки використовуються фільтри. Потім виходи фільтрів кодуються у бітовий векторний код. Відповідний етап узгодження обчислює відстань між кодами райдужної оболонки і вирішує, чи відповідає вона, чи розпізнає подану райдужну оболонку ока від суб’єктів у наборі даних на основі порогового рівня прийняття рішення.

Збереження коду райдужної оболонки у базі даних не є безпечним, оскільки у разі викрадення коду, людина не зможе згенерувати інший на основі тієї ж самої райдужної оболонки. У зв’язку з цим до схеми біометричної ідентифікації було додано застосування біометричного хешування до згенерованого коду райдужної оболонки. Зазвичай

біохешування поєднує набір випадкових векторів, характерних для користувача, з біометричними ознаками.

Було досліджено, що деякі алгоритми біохешування мають надзвичайно низькі показники помилок порівняно з єдиним біометричним підходом, коли використовується справжній маркер. Таким чином, у випадку викрадення біометричного коду, може бути згенеровано новий на основі тих же біометричних даних, але із застосуванням іншого кодування або вектору.

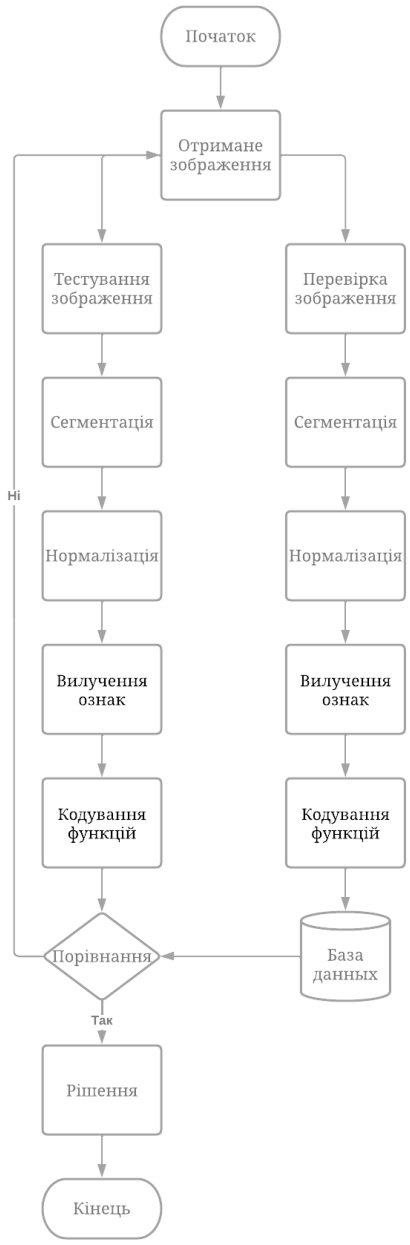


Рисунок 1.2 – Система розпізнавання райдужної оболонки ока

1.3 Огляд та аналіз наявних аналогів

Незважаючи на безліч переваг райдужної оболонки як біометричної риси, вона все ще не використовується на практиці так широко, як відбитки пальців або зображення обличчя. Хоча багато в чому це може пояснити права інтелектуальної власності, що стосуються розпізнавання райдужки технології, я вважаю, що тут також важливий фактор цінової політики існуючих систем розпізнавання райдужної оболонки ока.

Одним з прикладів буде IriTech IriShield BK 2121U. це камера з подвійною діафрагмою. Камера живиться через USB-порт і може використовуватися з ПК і ноутбуками з ОС Microsoft Windows, а також з пристроями під управлінням ОС Android або Linux.

Модуль захоплення райдужної оболонки поставляється в корпусі у формі бінокля, що означає, що райдужки знімаються на фіксованій відстані. Герметичний корпус та інфрачервоне світлодіодне підсвічування роблять камеру придатною для використання на вулиці.

Зроблені зображення райдужної оболонки ока відповідають стандарту ISO/IEC 19794-6.

Рисунок 1.3 – IriTech IriShield BK 2121U

Таблиця 1.3 – IriTech IriShield BK 2121U

Виробник	IriTech
Розмір зображення	640x480
Відстань захоплення райдужної оболонки	145 мм
Освітлення очей	Інфрачервоний
Розмір пристрою	200×145×72мм
Робоча температура	0°… +50°C
SDK	–
Підключення	USB
Підтримувана ОС	Windows, Linux, Android
Вартість	499€

Наступний приклад сканеру райдужної оболонки для UIDAI Aadhaar – Crossmatch I Scan2. Це сканер з подвійною райдужною оболонкою ока з підключенням USB. Конструкція сканера робить його придатним для використання в мобільних рішеннях.

Сканер може створювати зображення райдужної оболонки, що відповідають стандартам ANSI INCITS 379-2004 та ISO/IEC 19794-6. Покращує військові та цивільні програми безпеки за допомогою швидкої та надійної біометричної технології. Надійний і компактний сканер з подвійною райдужною оболонкою ока, I Scan 2 є невід’ємною частиною будь-якої комплексної програми керування ідентифікацією. За допомогою системи організації можуть розширити свої установки безпеки, додавши швидку мобільну реєстрацію, ідентифікацію та перевірку за допомогою програм для

захоплення райдужної оболонки ока. Ручний сканер з живленням від USB відповідає стандартам ANSI INCITS 379-2004 і ISO/IEC 19794-6, що забезпечує найякісніші зображення райдужної оболонки ока. І оскільки він заснований на стандартах, I Scan 2 сумісний з відомими алгоритмами узгодження райдужної оболонки і не потребує окремої ліцензії для отримання та заповнення зображень райдужки. Це найшвидший сканер для захоплення обох райдужної оболонки.

Рисунок 1.3 Сканер райдужної оболонки I Scan 2

Таблиця 1.3 – I Scan 2

Виробник	HID Global Inc.
Розмір зображення	2зображення по 480x480 кожне
Датчики	2×1,3
Освітлення очей	Близький інфрачервоний
Робоча відстань захоплення райдужної оболонки	125мм
Розмір пристрою у складеному вигляді	152×152×48мм
Робоча температура	0°…+49°C
Підключення	USB 2.0
Підтримувана ОС	Windows
Вага пристрою	0,5кг
Вартість	211€

Третім прикладом буде сканер райдужної оболонки 3M CIS 202. Цей сканер з живленням від USB дозволяє користувачам безпечно знімати зображення райдужної оболонки з високою роздільною здатністю, усуваючи при цьому нав’язливе положення зйомки або дискомфорт для користувача. CIS 202 має освітлювачі ближнього інфрачервоного випромінювання, світлодіодне освітлення та вбудований контроль чутливості спектрального зображення для захоплення світла в діапазоні 700-900 нанометрів.

Вихід зображення CIS 202 відповідає вимогам промислового формату обміну, підтримуючи записи ANSI NIST Type-17, із необробленими

зображеннями, які відповідають стандартам прямолінійних зображень ISO 19794-6. Закритий у міцний регульований корпус, який відповідає класу IP 54, міцний CIS 202, зручний для рук, дизайн окулярів ідеально підходить для біометричних застосувань, таких як реєстрація актів цивільного стану, видача біометричних документів, ідентифікація заявника, прикордонний контроль та управління звільненням ув’язнених.

Основні характеристики: одночасне захоплення подвійних райдужок, зручний дизайн окулярів, який відповідає рейтингу IP 54, не впливає на навколишнє освітлення, регульована міжзінична відстань, відповідає стандартам безпеки світлодіодних виробів, одночасно знімає подвійні райдужки.

Рисунок 1.3 Сканер райдужної оболонки 3M CIS 202

Таблиця 1.3 – 3M CIS 202

Виробник	3M Cogent
Розмір зображення	640x480
Розмір пристрою	142×120×64мм
Робоча відстань захоплення райдужної оболонки	120-140мм
Освітлення очей	Близький інфрачервоний
Робоча температура	0°…+40°C
Підключення	USB 2.0, USB 3.0
Підтримувана ОС	Windows
Вага пристрою	0,56кг
Вартість	300€

Незважаючи на досить широкий діапазон цін, комерційна райдужно-розпізнаванні системи все ще менш доступні, ніж конкуруючі недорогі системи, засновані, наприклад, на відбитках пальців.

Висновки до розділу 1

Отже, проаналізувавши якими бувають біометричні автентифікації, розпізнавання райдужної оболонки ока, повний шлях проходження отриманого зображення та загальну архітектуру систем розпізнавання, яка складається з 4 частин, робимо висновок, що загальна ефективність системи розпізнавання райдужної оболонки залежить від продуктивності її підсистем. Основним алгоритмом та методом нормалізації зображення який буде використовуватися обрано метод Даугмана, бо він є найефективнішим з усіх.

Проаналізувавши наявні аналоги пристроїв, виявлено велику націнку в вартості приладів. Враховуючи такий недолік, доцільно розробити застосунок який виконуватиме свої завдання, але буде набагато дешевшим за свої прототипи.

РОЗДІЛ 2
РОЗРОБКА АПАРАТНОЇ ЧАСТИНИ СИСТЕМИ ДЛЯ РОЗПІЗНАВАННЯ РАЙДУЖНОЇ ОБОЛОНКИ

У цьому розділі наведено основні структури вилучення ознак, особливостей, прийомів, що використовуються в техніках розпізнавання райдужної оболонки ока. Стадія попередньої обробки складається з локалізації та сегментації райдужної оболонки від отриманого зображення ока. Етап попередньої обробки складається з локалізації внутрішніх і зовнішніх меж райдужної оболонки і сегментації райдужки.

2.1 Локалізація

Локалізація меж райдужної оболонки дуже важлива для того, щоб видалити зіницю, склеру та іншу оклюзію, наприклад верхню та нижню повіку обстежуваного. Деякі методи, які використовуються дослідниками, серед них: пороговий метод та алгоритм Хаффа.

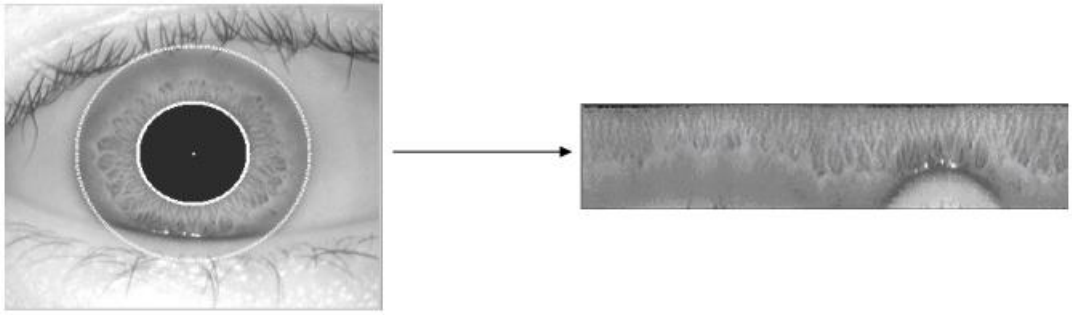


Рисунок 2.1. – Зразок райдужної оболонки після локалізації

Перетворення Хаффа є стандартним інструментом аналізу зображень для знаходження кривих, які можуть бути визначені в параметричній формі, таких як лінії і кола.

Визначення глобальної форми досягається за допомогою локальних шаблонів. Наприклад, розпізнавання кола може бути досягнуто шляхом розгляду сильних контурів на зображенні в якості локальних шаблонів і пошуку максимального значення колового перетворення Хаффа.

Метод локалізації, аналогічний методу Даугмана, також заснований на першій похідній зображення. У запропонованому методі Уайлдса карта контурів зображення може бути отримана шляхом знаходження порогу величини градієнта інтенсивності зображення:

$$|\nabla G(x,y)\cdot I(x,y)|, \qquad (2.1)$$

де

$$\nabla \equiv \left(\frac{\partial}{\partial_x},\frac{\partial}{\partial_y}\right) \quad \text{и} \quad G\cdot(x,y)=\frac{1}{2\pi\sigma^2}e^{\frac{-(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}, \qquad (2.2)$$

функція згладжування Гаусса з параметром масштабування σ для вибору правильного масштабу для аналізу контурів. Карта контурів потім використовується в процесі голосування, щоб максимізувати певне перетворення Хаффа для бажаного контуру.

Беручи отримані граничні точки як $(x_j,y_j)\ \ j=1,2,...,n$ перетворення Хаффа можна записати у вигляді:

$$H(x_c,y_c,r)=\sum_{j=1}^{n}h(x_j,y_j,x_c,y_c,r), \qquad (2.3)$$

$$h(x_j,y_j,x_c,y_c,r)=\begin{cases}1 \text{ якщо } g(x_j,y_j,x_c,y_c,r)=0\\ 0 \text{ в інших випадках}\end{cases}. \qquad (2.4)$$

Лімб і зіниця моделюються як кола, а параметрична функція g визначається як:

$$g(x_j,y_j,x_c,y_c,r)=(x_j-x_c)^2+(y_j-y_c)^2-r^2, \qquad (2.5)$$

припускаючи, що коло з центром (x_j, y_j) і радіусом r, граничні точки, які розташовані над колом, призводять до нульового значення функції.

Значення g потім перетвориться в 1 функцію h, яка являє собою локальний шаблон контуру. Локальні шаблони потім використовуються в процедурі голосування з використанням перетворення Хаффа, H, щоб знайти правильні межі зіниці і лімба. Для виявлення лімба використовується тільки інформація про вертикальний край.

Верхня і нижня частини, які містять інформацію про горизонтальний край, звичайно покриваються двома повіками.

Інформація про горизонтальний край використовується для виявлення верхніх і нижніх повік, які моделюються як параболічні дуги виду:

$$x(t) = (a_x t^2 + b_x t + c_x), \tag{2.6}$$

$$y(t) = (a_y t^2 + b_y t + c_y) \quad c \quad 0 \le t \le 1, \tag{2.7}$$

де параметри a_x, b_x, c_x, a_y, b_y і c_y, знову визначаються з використанням того ж підходу до моделювання на основі гістограми. Цей же процес використовується для визначення межі зіниці, але з наступними незначними змінами:

1. Зображення фільтрується за допомогою градієнтного крайового детектора, який не налаштований по напрямку. Це пов'язано з тим, що зінична межа менш схильна до оклюзії з боку вік;

2. Допустимі значення параметрів (x_c, y_c, r) обмежені колом, що описує межу райдужної оболонки.

На практиці вводиться акумуляторна матриця для знаходження точки перетину в просторі параметрів.

По-перше, нам потрібно розділити простір параметрів за допомогою сітки і створити акумуляторну матрицю відповідно до сітки.

Елемент в акумуляторній матриці позначає кількість кіл в просторі параметрів, що проходять через відповідну клітинку сітки в просторі параметрів. Цей номер також називається «номером для голосування».

Спочатку кожен елемент в матриці дорівнює нулю. Потім для кожної точки ребра в вихідному просторі ми можемо сформулювати коло в просторі параметрів і збільшити число голосів осередки сітки, через яку проходить коло. Цей процес називається голосуванням.

Після голосування, ми можемо знайти локальні максимуми в матриці накопичувача. Положення локальних максимумів відповідають центрам кіл у вихідному просторі. Після виявлення обох меж райдужної оболонки необхідно визначити розташування верхніх і нижніх повік, які можуть закривати райдужну оболонку. Для виконання цієї операції використовується крайовий детектор на основі градієнта, який налаштований на використання горизонтальних країв.

Представлені методи визначення місця розташування зіниці і лімба припускають, що межі є ідеальними колами. Хоча підходи різні, всі ці методи розглядають зіницю і лімб як колові криві, що може привести до неправильного виявлення меж.

На рис.2.1. деякі зображення райдужної оболонки ока з бази даних CASIA показують, що межі зіниці не є ідеальними колами.

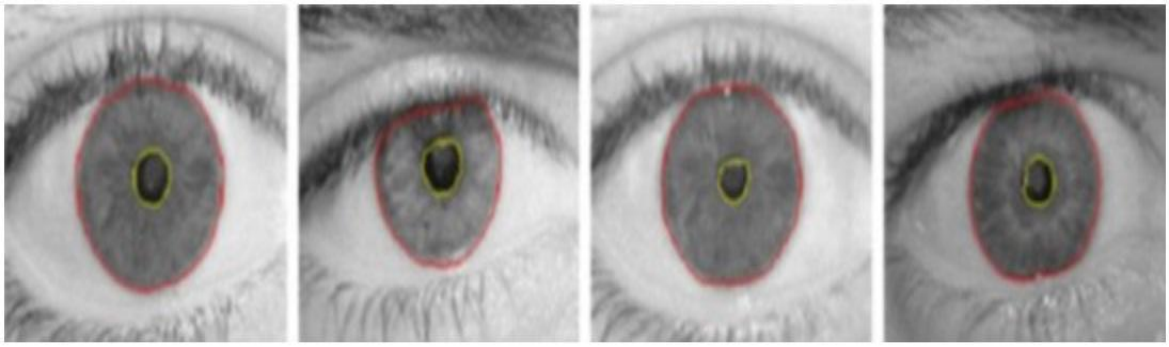


Рисунок 2.2 – Не ідеальні межі зіниць

Гуанчжу використовував модель семи рівнів для видалення оклюзії перед виділенням внутрішнього і зовнішнього кіл, що визначають область райдужної оболонки ока [15].

2.2 Сегментація

Сегментація ізолює область райдужної оболонки від усього ока. Цей етап є дуже важливим для правильної обробки системи розпізнавання. Деякі з використаних методів включають:

- Розширений інтегро-диференціальний метод;
- Рухомий агент;
- Перетворення Хафа;
- Кругове перетворення Хафа;
- Ітераційний алгоритм ;
- Метод активного контуру Чан-Везе;
- Спектральна густина Фур'є;
- Інтегро-диференціальний оператор;

Даугман запропонував інтегро-диференціальний оператор для визначення розташування внутрішніх і зовнішніх меж райдужної оболонки, а також верхнього і нижнього століття. Оператор обчислює часткову похідну середньої інтенсивності точок кола щодо збільшення радіуса r. Після згортання оператора з ядром Гаусса максимальна різниця між внутрішнім і зовнішнім колом визначатиме центр і радіус меж райдужної оболонки. Для виявлення верхніх і нижніх повік шлях інтеграції контуру змінюють від кругової до параболічної кривої. Уайлдс використовував виявлення країв і перетворення Хаффа, щоб локалізувати райдужну оболонку.

Детектор країв застосовується до зображення райдужної оболонки у відтінках сірого для створення карти країв. Фільтр Гауса застосовується для згладжування зображення, щоб вибрати правильний масштаб аналізу країв. Процедура голосування реалізується за допомогою перетворення Хаффа для пошуку потрібного контуру з карти країв. Центральна координатна

координата та радіус кола з максимальною кількістю крайових точок визначаються як контур, що цікавить. Для визначення повік контур визначається за допомогою параметра параболічної кривої замість параметра кола.

Метод пошуку зіниці був запропонований Тео і Еве для обчислення центру і площі зіниці [23]. Оскільки зіниця є найтемнішою областю на зображенні, цей підхід застосовує метод порогової сегментації, щоб знайти темні ділянки на зображенні райдужної оболонки. Темні ділянки називають «чорними дірами». Центр мас цих чорних дір обчислюється з глобального зображення.

Площа зіниці — загальна кількість цих чорних дір у регіоні. Радіус зіниці можна обчислити за формулою кола. Річард розробив підхід сегментації райдужної оболонки, який зміг компенсувати всі чотири типи шумів для досягнення більш високої точності. Він складається з чотирьох частин: спочатку локалізується зіниця за допомогою методів порогового визначення та кругового перетворення Хаффа. По-друге, дві області пошуку, включаючи зовнішню райдужну оболонку межі визначаються для розташування зовнішньої райдужки. Далі вибираються дві області пошуку на основі положення зіниці для виявлення верхньої та нижньої повіки і, нарешті, реалізується порогове значення для видалення вій, відображення та шумів зіниць. Виявилося, що ефективність методу в базі даних CASIA iris досягає 98,62% точності.

Для отримання центру та радіуса зіниці використовувався модифікований алгоритм Хаффа. Зовнішня межа райдужної оболонки була локалізована за допомогою інтегрального диференціального оператора. Для виявлення верхніх і нижніх повік була прийнята сегментація текстур. Енергія високого спектру в кожній області обчислюється для сегментації вій. Область з високою частотою вважається зоною вій. Верхні вії підходять параболічна дуга. Параболічна дуга показує положення верхньої повіки. Для виявлення

нижньої повіки використовується гістограма вихідного зображення. Область нижньої повіки сегментується для обчислення крайової точки нижньої повіки, а нижня повіка збігається з крайовими точками.

Конг і Ван запропонували фільтр Габора та підходи до дисперсії інтенсивності для виявлення вій [24]. Вії були розділені на роздільні вії та багатовійні. Роздільні вії визначаються за допомогою 1D фільтрів Gabor. Низька вихідна величина була отримана за рахунок згортки розділених вій з фільтром Gabor. Для кількох вій дисперсія інтенсивності у вікні менша, менша за поріг, центр вікна розглядався як вії.

2.3 Нормалізація

Нормалізація зазвичай використовується для зменшення сегментованих зображень райдужної оболонки до регуляризованого розміру для правильного виділення ознак. Існує багато методів нормалізації, використовуваних дослідниками. Ці методи включають: модель гумового листа, розроблену Даугманом, метод нормалізації трапеції, запропонований Мабубе та Абдолрезою. Деякі IRS були розроблені без нормалізації сегментованого зображення райдужки. Через складність інших методів нормалізації більшість розроблених систем використовують метод нормалізації гумового листа Даугмана.

У нормалізації область райдужної оболонки перетворюється таким чином, що вона має фіксовані розміри, щоб дозволити порівняння між однаковими зображеннями райдужної оболонки. Невідповідність між тими самими зображеннями ока пояснюється розтягуванням райдужної оболонки, викликаним розширенням зіниці від різного освітлення. Серед інших факторів, що викликають розширення є: обертання очей, поворот камери, нахил голови та змінна відстань зображення. Хороший процес нормалізації повинен створювати різні області райдужної оболонки для різних райдужних оболонок в однаковому стані, і він повинен давати постійні розміри для однієї і тієї ж райдужної оболонки в різних умовах. Інша велика проблема

полягає в тому, що область зіниць не завжди є концентричною в межах райдужної оболонки.

Модель гумового листа Даугмана пояснює відображення кожної точки області райдужки на полярні координати(r, Θ)

Відстань [0,1] дорівнює: r

Кут $[0, 2\pi]$ дорівнює : Θ

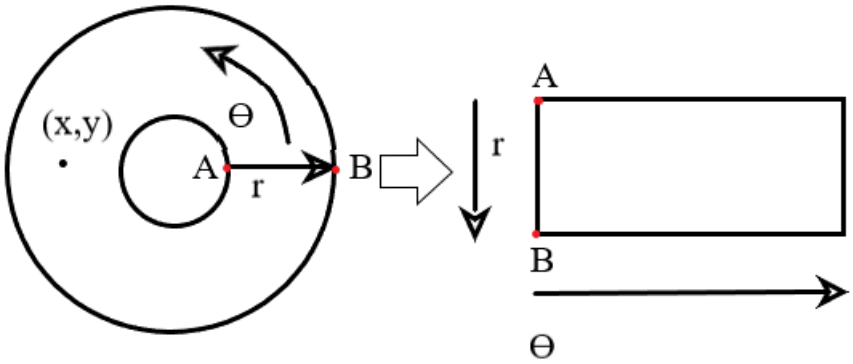


Рисунок 2.3 – Модель гумового листа Даугмана

Перетворення області райдужної оболонки з (x, y) декартових координат до нормалізованого неконцентричного полярного представлення моделюється як:

$$I\big(x(r,\Theta), y(r,\Theta)\big) \rightarrow I(r,\Theta), \qquad (2.3)$$

з

$$x(p,\Theta) = (1-p)\cdot x_p(\Theta) + p\cdot x_i(\Theta), \qquad (2.4)$$

$$y(p,\Theta) = (1-p)\cdot y_p(\Theta) + p\cdot y_i(\Theta), \qquad (2.5)$$

де зображення області райдужної оболонки ока: I(x,y),
початкові декартові координати: (x,y),
нормовані полярні координати: (r, Θ).

Координати меж зіниці та райдужної оболонки вздовж напрямку є: x_p і y_p та x_i і y_i. Модель гумового листа корисна для обліку розширення зіниць і

невідповідності розміру. Однак ця модель не компенсує невідповідності обертання. Для цієї проблеми два шаблони райдужної оболонки вирівнюються із збігом у зміщенні шаблонів райдужки в напрямку.

2.4 Відстань Хеммінга

Для зіставлення в якості метрики для розпізнавання використовується відстань Хеммінга, оскільки необхідні бітові порівняння. Даний використовуваний алгоритм відстані Хеммінга також включає маскування шуму, так що при обчисленні відстані Хеммінга між двома шаблонами райдужки використовуються тільки значні біти.

Відстань Хеммінга описується як HD у (2.4) та в розрахунках використовуються тільки ті біти, які відповідають 0 бітам в шумових масках обох малюнків райдужної оболонки. Малоймовірно, що два зображення однієї і тієї ж райдужної оболонки матимуть точно такий же код райдужної оболонки, але поріг дисперсії, як правило, від 10% до 20% може бути встановлений як прийнятний, оскільки, як правило, інша райдужна оболонка не буде настільки точно відповідати. Відстань буде розрахована з використанням тільки тих бітів, що генеруються з істинної області райдужної оболонки:

$$HD = \frac{1}{N - \sum_{k=1}^{N} x_{n_k} (OR) Y_{n_k}} \sum_{j=1}^{N} X_j (XOR) Y_j (AND) X_{n'_j} (AND) Y_{n'_j}, \qquad (2.4)$$

де X_j і Y_j - два бітових шаблони для порівняння;

X_{n_j} і Y_{n_j} - відповідні маски шуму для X_j і Y_j;

N - кількість бітів, представлених кожним шаблоном.

На рис.(2.4) показані зразки відстані Хеммінга між двома кодами райдужної оболонки одного ока зліва та відстані Хеммінга між двома кодами райдужної оболонки різних очей (самозванцями) праворуч. Середня відстань для самозванців становить близько 10%, тоді як середня відстань для

самозванців ближче до 50%. З цими даними поріг може бути встановлений ближче до 30% з незначною кількістю помилкових схвалень і помилкових відхилень.

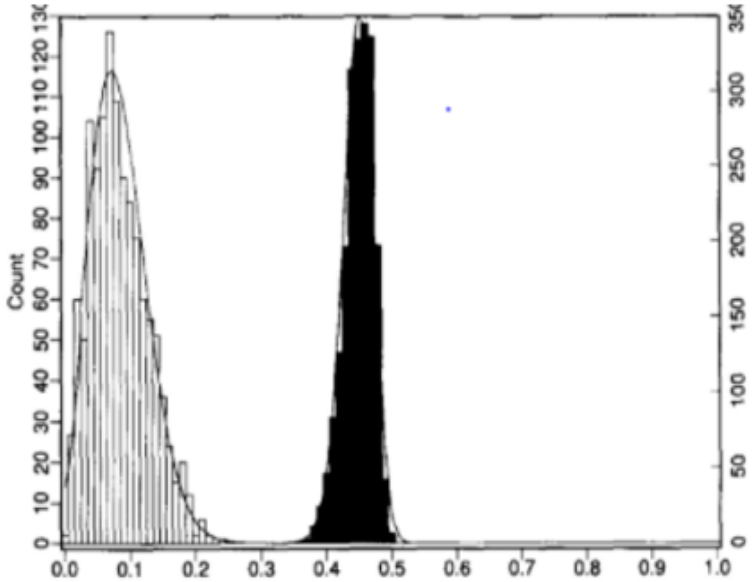


Рисунок 2.4 – Відстані Хеммінга

Оскільки окрема область райдужки містить ознаки з високим ступенем свободи, кожна область райдужної оболонки створюватиме бітову діаграму, яка не залежить від іншої, створюваної іншою радужкою, з іншого боку, два шаблони, створених з однієї і тієї ж райдужки, будуть скорельовані.

Якщо два бітових шаблони повністю незалежні, наприклад, райдужні шаблони, створені з різних райдужок, відстань Хеммінга між двома шаблонами має дорівнювати 0,5.

Це відбувається тому, що незалежність має на увазі, що два бітових шаблони будуть повністю випадковими, тому існує 0.5 шансу налаштувати біт в 1, і навпаки.

Якщо два шаблони отримані з однієї райдужної оболонки, відстань Хеммінга між ними буде близько до 0.0, так як вони дуже корельовані, і біти повинні погодитися між двома кодами райдужної оболонки.

2.5 Налаштування обладнання

Апаратний прилад портативного прототипу розпізнавання райдужної оболонки ока складається з трьох основних частин: основа, регульований по висоті окуляр, фотоапарат.

База містить одноплатний комп'ютер Raspberry PI 3, який керує процедурою отримання зображення, виконує обробку та проводить автентифікацію.

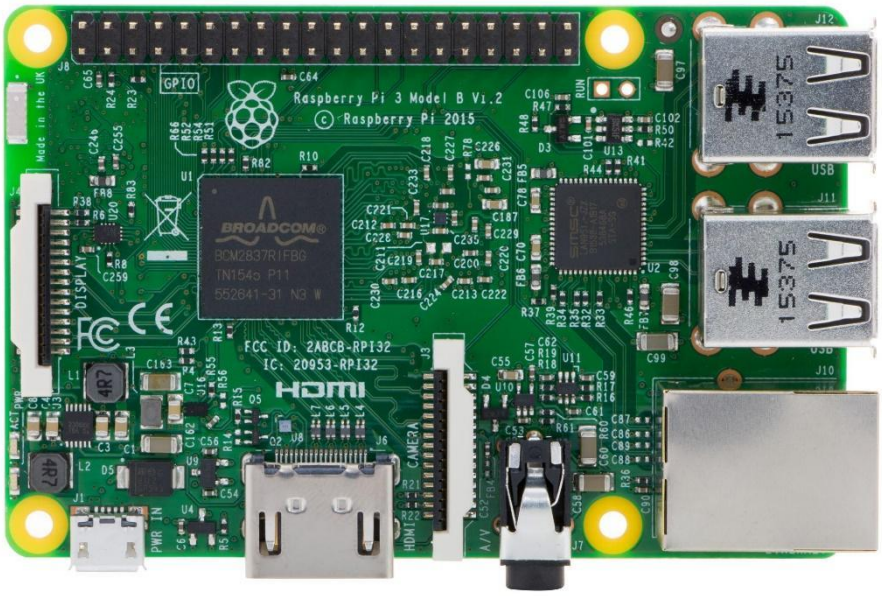


Рисунок 2.5 – Raspberry PI 3

Живлення йде від акумулятора ємністю 10Ah, розташованого поруч із Raspberry PI. Після розгортання акумулятор також можна замінити від зовнішнього джерела електроенергії. Основний контейнер виготовляється з пластику і прикручується до дерев'яної дошки для додаткової стабільності. Додаткові отвори для підключення проводів камери до Raspberry PI для полегшення живлення доступ і вихід HDMI також були просвердлені в основі.

Регульований по висоті окуляр дозволяє користувачеві правильно регулювати відстань між окуляром і камерою (закріпленої у фіксованому положенні) для зйомки чітке та зосереджене зображення райдужної оболонки. Основна частина окуляра виготовлена з пластикової меблевої

ніжки, що регулюється по висоті. Зігнуті і зварені між собою сталеві прути забезпечують стійкість і міцність окуляра. М'яка накладка для вух дозволяє користувачам зручно розташувати погляд для зйомки зображень. Під час роботи окуляр загортається темною тканиною, щоб зменшити кількість світла, що надходить ззовні джерела. Важливо, щоб освітленість райдужної оболонки можна було контролювати якомога точніше, щоб отримати найкращу контрастність малюнка райдужної оболонки і завжди забезпечити рівномірність освітленні зображення для конвеєра розпізнавання.

Камера складається з двох частин: 8-мегапіксельної фокусної камери Raspberry PI без IR-фільтра і друкована плата з 8 інфрачервоними і 4 білими прикріпленими до нього світлодіодними ліхтарями. Друкована плата дозволяє модифікувати інтенсивність інфрачервоного світла та інтенсивність світла можна налаштувати, щоб забезпечити достатнє освітлення, не було шкідливим для ока. Також було протестовано використовування білих світлодіодів для короткочасного освітлення ока та зменшення розширення зіниці але ефекти були мінімальними, а світло лише відволікало увагу. Під час отримання зображення сама камера розташовується приблизно на 6–7 сантиметрах від ока.

Камера Raspberry PI постачається з об’єктивом, налаштованим на фокус нескінченності, що означає, що об’єкти ближче 30 сантиметри виглядають розмитими. Тому відстань об’єктива від датчика була відрегульована та забезпечена фокусування від 5 до 8 сантиметрів.

Висновки до розділу 2

Отже, проаналізувавши основні структури вилучення ознак, особливостей, що використовуються в техніках розпізнавання райдужної оболонки ока, а саме з локалізації та сегментації райдужної оболонки від отриманого зображення ока, був обраний алгоритм сегментації, виділення ознак та відповідності Даугмана. Цей підхід був обраний, бо він добре задокументованим і одним з найбільш широко використовуваних.

Опрацювавши статті про алгоритми локалізації, вирішено обрати алгоритм Хаффа, який надавав найточніші результати.

Ця перша реалізація пристрою розпізнавання райдужної оболонки є відносно громіздкою та непрактичною для розгортання. Розмір наступного прототипу може бути значно зменшений за допомогою 3D-друку корпусу та окуляра, що робить його більш придатним для розгортання. Пристрій на даний момент також фіксує зображення лише одного ока і може стати більш практичним, а також краще працювати, якщо додати другий окуляр, що дозволяє пристрою одночасно знімати та розпізнавати обидві – ліву і права райдужну оболонку.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ РОЗПІЗНАВАННЯ РАЙДУЖНОЇ ОБОЛОНКИ

3.1 Python

Мовою для написання програмної частини розпізнавання райдужної оболонки було обрано Python. Саме цю мову було обрано, бо Python є потужною мовою програмування, має високорівневі ефективні структури даних і ефективний та простий у своєму реалізуванні підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис і динамічне введення Python разом з його інтерпретованою природою роблять його ідеальною мовою для написання сценаріїв і швидкої розробки додатків у багатьох областях та на великій кількості платформ.

Ця мова дає змогу написати сценарій оболонки Unix або пакетні файли Windows для деяких із цих завдань, але сценарії оболонки найкраще підходять для переміщення файлів і зміни текстових даних, що погано підходять для програм із графічним інтерфейсом або ігор. Ви можете написати програму на C/C++/Java, але це може зайняти багато часу на розробку, щоб отримати навіть першу чернетку програми. Python простіший у використанні, доступний в операційних системах Windows, macOS та Unix і допоможе виконувати роботу швидше.

Python простий у використанні, але це справжня мова програмування, яка пропонує набагато більше структури та підтримки великих програм, ніж можуть запропонувати сценарії оболонки або пакетні файли. З іншого боку, Python також пропонує набагато більше перевірки помилок, ніж C, і будучи мовою дуже високого рівня , він має вбудовані високорівневі типи даних, такі як гнучкі масиви та словники. Через свої більш загальні типи даних Python застосовний до набагато більшої проблемної області, ніж Awk або навіть Perl, але багато речей у Python принаймні так само легко зробити, як і в цих мовах.

Python дозволяє розділити програму на модулі, які потім можна повторно використати в інших програмах цієї ж мови. Він поставляється з великою колекцією стандартних модулів, які є сенс використовувати як основу для своїх програм або як приклади для початку навчання програмуванню на Python. Деякі з цих модулів забезпечують такі речі, як файлові вводи-виводи, системні виклики, сокети і навіть інтерфейси для графічних інструментів інтерфейсу користувача, таких як Tk.

Оскільки компіляція та зв’язування не потрібні Python, мова яка може заощадити час під час розробки програми. Інтерпретатор можна використовувати в інтерактивному режимі, що дозволяє легко експериментувати з особливостями мови, писати одноразові програми або тестувати функції під час розробки програми знизу вгору.

Python дозволяє писати програми компактно які будуть легко зрозуміти. Програми, написані на Python, зазвичай набагато коротші, ніж еквівалентні програми C, C++ або Java, з кількох причин:

– Типи даних високого рівня дозволяють виражати складні операції в одному операторі;
– Групування операторів здійснюється за допомогою відступів замість початкових і кінцевих дужок;
– Оголошення змінних або аргументів не потрібні.

3.2 Програмне рішення

Розробляючи програмну частину прототипу, вирішальною передумовою була обчислювальна ефективність. З огляду на це був розроблений ефективний конвеєр розпізнавання, який здатний швидко обробляти та вводити зображення очей і авторизувати або відхиляти користувачів.

Розроблений конвеєр розпізнавання райдужної оболонки, використаний у прототипі, заснований на конвеєрі розпізнавання райдужної

оболонки Даугмана модель [26] з невеликими змінами. Трубопровід може працювати в двох режимах: аутентифікація та контрольований режим реєстрації. Обидва режими складаються з шести основних етапів:

– Отримання зображень;
– Привласнення;
– Сегментація;
– Нормалізація;
– Кодування;
– Відповідність.

3.2.1 Отримання

Для отримання зображення користувач повинен нахилитися над пристроєм, притуливши око до пристрою. Потім розпочинається процедура зйомки за допомогою віддаленого робочого столу – можна використовувати для аутентифікації та реєстрації, або натисканням кнопки – може тільки використовуватися для аутентифікації. Коли користувач ініціює процедуру захоплення, вмикаються інфрачервоні світлодіодні лампи і незабаром після цього пристрій починає отримувати зображення.

```
class CrossEntropyLoss2d(nn.Module):

    def __init__(self, weight=None):
        super().__init__()

        self.loss = nn.NLLLoss(weight)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, outputs, targets):
        return self.loss(self.softmax(outputs), targets)

def jaccard_loss(true, logits, eps=1e-7):
```

```
    num_classes = logits.shape[1]
    if num_classes == 1:
        true_1_hot = torch.eye(num_classes + 1)[true.squeeze(1)]
        true_1_hot = true_1_hot.permute(0, 3, 1, 2).float()
        true_1_hot_f = true_1_hot[:, 0:1, :, :]
        true_1_hot_s = true_1_hot[:, 1:2, :, :]
        true_1_hot = torch.cat([true_1_hot_s, true_1_hot_f], dim=1)
        pos_prob = torch.sigmoid(logits)
        neg_prob = 1 - pos_prob
        probas = torch.cat([pos_prob, neg_prob], dim=1)
    else:
        true_1_hot = torch.eye(num_classes)[true.squeeze(1)]
        true_1_hot = true_1_hot.permute(0, 3, 1, 2).float()
        probas = F.softmax(logits, dim=1)
    true_1_hot = true_1_hot.type(logits.type())
    dims = (0,) + tuple(range(2, true.ndimension()))
    intersection = torch.sum(probas * true_1_hot, dims)
    cardinality = torch.sum(probas + true_1_hot, dims)
    union = cardinality - intersection
    jacc_loss = (intersection / (union + eps)).mean()
    return (1 - jacc_loss)
```

Рисунок 3.1 – Отримання зображення

Якщо користувач правильно розташований, не рухається і не блимає, процедура зйомки займає приблизно 3 секунди. Однак захоплення зображення може зайняти до 15 секунд, щоб захопити зображення у достатній якості. Кожне зображення зроблено з роздільною здатністю 8 мегапікселів, а зйомка займає приблизно 1 секунду і 0,5 секунди на обробку. Наразі зображення захоплюється зовнішнім процесом (засобом командного рядка Raspistill), записаним у файл, який потім зчитується розпізнавання діафрагми.

3.2.2 Захоплення

Захоплення складається з

– Початкових перетворень;

– Процедури видалення дзеркального відображення.

Початкові перетворення обрізають та масштабують вхідне зображення щоб забезпечити узгодження зображення для конвеєра розпізнавання, а процедура видалення дзеркального відображення, допомагає процесу сегментації.

З першим кроком трансформації ми хочемо обрізати фото з оком у центрі і масштабувати його до фіксованого розміру, щоб знайти центр ока, який нам потрібно і знайти зіницю. Ми робимо це за допомогою порогового значення зображення, створюючи маску пікселів, інтенсивність яких нижче порога. Потім ми знаходимо всі контури на пороговому зображенні та розглядаємо центр контуру зіниці як центр ока.

Також розраховуємо мінімальне окружне коло контуру який пізніше використовуємо для обмеження області, яка розглядається для видалення дзеркального відображення. Маскування та виявлення відображення виконується на меншому зображенні, 1/8 частини вихідного розміру, з метою підвищення обчислювальної ефективності. Отримане коло потім зменшується до оригінального розміру. З набутим розташуванням центру зіниці потім вирізаємо розміром в центрі ока 1920 × 1680 пікселів від вихідного зображення.

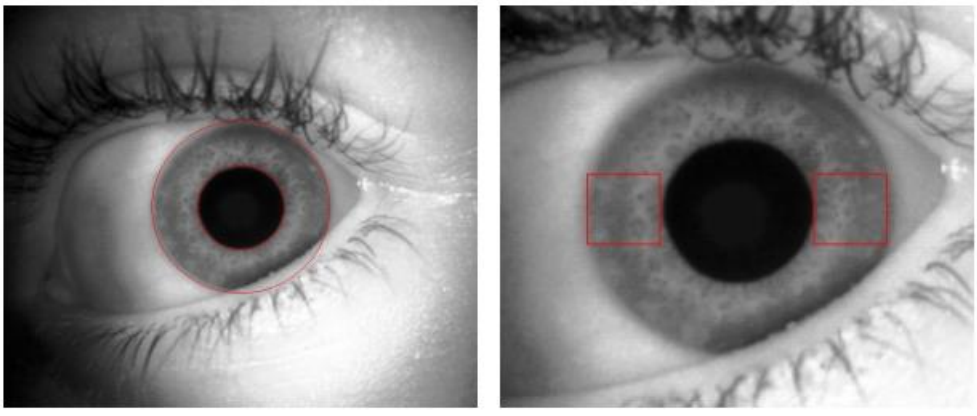


Рисунок 3.2 – Приклади проміжних результатів у райдужній оболонці програми

Виявлення меж райдужної оболонки с Інтегро-диференціальним оператором Даугмана – ліворуч, квадратні області, що використовуються для оцінки розмиття – праворуч.

На другому кроці ми видаляємо дзеркальні відблиски від перетвореного зображення. Ми робимо це шляхом заміни всіх пікселів у ділянці зіниці, це єдине місце, де присутні дзеркальні відображення, інтенсивність яких вище попередньо визначено дзеркальним порогом із середньою інтенсивністю усіх пікселів зіниць нижче порога дзеркала.

Відцентроване та замасковане зображення потім зменшується до кінцевого розміру 640 × 560 пікселів. Перед тим, як зображення буде передано до сегментації частини програми, виконаємо вирівнювання контрасту.

Вирівнювання контрасту робить сегментацію більш надійною, а також покращує якість вилученого коду райдужної оболонки. Попередня обробка процедури, використані в нашому прототипі, засновані на [5].

3.2.3 Сегментація

Для сегментації райдужної оболонки ми використовуємо інтегро-диференціальний оператор Даугмана, щоб знайти кругову сформовану райдужку. Інтегро-диференціальний оператор визначається як

$$max_{(r,x0,y0)} \left|G\sigma(r) * \frac{\delta}{\delta r} \oint_{r,x0,y0} \frac{I(x,y)}{2\pi r} ds\right|, \qquad (3.1)$$

де I(x, y) – зображення ока,

r – радіус пошуку,

Gσ(R) є гаусовою згладжуючою функцією,

s є контур кола, заданий r, x0, y0.

Оператор шукає коло, де є максимальна зміна інтенсивності пікселів, максимальний градієнт, шляхом тестування різних параметрів кола (r, x, y). Сам оператор застосовується ітеративно, зменшуючи згладжування на кожній ітерації, щоб отримати точне розташування. Ітеративний підхід також

дає кращі результати, оскільки ми лише точно шукаємо місцевість поблизу меж райдужної оболонки/зіниці.

```python
class OsIris(object):
    def __init__(self, cfg):
        self.cfg = cfg

    def get_mask(self, img_paths):
        # Call OSIRIS from command line
        txt_path = self.cfg["imagelist_path"] + str(int(time.time()*1000000%1000))+ '.txt'
        while os.path.exists(txt_path):
            txt_path = self.cfg["imagelist_path"] + str(int(time.time()*1000000%1000))+ '.txt'
        with open(txt_path, 'w') as f:
            for img_path in img_paths:
                f.write(img_path + "\n")

        os.chdir("/home/pi/Desktop/iris/OSIRIS_SEGM/src")
        os.system("./osiris "+ txt_path)

        masks = []
        for img_path in img_paths:
            img_name = self.cfg["output_path"] + img_path.split('/')[-1].split('.')[0]
            mask = np.array(Image.open(img_name+self.cfg["mask_suffix"]).convert('L'))
            mask = mask[:,:,0] if len(mask.shape) == 3 else mask
            masks.append(mask)

        return masks

    def get_circle(self, img_path):
        # Get circles from the circle coordinates given by OSIRIS
        img_name = self.cfg["output_path"] + img_path.split('/')[-1].split('.')[0]

        with open(img_name+self.cfg['param_suffix'], 'rb') as f:
            lines = f.readlines()
            lines = list(map(lambda l: [l.decode("utf-8").rstrip('\n')], lines))

        pupil_num = int(lines[0][0])
        iris_num = int(lines[1][0])
        pupil_coords = np.array(list(int(float(a)) for a in lines[2][0].split( ))).reshape(-1, 3)
        iris_coords = np.array(list(int(float(a)) for a in lines[3][0].split( ))).reshape(-1, 3)

        pupil_xyr = np.mean(pupil_coords, axis = 0).astype(int)
        pupil_xyr[2] = np.sqrt((pupil_coords[0,0]-pupil_xyr[0])**2 + (pupil_coords[0,1]-pupil_xyr[1])**2)
        pupil_xyr = [int(a) for a in pupil_xyr]

        iris_xyr = np.mean(iris_coords, axis = 0).astype(int)
        iris_xyr[2] = np.sqrt((iris_coords[0,0]-iris_xyr[0])**2 + (iris_coords[0,1]-iris_xyr[1])**2)
        iris_xyr = [int(a) for a in iris_xyr]

        return np.array(pupil_xyr), np.array(iris_xyr)
```

Рисунок 3.3 – Частина сегментації

Цей алгоритм дуже надійний, на попередньо обробленому зображенні, де контраст було вирівняно, а відображення були замасковані, дуже сильно покращили результати. Все ще існує ймовірність недосконалої сегментації, але це можна вирішити використовуючи, наприклад, словники для класів. Сегментацію також можна було б покращити шляхом маскування повік і вій.

Після сегментації ми оцінюємо якість зображення. Проводиться оцінка якості у два кроки:

Рисунок 3.4 – Нормалізоване зображення райдужної оболонки – зверху і остаточне код райдужної оболонки ока – внизу.

На першому кроці ми перевіряємо, чи була сегментація успішною. Це робиться шляхом перевірки, чи межі райдужної оболонки кола знаходяться в очікуваних межах. Різниця між виявленими центрами райдужки до зіниці і райдужної оболонки до меж не повинна перевищувати заздалегідь визначений поріг. Крім того, різниця в радіусах двох виявлених граничних кіл мають бути в межах заздалегідь визначеного діапазону. Якщо різниця менша ніж певний поріг, райдужка або занадто скорочена або сегментація не вдалася. Останнє також застосовується, якщо різниця вище певного порогу.

На другому кроці ми оцінюємо розмитість райдужної оболонки. Це робиться шляхом обчислення дисперсії Лапласіана [27] у двох квадратних областях з кожної сторони від райдужки. Тому що регіони розташовані у вертикальному центрі ока, найменше можуть бути закупорені повіками або віями. Середня дисперсія обох регіонів дає хорошу оцінку розмитості райдужки (вища дисперсія гостріша райдужка). За допомогою порогового значення дисперсії ми можемо гарантувати, що приймаються лише чіткі зображення.

3.2.3 Нормалізація та кодування діафрагми

Щоб обчислити райдужний код, потрібно спочатку перетворити діафрагму в полярну систему координат і відобразити її на кадрі, розмір якого визначається частотою дискретизації Θ і r. Щоб усунути різницю в роздільній здатності, ми використовуємо лінійну інтерполяцію.

Нормалізація мінімізує ефект розширення райдужки і робить подальшу обробку та підбір райдужної оболонки легше та швидше. Після нормалізації

зображення райдужної оболонки ми створюємо код діафрагми за допомогою фільтрації Log-Gabor [28]. Ми застосовуємо фільтр до представленої в частотній області нормалізованих зображень райдужки, а потім перетворюємо фазу, щоб отримати код діафрагми. Визначено використовуваний фільтр Log-Gabor як:

$$G(f) = exp(\frac{-(log(w*f0))^2}{2log(\sigma))^2})2, \quad (3.2)$$

де w являє собою довжину хвилі, використовуємо 18 одиниць,

f0 являє собою вектор центральної частоти використовуємо [0, 0,5] з кроком $\frac{N_\Theta}{2}$ де N_Θпредставляє кількість θ зразків у нормалізованому зображенні,

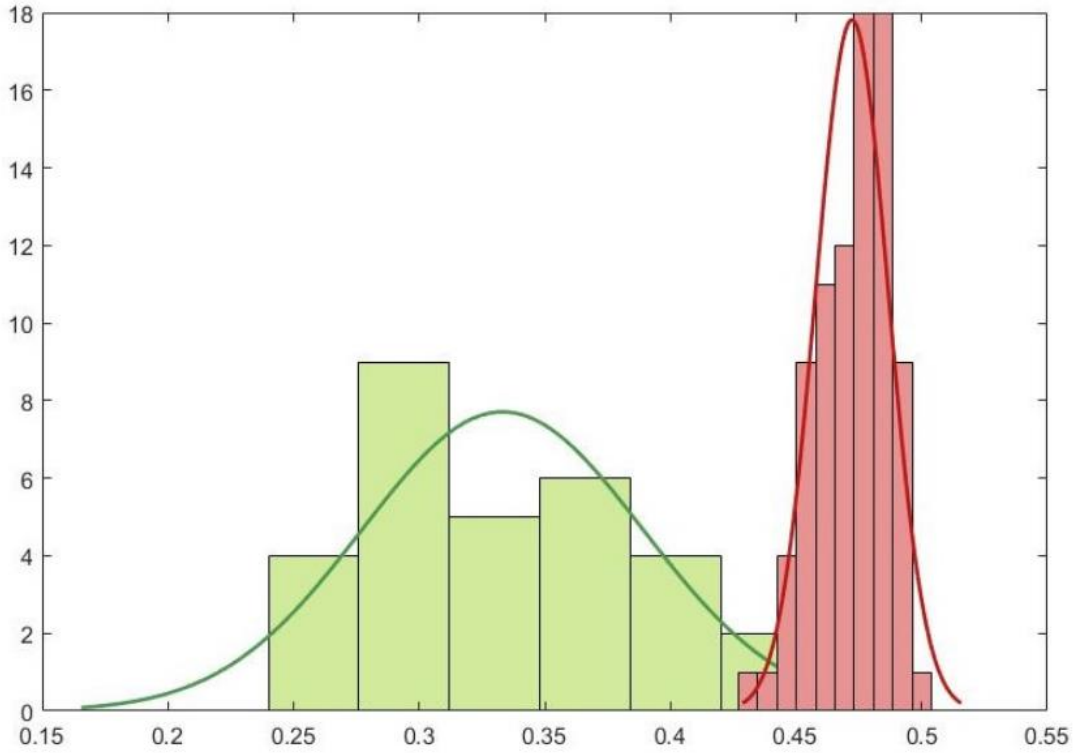
σ являє собою параметр, який впливає на пропускну здатність фільтра, використовуємо 0,5.

Весь процес генерації коду діафрагми може бути описано наступним чином. Спочатку трансформуємо кожен рядок нормалізованої райдужки до частотної області за допомогою FFT (Швидке Перетворення Фур'є). Ми згортаємо створений фільтр Log-Gabor для кожного рядка, а потім застосовуємо зворотне FFT щоб перетворити відфільтровані зображення назад у просторову область. За допомогою цього процесу ми отримуємо фазу, яку ми потім перетворюємо на чотирьох різних рівнях (по одному для кожного квадранта комплексної площини), представлених двома бітами. В кінцевому підсумку отримаємо двійкову матрицю розміру Nr×(N_Θ∗2), тобто використовується як фактичний код діафрагми. Нормована райдужна оболонка і кінцева код райдужної оболонки ока показано на рис. 3.2.

3.2.4 Збіг та реєстрація

Для реєстрації зображення користувачеві потрібно вручну оцінити якість зображення райдужної оболонки і сегментація. Зображення сегментованої райдужної оболонки та параметри якості, зазначені в підрозділі оцінки якості, надаються користувачеві, який потім може вирішити, чи є зображення належної якості для реєстрації. У потенційній

майбутній реалізації це могло б бути повністю автоматизованим шляхом покращення оцінки якості зображення.



Графік 3.1 – Порівняння з достовірним зображенням

У разі автентифікації отриманий код райдужної оболонки необхідно порівняти із записами, що зберігаються в базі даних прототипу. Оцінка відповідності визначається як відстань між двома (порівнюваними) кодами райдужної оболонки ока. Оцінка відповідності визначається як відстань Хеммінга між двома порівнюваними кодами райдужної оболонки ока. Для досягнення інваріантності обертання один з кодів діафрагми зміщується і для остаточного підрахунку вважається найкраща точка. Наш прототип зміщується до 16 разів у кожну сторону.

Рисунок 3.5 – Перевірка користувача

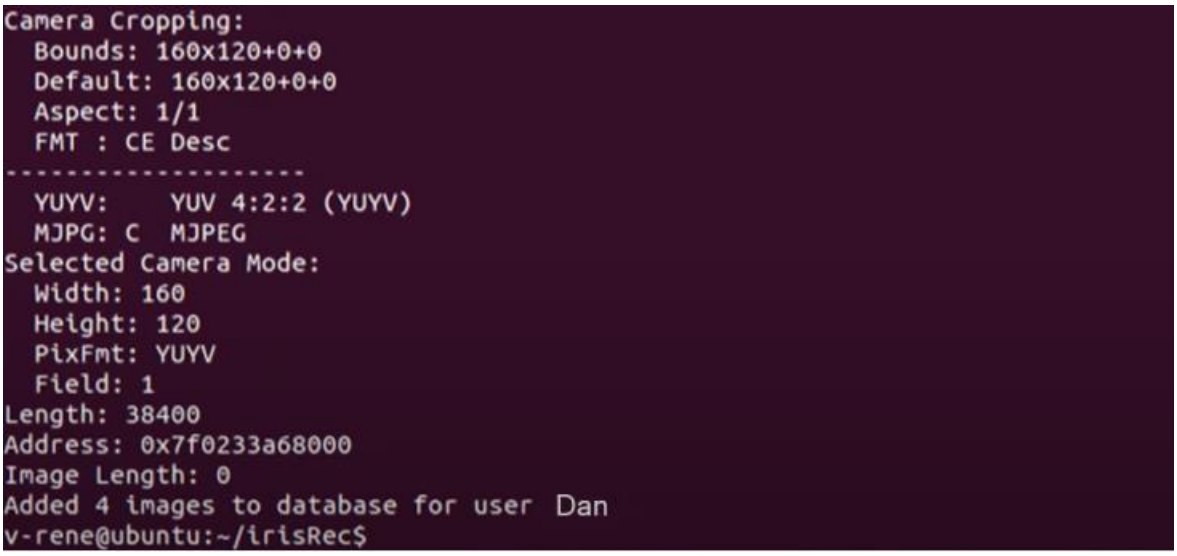


Рисунок 3.6 – Додавання користувача

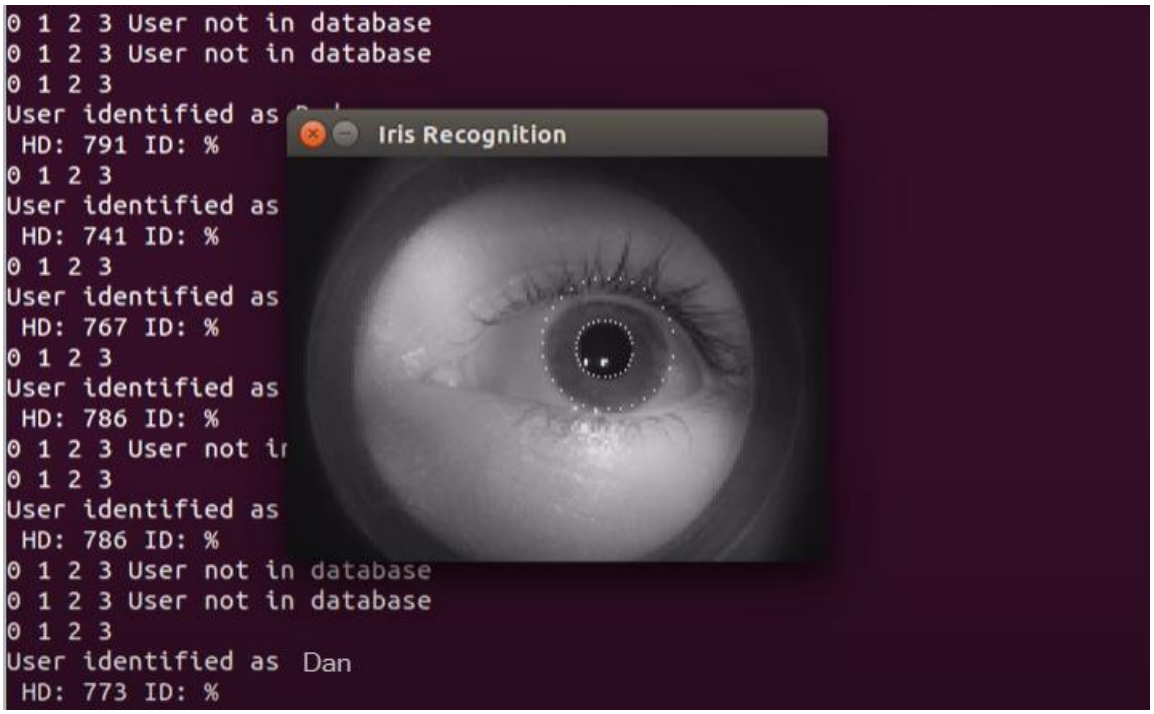


Рисунок 3.7 Ідентифікація користувача

Висновки до розділу 3

Отже, після тестування прототипу, записавши зображення ока за допомогою процедури, описаної вище, перевірено продуктивність прототипу і здатність пристрою фіксувати адекватні зображення, гарної якості для розпізнавання райдужної оболонки ока шляхом обчислення відповідності оцінки відстані Хеммінга.

Дивлячись на графік 3.1 ми бачимо, що недостовірний і достовірний розподіли розумно розділені, але є ще деяке перекриття. В основному це можна віднести до відсутності сегментації повік і вій у прототипі як і всі справжні зразки з відстанню Хеммінга вище 0,37 виявлено помірне перекриття повік. Для поточної реалізації критерій прийнятності розумної відстані Хеммінга може коливатися від 0,40 до 0,42. З рівним коефіцієнтом помилок нижче 2% отримані нами результати є задовільними, враховуючи той факт, що це наш перший прототип розпізнавання райдужної оболонки ока.

ВИСНОВКИ

У цій бакалаврській роботі представлено початковий прототип розпізнавання райдужної оболонки ока на базі Raspberry Pi побудований з використанням готових компонентів. Результати тестів показують, що прототип здатний виконувати розпізнавання райдужки з розумною продуктивністю. Із загальною вартістю близько 110€, наш прототип забезпечує доступну альтернативу дорожчим комерційним приладам по розпізнаванню райдужної оболонки ока системи.

Такий прототип може бути використаний в різних прикладних програмах, де потрібно підтвердити особистість людини. Області застосування такі як: контроль пасажирів в аеропортах, контроль доступу в зонах обмеженого доступу, прикордонний контроль, доступ до баз даних і фінансових послуг, доступ до мобільних приладів чи програм є прикладами застосування даного програмного-апаратного комплексу. Спираючись на графік порівняння зображення можна побачити, що недостовірний і достовірний розподіли розумно розділені, але є ще деяке перекриття. В основному це можна віднести до відсутності сегментації повік і вій у прототипі як і всі справжні зразки з відстанню Хеммінга вище 0,37 виявлено помірне перекриття повік. Для поточної реалізації критерій прийнятності розумної відстані Хеммінга може коливатися від 0,40 до 0,42. З рівним коефіцієнтом помилок нижче 2% отримані нами результати є задовільними.

Щоб покращити результати далі, можна застосувати кращий метод оцінки, який би зосередився більше на важливих областях райдужки. Подальші вдосконалення також можуть бути внесені в апаратну частину прототипу та програму розпізнавання, наприклад: покращити сегментацію, якість та метод підрахунку, що ще більше удосконалить продуктивність розпізнавання. У рамках майбутньої роботи правильним буде розробити прототип, надрукований на 3D, менший, практичніший. Щоб пришвидшити

роботу доцільно поставити та використовувати дві камери, які дозволять виконання аутентифікації за допомогою обох ірисів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Medically reviewed by the Healthline Medical Network — Written by The Healthline Editorial Team on January 19, 2018. URL: https://www.healthline.com/human-body-maps/iris-eye#1.

2. Bakk, Medien-Inf and Tilo Burghardt, (2002). Inside iris recognition. Report on identity verification: Information security. URL: https://patents.google.com/patent/US20020112177.

3. Iris recognition system. URL: https://patents.justia.com/inventor/aran-safir.

4. J. Daugman, "How iris recognition works," IEEE TCSVT, vol. 14, no. 1, pp. 21–30, 2004.

5. "Information theory and the iriscode," IEEE TIFS, vol. 11, no. 2, pp. 400–409, 2016.

6. Охорона праці (практикум) : навч. посіб. / за заг. ред. канд. техн. наук, доц. І. П. Пістуна. Львів : Тріада плюс, 2011. 436 с.

7. Iris Recognition Used to Secure Borders. – 2017. URL:https://www.biometricupdate.com/201205/irisrecognition-used-to-secure-borders.

8. Aishwarya, R. K, Vishnu, P. M, Prowino, M.P. G. and Saranya M., (2011). Human iris recognition using fuzzy neural concepts. International Conference on Bioscience, Biochemistry and Bioinformatics, vol. 5, 256- 260.

9. Falohun, S.A., Omidiora, E.O., Fakolujo, A.O. and Ojo, J.A, (2013). Development of an Iris-Based Access Control System Using a Two-Level Segmentation Aproach. International Journal of Computer Trends and Technology (IJCIT), 4(5), 1318 -1326.

10. Adegoke, B. O., Omidiora, E.O., Ojo, J. A. and Falohun, S. A., (2013). Iris feature extraction: a survey. Computer Engineering and Intelligent Systems, 4(9): 7-13.

11. Hunny M., Banshidar M and Phalguni G, (2008). Multi-algorithmic Iris Authentication System. World Academy of Science, Engineering and Technology, 148-151.

12. Сканер райдужної оболонки для UIDAI Aadhaar - Crossmatch I Scan2. URL: https://www.radiumbox.com/product/iris-scanner-of-crossmatch-for-aadhaar-uidai.

13. The IriShield USB BK 2121U. URL: https://www.biometricsupply.com/product/iritech-irishield-bk-2121u/.

14. Сканер райдужної оболонки 3M CIS 202. URL: https://www.indiamart.com/proddetail/gemalto-3m-cis-202-iris-scanner-22264525962.html.

15. Guangzhu X., Zaifeng Z. and Yide M., (2006). Improving the performance of Iris Recognition System using eyelids and eyelashes detection and Iris Image enhancement. Proceedings 6th IEEE International Conference on Cognitive Informatics (ICCI' 06), pp. 871 – 876.

16. Wildes, R.P., (1997). Iris recognition: an emerging biometric technology, Proceedings of the IEEE. Vol. 85, pp. 1348-1363.

17. Proenca, H. and Alexandre, L., (2006). Iris Segmentation Methodology for non- cooperative recognition. IEEE Proceedings on Vision, Image and Signal Processing, 153(2): 199-205.

18. Sastry, A. V.G.S and Durga, S. B., (2013). Enhamced Segmentation Method for Iris Recognition. International Journal of Computer Trends and Technology, 4(2): 68-71.

19. Shamsi, M., Saad, P., Ibrahim, S. and Kenari, A., (2009). Fast algorithm for Iris localization using Daugman circular integro-differential operator. International Conference of Soft Computing and Pattern Recognition, 393-398.

20. Sankowski, W., grabowski, K., Napieralska, M., Zubert, M., and Napieralski, A., (2010). Reliable Algorithm for Iris Segmentation in eye image. Image Vission Computing, vol., 28, 231-237.

21. De Martin-Roche, D., Sanchez-Avila, C. and Sanchez-Reillo, R., (2001). Iris Recognition for biometric identification using dyadic wavelet transform zero-crossing. In: IEEE International Canahan Conference on Security technology, pp.272-277.

22. Niladri B., Puhan, N. S. and Anirudh, S. K., (2011). Efficient segmentation technique for noisy frontial view iris images using Fourier spectral density. SIViP, Springer verlage, vol., 5, pp. 105-119.

23. Teo, C.C. and Ewe, H.T., (2005). An efficient one-dimesional fractal analysis for iris recognition. Proceedings of the 13th WSCG International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, pp. 157-160.

24. Kong, W. and Zhang, D., (2001). Accurate iris segmentation based on novel reflection and eyelash detetection model. Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing.

25. Мельниченко Д. А., Бурлаченко І. С. Особливості апаратної реалізації мультиагентних систем для розпізнавання райдужної оболонки. Могилянські читання–2021 : досвід та тенденції розвитку суспільства в Україні : глобальний, національний та регіональний аспекти. Комп’ютерні науки. Технічні науки : XXIV Всеукр. наук.-практ. конф. 8–12 листоп. 2021 р., м. Миколаїв : тези / М-во освіти і науки України ; ЧНУ ім. Петра Могили . – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. – С. 50–53.

26. J. Daugman, “Iris encoding and recognition using gabor wavelets,” Encyclopedia of Biometrics, pp. 973–983, 2015.

27. J. L. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez, and ´ J. Fernandez-Valdivia, “Diatom autofocusing in brightfield mi- ´ croscopy: a comparative study,” in ICPR, vol. 3. IEEE, 2000, pp. 314–317.

28. P. Yao, J. Li, X. Ye, Z. Zhuang, and B. Li, “Iris recognition algorithm using modified log-gabor filters,” in ICPR, vol. 4. IEEE, 2006, pp. 461–464.

Додаток А

Важливі фрагменти коду

```python
def load_image(file):
    return Image.open(file).convert('L')

def image_path(root, basename, extension):
    flnm = "{bs}{ext}".format(bs = basename, ext = extension)
    return os.path.join(root, flnm)
    #return os.path.join(root, f'{basename}{extension}')

def image_basename(filename):
    return os.path.basename(os.path.splitext(filename)[0])


class IrisSegmDataset(Dataset):
    def __init__(self, image_dir, mask_dir, filename, input_transform=None, target_transform=None, cvParam = 0.9, Train = True):
        self.image_dir = image_dir
        self.mask_dir = mask_dir

        self.input_transform = input_transform
        self.target_transform = target_transform

        with open(filename, 'rb') as f:
            lines = f.readlines()
            meta = list(map(lambda l: [i.rstrip('\n') for i in l.decode("utf-8").split(',')], lines))
            meta = meta[1:]

        self.image_paths = []
        self.mask_paths = []
        for m in meta:
            image_path = image_dir + m[0][2:]
            mask_path = mask_dir + m[0][2:]
            self.image_paths.append(image_path)
            self.mask_paths.append(mask_path)

        # Split train and test
        if Train:
            self.image_paths = self.image_paths[:int(len(self.image_paths)*cvParam)]
            self.mask_paths = self.mask_paths[:int(len(self.mask_paths)*cvParam)]
        else:
            self.image_paths = self.image_paths[int(len(self.image_paths)*cvParam):]
            self.mask_paths = self.mask_paths[int(len(self.mask_paths)*cvParam):]

        self.length = len(self.image_paths)

    def __getitem__(self, index):
        # Отримання зображення та маски
        image = load_image(self.image_paths[index])
        mask = load_image(self.mask_paths[index])

        # Resize image and mask
        image = image.resize((320, 240), Image.BILINEAR)
        mask = mask.resize((320, 240), Image.NEAREST)

        # Збільшення даних
        # Горизонтальний переворот
        if random.random() < 0.5:
            image.transpose(Image.FLIP_LEFT_RIGHT)
            mask.transpose(Image.FLIP_LEFT_RIGHT)

        # розмиття по Гауссу (ступінь 2,3,4) і посилення країв
        if random.random() < 0.83:
            random_degree = np.random.choice([1,2,3,4,5])
            if random_degree >= 2 and random_degree <= 4:
                # gaussian blurring
                image = image.filter(ImageFilter.GaussianBlur(random_degree))
            elif random_degree == 1:
                image = image.filter(ImageFilter.EDGE_ENHANCE)
            else:
                image = image.filter(ImageFilter.EDGE_ENHANCE_MORE)

        # обрізання зображення
        if random.random() < 0.7:
            crop_left = np.random.randint(20)
            crop_top = np.random.randint(15)
            crop_right = 320 - np.random.randint(20)
            crop_bottom = 240 - np.random.randint(15)

            image = image.crop((crop_left, crop_top, crop_right, crop_bottom))
            mask = mask.crop((crop_left, crop_top, crop_right, crop_bottom))
            image = image.resize((320, 240), Image.BILINEAR)
            mask = mask.resize((320, 240), Image.NEAREST)

        if self.input_transform is not None:
            image = self.input_transform(image)

        if self.target_transform is not None:
            mask = np.array(mask)
            mask[mask<128] = 0
            mask[mask>=128] = 1
            mask = self.target_transform(mask)

        return {"image": image, \
                "mask": mask}

    def __len__(self):
        return self.length
```

Рисунок 8 – Оброблення зображення

```
4   from PIL import Image
5
6   def colormap(n):
7       cmap=np.zeros([n, 3]).astype(np.uint8)
8
9       for i in np.arange(n):
10          r, g, b = np.zeros(3)
11
12          for j in np.arange(8):
13              r = r + (1<<(7-j))*((i&(1<<(3*j))) >> (3*j))
14              g = g + (1<<(7-j))*((i&(1<<(3*j+1))) >> (3*j+1))
15              b = b + (1<<(7-j))*((i&(1<<(3*j+2))) >> (3*j+2))
16
17          cmap[i,:] = np.array([r, g, b])
18
19      return cmap
20
21  class Relabel:
22
23      def __init__(self, olabel, nlabel):
24          self.olabel = olabel
25          self.nlabel = nlabel
26
27      def __call__(self, tensor):
28          tensor[tensor == self.olabel] = self.nlabel
29          return tensor
30
31
32  class ToLabel:
33
34      def __call__(self, image):
35          return torch.from_numpy(image).unsqueeze(0)
36
```

Рисунок 9 – Оброблення зображення

```
63  dataset = "WACV"
64  if dataset == "WACV":
65      # Load data for WACV
66      with open('/home/pi/Desktop/iris/WACV_Data/metadata.csv', 'rb') as f:
67          lines = f.readlines()
68      meta = list(map(lambda l: [i.rstrip('\n') for i in l.decode("utf-8").split(',')], lines))
69      meta = meta[1:]
70  else:
71      # Load data for NDIris3D
72      with open('/media/pi/PKBACK/LG4000_meta.csv', 'rb') as f:
73          lines = f.readlines()
74      orig_meta = list(map(lambda l: [i.rstrip('\n') for i in l.decode("utf-8").split(',')], lines))
75      x = scipy.io.loadmat('/media/pi/PKBACK/index_label.mat')
76      index_cross = x['index_cross'].flatten()
77      index_direct = x['index_direct'].flatten()
78      meta = []
79      for i in range(len(index_cross)):
80          orig_cross = orig_meta[index_cross[i]]
81          tmp_cross = [orig_cross[0], str(i+1), orig_cross[1]]
82          tmp_cross.append('live' if orig_cross[5]=='no' else 'fake')
83          tmp_cross.append(orig_cross[2])
84          tmp_cross.append(orig_cross[3])
85          tmp_cross.append(orig_cross[6])
86          tmp_cross.append(orig_cross[-2])
87
88          orig_direct = orig_meta[index_direct[i]]
89          tmp_direct = [orig_direct[0], str(i+1), orig_direct[1]]
90          tmp_direct.append('live' if orig_direct[5]=='no' else 'fake')
91          tmp_direct.append(orig_direct[2])
92          tmp_direct.append(orig_direct[3])
93          tmp_direct.append(orig_direct[6])
94          tmp_direct.append(orig_direct[-2])
95
96          meta.append(tmp_cross)
97          meta.append(tmp_direct)
98
```

Рисунок10 Тестування зображення

```
class OsIris(object):
    def __init__(self, cfg):
        self.cfg = cfg

    def get_mask(self, img_paths):
        # Call OSIRIS from command line
        txt_path = self.cfg["imagelist_path"] + str(int(time.time()*1000000%1000))+ '.txt'
        while os.path.exists(txt_path):
            txt_path = self.cfg["imagelist_path"] + str(int(time.time()*1000000%1000))+ '.txt'
        with open(txt_path, 'w') as f:
            for img_path in img_paths:
                f.write(img_path + "\n")

        os.chdir("/home/pi/Desktop/iris/OSIRIS_SEGM/src")
        os.system("./osiris "+ txt_path)

        masks = []
        for img_path in img_paths:
            img_name = self.cfg["output_path"] + img_path.split('/')[-1].split('.')[0]
            mask = np.array(Image.open(img_name+self.cfg["mask_suffix"]).convert('L'))
            mask = mask[:,:,0] if len(mask.shape) == 3 else mask
            masks.append(mask)

        return masks

    def get_circle(self, img_path):
        # Get circles from the circle coordinates given by OSIRIS
        img_name = self.cfg["output_path"] + img_path.split('/')[-1].split('.')[0]

        with open(img_name+self.cfg['param_suffix'], 'rb') as f:
            lines = f.readlines()
            lines = list(map(lambda l: [l.decode("utf-8").rstrip('\n')], lines))

        pupil_num = int(lines[0][0])
        iris_num = int(lines[1][0])
        pupil_coords = np.array(list(int(float(a)) for a in lines[2][0].split( ))).reshape(-1, 3)
        iris_coords = np.array(list(int(float(a)) for a in lines[3][0].split( ))).reshape(-1, 3)

        pupil_xyr = np.mean(pupil_coords, axis = 0).astype(int)
        pupil_xyr[2] = np.sqrt((pupil_coords[0,0]-pupil_xyr[0])**2 + (pupil_coords[0,1]-pupil_xyr[1])**2)
        pupil_xyr = [int(a) for a in pupil_xyr]

        iris_xyr = np.mean(iris_coords, axis = 0).astype(int)
        iris_xyr[2] = np.sqrt((iris_coords[0,0]-iris_xyr[0])**2 + (iris_coords[0,1]-iris_xyr[1])**2)
        iris_xyr = [int(a) for a in iris_xyr]

        return np.array(pupil_xyr), np.array(iris_xyr)

```

Рисунок 11 – Сегментація зображення

```
def s2i(size, bits, row, col, bit):
    # C + + and python use row - major order, so the last dimension is contiguous
    # in doubt, refer to https: // en.wikipedia.org / wiki / Row - _and_column - major_order  # Column-major_order
    return (bit + bits * (col + size * row))

def generateHistogram(src, size, bits, segmentationType):
    # load image
    if (segmentationType == "bg"):
        src = src[125:375, 195:445]

    downsample = False
    if ((size%2) == 0):
        downsample = True
        size = size//2

    if downsample:
        # Downsample (size is defined as x,y for this function not row,col)
        src = cv2.pyrDown(src, dstsize=(src.shape[1]//2, src.shape[0]//2))

    # initialize matrix of ones (to hold BSIF results)
    codeImg = np.ones((src.shape[0], src.shape[1]), np.int64)

    # Create wrapping border around the image
    # e.g. if size is 3x3, want a border of 1 to account for edges
    border = int(size) // 2

    imgWrap = cv2.copyMakeBorder(src, border, border, border, border, cv2.BORDER_WRAP)
    # Load hard coded filters
    filter = bsif.load(size, bits)
    print(filter)

    # Initialize current filter
    currentFilter = np.empty((size,size), np.float64)

    # Loop through filters, starting with the last one
    filterNum = bits - 1
    while (filterNum >= 0):
        # Load current filter (need to do it this way due to the storage of the filter - matlab file)
        for row in range(size):
            for col in range(size):
                currentFilter[row, col] = filter[s2i(size, bits, row, col, filterNum)]

        print(currentFilter)
        # Filter with filter2d - need to specify no image wrapping since we have done this previously
        # using default kernel anchor (centerpoint is anchor)
        filteredImg = cv2.filter2D(imgWrap, ddepth=cv2.CV_64F, kernel=currentFilter,delta=0,borderType=cv2.BORDER_CONSTANT)

        # Convert any positive values in the matrix to 2^(i-1) as in the matlab software
        for row in range(src.shape[0]):
            for col in range(src.shape[1]):
                # need to ignore the added border
                # for statements loop over the ORIGINAL image size so the end border (bottom and left) will be ignored
                if (filteredImg[(row + border), (col + border)] > 0.001):
                    # add the binary amount corresponding to the filter number
                    codeImg[row,col] += 2**(bits - 1 - filterNum)

        # Move to next filter
        filterNum -= 1

    # Create the histogram
    bins = 2**bits # for example, 256 bins for 8 bit filters (1-256)
    # This occurs because the image is initialized to ones, so zero position will need to be ignored

    hist = np.histogram(codeImg, bins=bins)
    hist = np.asarray(hist[0])

    # normalize
    mean = np.mean(hist)
    std = np.std(hist)
    hist = (hist - mean) / std

    return hist

if __name__ == "__main__":
    generateHistogram(np.random.rand(256,256), 22, 12, 'bg')
```

Рисунок 12 – Фільтрація

```
class OSPAD_3D(object):
    def __init__(self, cfg):
        self.cfg = cfg

        # setup light
        self.tilt_left = self.cfg["tilt_left"]
        self.tilt_right = self.cfg["tilt_right"]
        self.slant = self.cfg["slant"]
        self.left_light_pinv = self.get_light_pinv(left_eye=True)
        self.right_light_pinv = self.get_light_pinv(left_eye=False)

        # set up morph
        self.kernel_erode = np.ones((9,9), np.uint8)
        self.kernel_dilate = np.ones((2,2), np.uint8)

        print("Initialized OSPAD 3D")

    def get_light_pinv(self, left_eye = True):
        # Get light
        if left_eye:
            tilt = self.tilt_left
        else:
            tilt = self.tilt_right
        light_num = 2
        light = np.zeros([2,3])
        for ctr in range(0, light_num):
            light[ctr,0] = np.sin(np.deg2rad(self.slant[ctr]))*np.cos(np.deg2rad(tilt[ctr]))
            light[ctr,1] = np.sin(np.deg2rad(self.slant[ctr]))*np.sin(np.deg2rad(tilt[ctr]))
            light[ctr,2] = np.cos(np.deg2rad(self.slant[ctr]))

        light_pinv = linalg.pinv(light)

        return light_pinv

    def predict(self, imgs, masks, pupil_xyr, iris_xyr, left = True, iris_pct = 0.5):
        if left:
            light_pinv = self.left_light_pinv
        else:
            light_pinv = self.right_light_pinv

        images = np.zeros([256, 256, imgs.shape[2]])
        mask = np.ones([256, 256], np.uint8)

        # image preprocessing
        for ctr in range(imgs.shape[2]):
            # Read the mask
            mask_ctr = masks[:,:,ctr]

            # Get iris center
            iris_x = iris_xyr[ctr][0]
            iris_y = iris_xyr[ctr][1]

            # Calculate pupil center
            pupil_x = pupil_xyr[ctr][0]
```

```
            pupil_y = pupil_xyr[ctr][1]

            # Iris radius
            iris_radius = iris_xyr[ctr][2]
            iris_radius_internal = int(round(iris_radius * iris_pct))

            # Image cropping
            img_ctr = imgs[iris_y-iris_radius : iris_y+iris_radius, iris_x-iris_radius : iris_x+iris_radius, ctr].astype(np.double)
            img_ctr = cv2.resize(img_ctr, (256, 256), interpolation = cv2.INTER_AREA)
            images[:,:,ctr] = img_ctr

            # Larger pupil removal of mask
            mask_larger_pupil = np.copy(mask_ctr)
            for pixh in range(max(0, pupil_y - iris_radius_internal), min(480, pupil_y + iris_radius_internal)):
                for pixw in range(max(0, pupil_x - iris_radius_internal), min(640, pupil_x + iris_radius_internal)):
                    x = np.abs(pixw - pupil_x)
                    y = np.abs(pixh - pupil_y)

                    # Mask the pixels within the internal iris
                    if np.sqrt(x**2 + y**2) < iris_radius_internal:
                        mask_larger_pupil[pixh, pixw] = 0
                    else:
                        break

            mask_larger_pupil = mask_larger_pupil[iris_y-iris_radius : iris_y+iris_radius, iris_x-iris_radius : iris_x+iris_radius]
            mask_larger_pupil = cv2.resize(mask_larger_pupil, (256, 256), interpolation = cv2.INTER_AREA)

            # Mask cropping
            mask_ctr = mask_ctr[iris_y-iris_radius : iris_y+iris_radius, iris_x-iris_radius : iris_x+iris_radius]
            mask_ctr = cv2.resize(mask_ctr, (256, 256), interpolation = cv2.INTER_AREA)

            # Morphological opening of the mask
            mask_morph_ctr = np.copy(mask_ctr)
            mask_morph_ctr = cv2.erode(mask_morph_ctr, self.kernel_erode, iterations = 1)
            mask_morph_ctr = cv2.dilate(mask_morph_ctr, self.kernel_dilate, iterations = 1)

            # Specular relfection removal
            img_spec = img_ctr > 240
            [r, c] = img_spec.nonzero()
            mask_ctr[r,c] = 0

            # Aggregate current mask on the overall mask
            mask *= mask_ctr*mask_morph_ctr*mask_larger_pupil

        [r, c] = mask.nonzero()

        # Calculate normal vectors
        normal = np.dot(images[r, c, :], np.transpose(light_pinv))
        normal = normal / linalg.norm(normal, axis=1)[:, np.newaxis]
        normal[np.isnan(normal)] = 0

        # Convert back to surface normal maps
        map_normal = np.zeros([images.shape[0], images.shape[1], 3])
        map_normal[r,c,:] = normal
```

```
            # Calculate the PAD score
            diff = normal - np.mean(normal, axis = 0)
            diff_norm = linalg.norm(diff, axis = 1)
            pad_score = np.var(diff_norm)
            return pad_score, map_normal
```

Рисунок 13 – Фільтрація зображення

```python
class IrisRecognition(object):
    def __init__(self, cfg):
        self.height = cfg["rubbersheet_height"]
        self.width = cfg["rubbersheet_width"]
        self.angles = angles = np.arange(0, 2 * np.pi, 2 * np.pi / self.width)
        self.cos_angles = np.zeros((self.width))
        self.sin_angles = np.zeros((self.width))
        for i in range(self.width):
            self.cos_angles[i] = np.cos(self.angles[i])
            self.sin_angles[i] = np.sin(self.angles[i])

        self.filter_size = cfg["recog_filter_size"]
        self.num_filters = cfg["recog_num_filters"]
        self.max_shift = cfg["recog_max_shift"]
        self.filter = scipy.io.loadmat(cfg["recog_bsif_dir"]+'ICAtextureFilters_{0}x{1}_{2}bit.mat'.format(self.filter_size, self.filter_size, self.num_filters))['ICAtextureFilters']
        print("Initialized IrisRecognition")

    # rubbersheet model for iris recognition
    def get_rubbersheet(self, image, cx, cy, pupil_r, iris_r):
        # Angle value
        rs = np.zeros((self.height, self.width), np.uint8)

        for j in range(self.height):
            rad = j /self.height

            x_lowers = cx + pupil_r * self.cos_angles
            y_lowers = cy + pupil_r * self.sin_angles
            x_uppers = cx + iris_r * self.cos_angles
            y_uppers = cy + iris_r * self.sin_angles

            # Fill in the rubbersheet
            Xc = (1 - rad) * x_lowers + rad * x_uppers
            Yc = (1 - rad) * y_lowers + rad * y_uppers

            rs[j, :] = image[Xc.astype(int), Yc.astype(int)]

        return rs

    # extract iris code
    def extract_code(self, rs):
        # wrap image
        r = int(np.floor(self.filter_size / 2));
        imgWrap = np.zeros((r*2+self.height, r*2+self.width))
        imgWrap[:r, :r] = rs[-r:, -r:]
        imgWrap[:r, r:-r] = rs[-r:, :]
        imgWrap[:r, -r:] = rs[-r:, :r]

        imgWrap[r:-r, :r] = rs[:, -r:]
        imgWrap[r:-r, r:-r] = rs
        imgWrap[r:-r, -r:] = rs[:, :r]

        imgWrap[-r:, :r] = rs[:r, -r:]
        imgWrap[-r:, r:-r] = rs[:r, :]
        imgWrap[-r:, -r:] = rs[:r, :r]
```

```python

        # Loop over all kernels in the filter set
        codeBinary = np.zeros((self.height, self.width, self.num_filters))
        for i in range(1,self.num_filters+1):
            ci = scipy.signal.convolve2d(imgWrap, np.rot90(self.filter[:,:,self.num_filters-i],2), mode='valid')
            codeBinary[:,:,i-1] = ci>0

        return codeBinary

    def matchCodes(self, code1, code2, mask1, mask2):
        margin = int(np.ceil(self.filter_size/2))
        self.code1 = code1[margin:-margin, :, :]
        self.code2 = code2[margin:-margin, :, :]
        self.mask1 = mask1[margin:-margin, :]
        self.mask2 = mask2[margin:-margin, :]

        scoreC = np.zeros((self.num_filters, 2*self.max_shift+1))
        for shift in range(-self.max_shift, self.max_shift+1):
            andMasks = np.logical_and(self.mask1, np.roll(self.mask2, shift, axis=1))
            xorCodes = np.logical_xor(self.code1, np.roll(self.code2, shift, axis=1))
            xorCodesMasked = np.logical_and(xorCodes, np.tile(np.expand_dims(andMasks,axis=2),self.num_filters))
            scoreC[:,shift] = np.sum(xorCodesMasked, axis=(0,1)) / np.sum(andMasks)

        scoreC = np.min(np.mean(scoreC, axis=0))

        return scoreC
```

Рисунок 14 – Розпізнавання райдужної оболонки

```
class MaxPoolingWithIndices(Layer):
    def __init__(self, pool_size=2, strides=2 ,padding='SAME',**kwargs):
        super(MaxPoolingWithIndices, self).__init__(**kwargs)
        self.pool_size=pool_size
        self.strides=strides
        self.padding=padding
        return
    def call(self,x):
        pool_size=self.pool_size
        strides=self.strides
        if isinstance(pool_size,int):
            ps=[1,pool_size,pool_size,1]
        else:
            ps=[1,pool_size[0],pool_size[1],1]
        if isinstance(strides,int):
            st=[1,strides,strides,1]
        else:
            st=[1,strides[0],strides[1],1]
        output1,output2=tf.nn.max_pool_with_argmax(x,ps,st,self.padding)
        return [output1,output2]
    def compute_output_shape(self, input_shape):
        if isinstance(self.pool_size,int):
            output_shape=(input_shape[0],input_shape[1]//self.pool_size,input_shape[2]//self.pool_size,input_shape[3])
        else:
            output_shape=(input_shape[0],input_shape[1]//self.pool_size[0],input_shape[2]//self.pool_size[1],input_shape[3])
        return [output_shape,output_shape]


class UpSamplingWithIndices(Layer):
    def __init__(self, **kwargs):
        super(UpSamplingWithIndices, self).__init__(**kwargs)
        return
    def call(self,x):
        argmax=K.cast(K.flatten(x[1]),'int32')
        max_value=K.flatten(x[0])
        with tf.variable_scope(self.name):
            input_shape=K.shape(x[0])
            batch_size=input_shape[0]
            image_size=input_shape[1]*input_shape[2]*input_shape[3]
            output_shape=[input_shape[0],input_shape[1]*2,input_shape[2]*2,input_shape[3]]
            indices_0=K.flatten(tf.matmul(K.reshape(tf.range(batch_size),(batch_size,1)),K.ones((1,image_size),dtype='int32')))
            indices_1=argmax%(image_size*4)//(output_shape[2]*output_shape[3])
            indices_2=argmax%(output_shape[2]*output_shape[3])//output_shape[3]
            indices_3=argmax%output_shape[3]
            indices=tf.stack([indices_0,indices_1,indices_2,indices_3])
            output=tf.scatter_nd(K.transpose(indices),max_value,output_shape)
            return output
    def compute_output_shape(self, input_shape):
        return input_shape[0][0],input_shape[0][1]*2,input_shape[0][2]*2,input_shape[0][3]

```

Рисунок 15 – Сегментація

```
keras.backend.set_learning_phase(0)
class SegNet(object):
    def __init__(self, cfg):
        # load the model
        self.cfg = cfg
        self.model = load_model(self.cfg['segnet_model_path'],
            custom_objects={
                'MaxPoolingWithIndices': MaxPoolingWithIndices,
                'UpSamplingWithIndices': UpSamplingWithIndices
            },
            compile=False
        )
        self.avgimage = np.load(self.cfg['avgimage_path'])
        print("Initialized SegNet")

    def get_mask(self, imgs):
        for idx, img in enumerate(imgs):
            img = np.array(img.resize((320, 240), Image.BILINEAR))
            img = np.stack([img, img, img], axis=2)
            # subtract the training average
            img = img - self.avgimage
            imgs[idx] = img

        # predict with the model
        inbatch = np.stack(imgs, axis = 0)
        t1 = time.time()
        out = self.model.predict(inbatch)
        print("predict time =", time.time() - t1)
        masks = out[:,:,:,1]<0.5
        masks = masks.astype('uint8') * 255
        masks = [masks[i] for i in range(len(masks))]

        return masks

    def get_circle(self, mask):
        # find iris circle
        mask_for_iris = 255 - mask
        iris_indices = np.where(mask_for_iris[40:-40, 40:-40] == 0)
        iris_radius_estimate = max(max(iris_indices[0]) - min(iris_indices[0]), max(iris_indices[1]) - min(iris_indices[1])) // 2
        iris_circle = cv2.HoughCircles(mask_for_iris, cv2.HOUGH_GRADIENT, 1, 50,
                                       param1=self.cfg["iris_hough_param1"],
                                       param2=self.cfg["iris_hough_param2"],
                                       minRadius=iris_radius_estimate-self.cfg["iris_hough_lowermargin"],
                                       maxRadius=iris_radius_estimate+self.cfg["iris_hough_uppermargin"])
        iris_x, iris_y, iris_r = np.rint(np.array(iris_circle[0][0])*2).astype(int)

        # find pupil circle
        pupil_circle = cv2.HoughCircles(mask, cv2.HOUGH_GRADIENT, 1, 50,
                                        param1=self.cfg["pupil_hough_param1"],
                                        param2=self.cfg["pupil_hough_param2"],
                                        minRadius=self.cfg["pupil_hough_minimum"],
                                        maxRadius=iris_r//2-self.cfg["pupil_hough_margin"])
        pupil_x, pupil_y, pupil_r = np.rint(np.array(pupil_circle[0][0])*2).astype(int)

        if np.sqrt((pupil_x-iris_x)**2+(pupil_y-iris_y)**2) > self.cfg["max_separation"]:
            pupil_x = iris_x
            pupil_y = iris_y
            pupil_r = iris_r // 3
        mask = cv2.resize(mask, (640, 480), interpolation = cv2.INTER_AREA)

        return mask, np.array([pupil_x, pupil_y,pupil_r]), np.array([iris_x, iris_y,iris_r])

```

Рисунок 16 – Сегментація райдужної оболонки

```python
class UNet(object):
    def __init__(self, cfg):
        self.cfg = cfg
        self.load_graph(self.cfg["unet_model_path"])

    def load_graph(self, model_filepath):
        self.graph = tf.Graph()
        self.sess = tf.InteractiveSession(graph = self.graph)

        with tf.gfile.GFile(model_filepath, 'rb') as f:
            gf = tf.GraphDef()
            gf.ParseFromString(f.read())

        self.input = tf.placeholder(np.float32, shape = [None, 1, 240, 320], name = 'test_input')

        tf.import_graph_def(gf, {'0': self.input})

        self.output_tensor = self.graph.get_tensor_by_name("import/231:0")

    def get_mask(self, imgs):
        masks = []
        for idx, img in enumerate(imgs):
            img = np.array(img.resize((320, 240), Image.BILINEAR))
            img = img / 255
            out = self.sess.run(self.output_tensor, feed_dict={self.input: img.reshape(1,1,240,320)})
            out = np.argmax(out, axis = 1).astype('uint8') * 255
            out = out.reshape(240,320)
            nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(out, connectivity=8)
            sizes = stats[1:,-1]
            nb_components -= 1
            min_size = 500

            for i in range(0, nb_components):
                if sizes[i] <= min_size:
                    out[output==i+1] = 0
            masks.append(out)

        return masks

    def get_circle(self, mask):
        # find iris circle
        mask_for_iris = 255 - mask
        iris_indices = np.where(mask_for_iris[40:-40, 40:-40] == 0)
        if len(iris_indices[0]) == 0:
            return None, None, None
        y_span = max(iris_indices[0]) - min(iris_indices[0])
        x_span = max(iris_indices[1]) - min(iris_indices[1])

        if x_span > y_span + 40:
            iris_radius_estimate = y_span // 2 + 20
        else:
            iris_radius_estimate = x_span // 2

        iris_circle = cv2.HoughCircles(mask_for_iris, cv2.HOUGH_GRADIENT, 1, 50,
                                       param1=self.cfg["iris_hough_param1"],
                                       param2=self.cfg["iris_hough_param2"],
                                       minRadius=iris_radius_estimate-self.cfg["iris_hough_lowermargin"],
                                       maxRadius=iris_radius_estimate+self.cfg["iris_hough_uppermargin"])
        if iris_circle is None:
            return None, None, None
        iris_x, iris_y, iris_r = np.rint(np.array(iris_circle[0][0])*2).astype(int)

        # find pupil circle
        pupil_circle = cv2.HoughCircles(mask_for_iris, cv2.HOUGH_GRADIENT, 1, 50,
                                        param1=self.cfg["pupil_hough_param1"],
                                        param2=self.cfg["pupil_hough_param2"],
                                        minRadius=self.cfg["pupil_hough_minimum"],
                                        maxRadius=iris_r//3+self.cfg["pupil_hough_margin"])
        if pupil_circle is None:
            return None, None, None
        pupil_x, pupil_y, pupil_r = np.rint(np.array(pupil_circle[0][0])*2).astype(int)

        if np.sqrt((pupil_x-iris_x)**2+(pupil_y-iris_y)**2) > self.cfg["max_separation"]:
            pupil_x = iris_x
            pupil_y = iris_y
            pupil_r = iris_r // 3
        mask = cv2.resize(mask, (640, 480), interpolation = cv2.INTER_AREA)

        return mask, np.array([pupil_x, pupil_y,pupil_r]), np.array([iris_x, iris_y,iris_r])
```

Рисунок 17 – Сегментація зображення

```
def get_cfg(cfg_path):
    cfg = yaml.load(open(cfg_path, 'r'))
    return cfg

def xyr_from_txt(param_path):
    with open(param_path, 'rb') as f:
        lines = f.readlines()
        lines = list(map(lambda l: [l.decode("utf-8").rstrip('\n').rstrip(' ')], lines))
    pupil_num = int(lines[0][0])
    iris_num = int(lines[1][0])
    pupil_coords = np.array(list(int(float(a)) for a in lines[2][0].split(' '))).reshape(-1,3)
    iris_coords = np.array(list(int(float(a)) for a in lines[3][0].split(' '))).reshape(-1,3)

    pupil_xyr = np.mean(pupil_coords, axis=0).astype(int)
    pupil_xyr[2] = np.sqrt((pupil_coords[0,0]-pupil_xyr[0])**2 + (pupil_coords[0,1]-pupil_xyr[1])**2)
    pupil_xyr = [int(a) for a in pupil_xyr]

    iris_xyr = np.mean(iris_coords, axis=0).astype(int)
    iris_xyr[2] = np.sqrt((iris_coords[0,0]-iris_xyr[0])**2 + (iris_coords[0,1]-iris_xyr[1])**2)
    iris_xyr = [int(a) for a in iris_xyr]

    return pupil_xyr, iris_xyr

def show(x):
    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.resizeWindow('image',640,480)
    cv2.imshow('image',x)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def save(x, name):
    x = Image.fromarray(x)
    x.save(name)

class pi_camera(object):
    def __init__(self, cfg):
        self.camera = PiCamera()
        self.camera.sensor_mode = 2
        self.camera.resolution = (2592, 1944)
        self.left_illum = LED(cfg["left_illum_gpio_pin"])
        self.right_illum = LED(cfg["right_illum_gpio_pin"])
        self.cfg = cfg

    def capture_images(self, image_prefix):

        # capture w/ left illum
        self.left_illum.on()
        self.right_illum.off()
        #self.camera.start_preview()
        self.camera.start_preview(fullscreen=False, window=(800,100,640,480))
        input("Press Enter to capture left illumination...")
        left_i_name = "{0}/{1}_l".format(self.cfg["image_save_dir"], image_prefix)
        self.camera.capture(left_i_name+'.jpg')
```

```
62
63          # capture w/ right illum
64          self.left_illum.off()
65          self.right_illum.on()
66          input("Press Enter to capture right illumination...")
67          right_i_name = "{0}/{1}_r".format(self.cfg["image_save_dir"], image_prefix)
68          self.camera.capture(right_i_name+'.jpg')
69          self.camera.stop_preview()
70          self.right_illum.off()
71
72          # Post-process pictures
73          left_i_name = "{0}/{1}_l".format(self.cfg["image_save_dir"], image_prefix)
74          right_i_name = "{0}/{1}_r".format(self.cfg["image_save_dir"], image_prefix)
75          self.post_process(left_i_name)
76          self.post_process(right_i_name)
77
78          print("Capture Complete!")
79
80      def post_process(self, img_name):
81          # Locate iris and save images
82          o_img = Image.open(img_name+'.jpg').convert('L')
83          o_img = np.array(o_img)
84          img = (o_img < 50)*255
85          img = img.astype('uint8')
86          nb_components, output, stats, centoids = cv2.connectedComponentsWithStats(img, connectivity=8)
87          sizes = stats[1:,-1]
88          nb_components -=1
89          max_size = 90000
90          for i in range(nb_components):
91              if sizes[i] > max_size:
92                  img[output==i+1] = 0
93
94          kernel = np.ones((9,9),np.uint8)
95          img = cv2.erode(img, kernel, iterations=2)
96
97          center = np.mean(np.where(img == 255), axis=1).astype('int')
98          img = Image.fromarray(o_img[center[0]-480:center[0]+480, center[1]-640:center[1]+640])
99          img = img.resize((640, 480), Image.BILINEAR)
100         img.save(img_name+'_processed.jpg')
```

Рисунок 18 – Підключення камери