Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Лисенко Д.С.
«____» _____ 2023 р.

Керівник проекту
доктор технічних наук, доцент
_____ Коваленко О.В.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*"*Інформаційні технології*"*_____

Спеціальність *125 "*Кібербезпека*"*

Освітньо-професійна (освітньо-наукова) програма _*"*Кібербезпека*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Лисенку Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF*

2. Керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 13-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Коваленко О.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Лисенко Д.С.
(прізвище та ініціали)

АНОТАЦІЯ

Лисенко Д.С. Програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Метою розробки є програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Результат роботи – програмна реалізація системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Visual C++.

Ключові слова: кібербезпека, біопараметричні характеристики, SURF

ABSTRACT

Lysenko D.S. Cybersecurity system software for identification by bioparametric characteristics based on the SURF method. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for a cyber security system for identification by bioparametric characteristics based on the SURF method.

The purpose of the development is the cyber security system software for identification by bioparametric characteristics based on the SURF method.

The result of the work is the software implementation of the cyber security system for identification by bioparametric characteristics based on the SURF method.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Visual C++ environment.

Keywords: cyber security, bioparametric characteristics, SURF

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

EOM – електронно–обчислювальна машина

IT – інформаційні технології

API – прикладний програмний інтерфейс

AppWizard – засоби автоматизованого створення додатків

CEBIT – комп’ютерна виставка

FAR – False Acceptance Rate

FRR – False Rejection Rate

ISO/IEC – стандарт

JTC1 – міжнародний комітет зі стандартизації в області IT

Md5 – алгоритм шифрування

MFC – Microsoft Foundation Class library

NIST – національний інститут стандартизації

OLE – технологія зв'язування й вбудовування об'єктів

PIN – personal identification number

SC37 – спеціальний біопараметричний комітет

SSL – Secure Sockets Layer

USB – Universal Serial Bus

ВСТУП

Актуальність теми. Біометрія – це біологічні вимірювання або фізичні характеристики, які можна використовувати для ідентифікації людей. Наприклад, відображення відбитків пальців, розпізнавання обличчя та сканування сітківки ока – це всі види біометричних технологій, але це лише найвідоміші варіанти.

Дослідники стверджують, що форма вуха, те, як людина сидить і йде, унікальні запахи тіла, вени на руках і навіть викривлення обличчя є іншими унікальними ідентифікаторами. Ці риси ще більше визначають біометрію.

Хоча вони можуть мати й інші застосування, біометрія часто використовується в безпеці, і ви можете поділити біометрію на три групи:

– Біологічна біометрія.

– Морфологічна біометрія.

– Поведінкова біометрія.

Біологічна біометрія використовує ознаки на генетичному та молекулярному рівнях. Це можуть бути такі особливості, як ДНК або ваша кров, які можна оцінити за зразком рідин вашого організму.

Морфологічна біометрія стосується будови вашого тіла. Більше фізичних рис, як-от ваше око, відбиток пальця чи форма вашого обличчя, можна нанести на карту для використання зі сканерами безпеки.

Поведінкова біометрія базується на шаблонах, унікальних для кожної людини. Те, як ви ходите, говорите або навіть друкуєте на клавіатурі, може бути ознакою вашої особи, якщо ці шаблони відстежуються.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для ідентифікації за біопараметричними характеристикам на основі методу SURF.

– Дослідження системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

– Програмна реалізація системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Біометрична ідентифікація відіграє все більшу роль у нашій щоденній безпеці. Фізичні характеристики є відносно фіксованими та індивідуальними – навіть у випадку близнюків. Унікальна біометрична ідентифікація кожної людини може бути використана для заміни або принаймні розширення систем паролів для комп'ютерів, телефонів і приміщень і будівель з обмеженим доступом.

Отримані та нанесені на карту біометричні дані зберігаються для порівняння з майбутніми спробами доступу. У більшості випадків ці дані зашифровані та зберігаються на пристрої або на віддаленому сервері.

Біометричні сканери – це апаратне забезпечення, яке використовується для отримання біометричних даних для підтвердження особи. Ці сканування збігаються зі збереженою базою даних, щоб схвалити або заборонити доступ до системи.

Іншими словами, біометричний захист означає, що ваше тіло стає «ключем», щоб розблокувати ваш доступ.

Біометрія широко використовується через дві основні переваги:

– **Зручність використання:** біометрія завжди з вами, її неможливо втратити чи забути.

– **Важко вкрасти або видати себе за себе:** біометричні дані не можна вкрасти, як пароль або ключ.

Хоча ці системи не є досконалими, вони пропонують багато обіцянок для майбутнього кібербезпеки.

1.2 Область застосування

Ось кілька типових прикладів біометричного захисту:

– Розпізнавання голосу.

– Сканування відбитків пальців.

– Розпізнавання обличчя.

– Розпізнавання райдужки.

– Датчики серцевого ритму.

На практиці біометрична безпека вже знайшла ефективне застосування в багатьох галузях.

Удосконалена біометрія використовується для захисту конфіденційних документів і цінностей. Citibank вже використовує розпізнавання голосу, а британський банк Halifax тестує пристрої, які відстежують серцебиття для перевірки особистості клієнтів. Ford навіть розглядає можливість встановлення біометричних датчиків в автомобілях.

Біометричні дані включені в електронні паспорти по всьому світу. У Сполучених Штатах електронні паспорти мають чіп, який містить цифрову фотографію обличчя, відбиток пальця або райдужної оболонки ока, а також технологію, яка запобігає зчитуванню чіпа – і зняттю даних – неавторизованими зчитувачами даних.

Коли ці системи безпеки розгортаються, ми бачимо, як плюси та мінуси відображаються в режимі реального часу.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Автоматизована система біоідентифікації «Ассад-ID»

Автоматизована система біоідентифікації «Ассад-ID» являє собою безконтактну, автономну, розподілену, відказостійку систему верифікації людини по зображенню його особи.

Режими роботи:

1. Верифікація (підтвердження дійсності абонента).

2. Ідентифікація (визначення абонента).

Состав системи:

– Сервер АССаД-ID здійснює ведення бази даних (дані про абонентів і їхні біометричні шаблони, конфігураційна інформація системи); надає користувальницький інтерфейс (через Web-Сервер) для конфігурування, моніторингу й керування системою; у системі може бути кілька серверів, між якими автоматично підтримується ідентичність інформації.

– Станція розпізнавання виконує розпізнавання абонентів, при якому відбувається порівняння відеозображення, отриманого з 4-х консолей розпізнавання, із шаблонами абонентів з бази даних сервера; система може містити необмежену кількість станцій розпізнавання.

– Консоль розпізнавання розміщається безпосередньо в точці доступу (кабіна, двері), має у своєму составі відеокамеру й touch-screen монітор, формує відеозображення особи абонента, а також відображає процес і результат розпізнавання.

– Автоматизовані робочі місця служать для уведення абонентів і їхніх біометричних шаблонів, конфігурування моніторингу й керування системою; робота здійснюється через Web-браузер.

Загальні характеристики:

– висока точність розпізнавання;

– підтримка функцій верифікації й ідентифікації в одному продукті;

– простота експлуатації й модернізації (побудована з використанням новітніх технологій в області розробки програмного забезпечення);

– висока надійність і відказостійкість («гаряче» резервування основних вузлів системи);

– можливість інтеграції з будь-якими системами контролю й керування доступом;

– сполучення функцій цифрової відеореєстрації й розпізнавання на одному встаткуванні.

Biolink IDenium

Призначення – система захисту інформації від несанкціонованого доступу за допомогою надійного й зручного механізму біометричної ідентифікації користувачів корпоративних мереж і додатків.

Розв'язувані завдання**:**

– істотне зниження ризиків (фінансових, конкурентних, репутаційних), пов'язаних з несанкціонованим доступом до інформаційних ресурсів підприємства;

– значне скорочення витрат на адміністрування паролів у масштабах організації;

– зниження непродуктивних витрат робочого часу співробітників, пов'язаних з рішенням проблем з паролями.

Функції:

– біометрична ідентифікація користувачів;

– ідентифікація при вході в операційну систему підтримуються ОС Microsoft Windows 10/11, Windows Server 2022;

– ідентифікація при вході в будь-яке Windows додаток за допомогою біометричного SSO BioLink Password Vault або шляхом інтеграції в додаток власної розробки за допомогою IDenium SDK;

– ідентифікація при роботі в термінальній сесії;

– підтримка Microsoft RDP і Citrix;

– при вході в термінальну сесію;

– при вході в будь-яке Windows додаток через термінальну сесію;

– підтримка «тонких клієнтів» на платформі Windows.

Можливості:

– централізоване зберігання й адміністрування біометричних даних;

– вхід по відбитках пальців з будь-якої робочої станції мережі;

– підвищений рівень безпеки;

– біометричний сервер для швидкого пошуку;

– біометрична ідентифікація «до-багатьох» у реальному часі;

– масштабованість і відказостійкість за рахунок установки додаткових серверів, підтримка «гарячої заміни»;

– повна інтеграція зі службою каталогів Microsoft Active Directory;

– підтримка будь-яких складних багатодомених конфігурацій (лісу, дерева доменів);

– автоматична реплікація біометричних даних по регіональних підрозділах (сайтам Active Directory);

– централізоване настроювання й редагування сценаріїв входу в додатки;

– призначувані політики безпеки для гнучкого настроювання;

– вхід по відбитку пальця, смарт-карті, паролю або їхній комбінації;

– дозвіл самостійної зміни біометричних ідентифікаторів на робочій станції;

– дозвіл кешування даних на робочій станції;

– інтегровані засоби адміністрування;

– засоби адміністрування інтегровані в стандартне оснащення Active Directory Users and Computers;

– зручність керування обліковими записами й біометричними даними користувача з єдиного стандартного інтерфейсу;

– автоматичне розгортання клієнтського ПЗ;

– після підключення пристрою й установки драйвера, клієнтське ПЗ й відновлення можуть бути доставлені автоматично можливість вилученої реєстрації біометричних даних користувача.

BioTime

BioTime – це сучасна біометрична система обліку робочого часу й контролю фізичного доступу. BioTime інтегрує й ефективно реалізує ключові функції керування персоналом (облік, аналіз і контроль використання робочого часу) і безпеки (розмежування доступу в будинки й приміщення).

Вірогідність звітів і надійність захисту забезпечується ідентифікацією співробітників по унікальних біометричних параметрах – відбиткам пальців. Реєстрація приходів і відходів співробітників здійснюється по пред'явленні ідентифікаторів автоматично – з оптимальною швидкістю й незмінною точністю.

Із системою BioTime працювати зручно й легко. Дружній, продуманий інтерфейс можна швидко налаштовувати під конкретні завдання, велика номенклатура звітів задовольняє максимум потреб керівників, співробітників кадрових служб і відділів безпеки, бухгалтерів і менеджерів оперативної й тактичної ланки. Інтеграція BioTime у корпоративну інформаційну систему також здійснюється вільно й плавно, і не випадково керівники IT -служб і системних адміністраторів одними з перших підтримують впровадження BioTime.

Система BioTime адаптована під потреби підприємств, що діють у різних масштабах.

Директори компаній малого й середнього бізнесу по достоїнству оцінюють ефективність і економічність BioTime, простоту експлуатації цієї системи, розумні системні вимоги, які виконуються навіть у невеликому офісі.

Керівників великих підприємств залучає можливість застосовувати BioTime у мережі територіально розподілених філій і відділень – при тім, що повністю забезпечені централізоване керування системою й можливості її масштабування при успішному розвитку й подальшому росту компанії. BioTime ураховує управлінську ієрархію, дозволяє налаштовувати права й повноваження менеджерів, чітко регламентуючи можливості доступу кожного з них до необхідної інформації.

Безупинно нарощується функціонал системи BioTime у частині контролю фізичного доступу. Впроваджувати біометричні технології для рішення цих завдань можна й з «чистого аркуша» (при в'їзді в новий офіс або зміні робочого приміщення), і у вже існуючі системи контролю доступу (доповнюючи, наприклад, наявні карткові зчитувачі ідентифікацією по відбитках пальців при доступі в особливо важливі приміщення).

Біометрія визнана експертами самою перспективної й швидко, що розвивається технологією, контролю доступу, що використовують у найбільших банках, державних установах, на підприємствах і виробництвах особливої важливості (аеропорти, вокзали, порти, електростанції, машинобудівна, хімічна, оборонна промисловість, підприємства аерокосмічної галузі й т.д.).

Системи ідентифікації по райдужній оболонці ока

Цей метод ідентифікації заснований на унікальності райдужної оболонки ока. Алгоритми розпізнавання створені й запатентовані фірмою «Iridian». На сьогоднішній день це самі надійні системи ідентифікації. Імовірність помилкового допуску становить порядку 0,02%, а ймовірність недопуску «свого» – 2%. На патентах «Iridian» засновані рішення, пропоновані фірмами «Panasonic» і «LG».

«Panasonic» позиціонує три пристрої, заснованих на розпізнанні райдужної оболонки ока. Слід зазначити, що всі ці вироби для зйомки радужки використовують камери з інфрачервоним підсвічуванням. Виріб «BM-DT1200» являє собою пластиковий корпус, у якому встановлені дві камери, одна з яких є камерою відеоспостереження, а інша – інфрачервона камера – призначена для зйомки радужки ока. Слід зазначити наявність оптичної системи позиціонування ока щодо оптичної осі камери. Живлення й зв'язок здійснюються по USB-інтерфейсу. Оскільки даний виріб призначений винятково для авторизації користувача при вході в операційну систему сімейства Windows, його використання на підприємстві проблематично.

Вироби «BM-ET500» і «BM-ET300A» призначені для побудови системи доступу. Кожне з них має дві інфрачервоні камери для одночасної зйомки лівої й правої радужки, камеру відеоспостереження, систему візуального (і голосового в «BM-ET300A») позиціонування користувача щодо пристрою.

«BM-ET300» являє собою функціонально-закінчений пристрій, що має два Weigand-інтерфейси для підключення зчитувача карток і панелі контролю доступу, Ehernet-вихід для системної інтеграції (уведення користувача й т.д.), і чотири сигнальних виходи – допуск або відмова розпізнавання, статус живлення, детекція зовнішнього впливу. Є убудована цифрова камера відеоспостереження. Реєстрація користувача здійснюється на спеціалізованому сервері з наступною передачею еталона по мережі Ethernet. Автентифікація користувача здійснюється локально. Адміністрування системи здійснюється за допомогою спеціального програмного забезпечення «BM-ES300A». Кількість користувачів у системі обмежується ліцензією. Максимально можлива реєстрація до 5000 користувачів.

«BM-ET500» являє собою функціонально-закінчений пристрій ідентифікації по радужці. Додатково користувачеві може призначатися 10-символьний код. Об'єднання пристроїв в Ethernet-Мережу й керування замками забезпечуються спеціальним керуючим блоком «BM-ED500». Керування

здійснюється за допомогою спеціального програмного забезпечення «BM-ES500».

«LG» представляє систему доступу «IrisAccess 3000». На відміну від виробів «Panasonic», у ній використовується зображення радужки тільки одного ока в оптичному діапазоні з відповідним підсвічуванням. Є система оптичного й голосового позиціонування користувача.

Система доступу включає кілька серверів уведення еталонних зображень радужки «EOU3000». Взаємодія між серверами забезпечується за рахунок виділеного сервера – «суперсервера». Еталонне зображення по мережі Ethernet передається на керуючий блок «ICU3000», до якого приєднаються до 4 виробів уведення зображення радужки «ROU3000». Керування замками, підключення додаткового Weigand-інтерфейсу здійснюються за допомогою додаткової плати, що вставляється в «ICU3000». Є плата захвата відеозображення для використання в сервері або «ICU3000». Дані про кількість користувачів у системі, часі розпізнавання в рекламних матеріалах компанії відсутні.

Розглянуті вироби фірм «LG» і «Panasonic» не мають відкритого інтерфейсу й призначені для експлуатації тільки в рамках СКУД виробника. Використання декількох СКУД, що не мають єдиної інформаційної бази, на одному об'єкті, очевидно, недоцільно.

Системи ідентифікації користувача по особі

Є безліч систем, що працюють із зображенням людської особи. У Україні представлені вироби фірм «ISS» («Face Інспектор»), «Авп-Софт» («ІБС») і «A4Vision» («Vision Access System»).

«Face Інспектор» від «ISS»

Модуль розпізнавання осіб працює в середовищі «Інспектор+», використовуючи його відеопідсистему для одержання відеозображення. «Face Інспектор» автоматично шукає й виділяє особу людини, що рухається в контролюємій зоні. На основі отриманого зображення алгоритми модуля роблять обробку зображення на предмет виділення особи. Після визначення особи

програма виводить на монитор детектора осіб фрагмент зображення, пізнаний як особа. Далі програма «веде» захоплену особу для пошуку найкращого ракурсу. Кращий знайдений ракурс записується в базу даних.

З кожною особою, записаною в базу даних, може бути співвіднесений повний кадр камери, зображення з якої дає повну сцену, що містить обрана особа. Найкращий відеокадр з'являється в збільшеному масштабі на екрані комп'ютера. Складається фототека осіб (до 10 млн осіб), що зберігається в понадтривалому архіві.

Продукти «ISS» мають розвинені засоби інтеграції. Однак по своїй ідеології розглянутий модуль призначений скоріше для пошуку правопорушників на вулицях міста, чим для організації доступу. Його використання в СКУД досить проблематично, проте інтеграція даного виробу з детекторами металу й матеріалів, що діляться, у рамках СКУД дозволить забезпечити контроль за переміщенням відповідних матеріалів без участі персоналу, що досмотрює. Це може значно підвищити ефективність використання СКУД на підприємстві.

«Інтелектуальна біометрична система» від «АВП Софт»

Виріб являє собою набір бібліотек для операційних систем Windows і Linux, що забезпечують розпізнавання по особі, зображенню кисті руки або відбитку пальця. На виробника СКУД покладають завдання по інженерному забезпеченню зняття біометричних характеристик, що саме по собі є складним технічним завданням.

«Vision Access System» від «A4Vision»

До складу системи «Vision Access» входять:

– Запатентоване «A4Vision» біометричне встаткування для розгортання пунктів контролю доступу «Face Reader».

– Станції попередньої реєстрації користувачів «Enrollment Station» для збору біометричної бази даних шаблонів.

– Комплект розроблювача «Vision Access SDK» для інтеграції біометричних засобів у систему безпеки підприємства. Біометричне встаткування

працює в інфрачервоному діапазоні з відповідним підсвічуванням і забезпечує зняття 3D-зображення особи.

Дана технологія дозволяє домогтися:

– нечутливості до зовнішнього висвітлення;

– стійкості розпізнавання до рухів особи й поворотам голови;

– простоти зняття біометричних шаблонів.

Дані про кількість користувачів у системі й часі розпізнавання в рекламних матеріалах компанії відсутні.

Справжній виріб, очевидно, найбільшою мірою підходить для інтеграції в СКУД промислового підприємства.

Використання технічних засобів біоідентифікації в СКУД промислового підприємства натрапляє на цілий ряд проблем. Так, розглянуті системи ідентифікації по радужці не призначені для інтеграції в СКУД, відмінні від систем відповідних виробників. Система «Face Inspector» може мати обмежене застосування в СКУД підприємства. «Інтелектуальна біометрична система» не має технічного забезпечення для зняття біометричних характеристик, а у виробника й/або інсталятора СКУД найчастіше відсутня можливість виготовлення або адаптації відповідного встаткування.

Найбільш перспективним виробом, очевидно, для використання в СКУД промислового підприємства є система «Vision Access System» від «A4Vision», що задовольняє всім вищевикладеним вимогам.

MegaMatcher SDK

Особливості та можливості:

– Перевірено в проектах національного масштабу, включно з видачею паспортів та дедуплікацією виборців.

– Система відбитків пальців, сумісна з NIST MINEX, перевірена система ірису NIST IREX.

– Мультибіометричне рішення під ключ для проектів ідентифікації національного масштабу з MegaMatcher ABIS.

– Високопродуктивне зіставлення для великих систем за допомогою MegaMatcher Accelerator.

– Відбитки пальців, райдужну оболонку очей і обличчя можна порівняти на смарт-картках за допомогою MegaMatcher On Card.

– Включає модальності відбитків пальців, райдужки, обличчя, голосу та долоні.

– Зіставлення згорнутих, плоских і прихованих відбитків пальців.

– Підтримка BioAPI 2.0 та інших біометричних стандартів ANSI та ISO.

– Перевірка зображення обличчя на відповідність вимогам ICAO.

– Ефективне співвідношення ціна/продуктивність, гнучке ліцензування та безкоштовна підтримка клієнтів.

Технологія MegaMatcher для великомасштабних автоматизованих систем біометричної ідентифікації була представлена в 2005 році. З того часу технологія постійно вдосконалювалася, на сьогодні випущено понад 10 основних і проміжних версій.

Технологія MegaMatcher доступна як багатоплатформний SDK, який включає механізми розпізнавання відбитків пальців, обличчя, динаміка, райдужної оболонки ока та долоні, а також об’єднаний алгоритм для швидкої та надійної ідентифікації у великих системах. Біометричні програмні механізми базуються на глибоких нейронних мережах і містять багато власних алгоритмічних рішень, які особливо корисні для великомасштабних проблем ідентифікації. Деякі з цих рішень наведено в описі системи біометричної ідентифікації відбитків пальців, обличчя, голосу та райдужної оболонки ока нижче.

Механізм вилучення шаблонів відбитків пальців і зіставлення MegaMatcher:

– Повна відповідність MINEX. NIST визнав алгоритм відбитків пальців MegaMatcher сумісним з MINEX і придатним для використання в програмних програмах перевірки особи (PIV).

– Відбитки згорнутих і плоских пальців збігаються. Механізм відбитків пальців MegaMatcher зіставляє згорнуті та плоскі відбитки пальців між собою. Як правило, звичайні «плоскі» алгоритми ідентифікації відбитків пальців виконують порівняння між плоскими та згорнутими відбитками менш надійно через специфічні деформації згорнутих відбитків пальців. MegaMatcher дозволяє зіставляти відбитки пальців від плоского до плоского, від плоского до згорнутого або згорнутого до згорнутого з високим ступенем надійності та точності. Алгоритм зіставляє до 200 000 плоских записів відбитків пальців на секунду на одному ПК.

– MegaMatcher включає визначення якості зображення відбитків пальців, яке можна використовувати під час реєстрації, щоб переконатися, що в базі даних зберігатиметься лише шаблон відбитків пальців найкращої якості. Визначення якості зображення може визначити, чи палець занадто вологий, занадто сухий, натиснутий занадто сильно чи недостатньо, чи присутні лише кінчики пальців.

– Підробка виявлення відбитків пальців. Класифікація відсканованих зображень відбитків пальців на основі глибокого навчання використовується для розділення живих і неживих відбитків пальців для виявлення атаки презентації пальця. Ця функція охоплює спроби спуфінгу, здійснені за допомогою ecoflex, клею для дерева, латексу та желатину, і корисна для виявлення шахрайства.

– Узагальнення шаблону використовується для створення якіснішого шаблону з кількох відбитків пальців. Краща якість шаблонів забезпечує вищий рівень точності ідентифікації.

– MegaMatcher стійкий до перекладу відбитків пальців, обертання та деформації. Він використовує власний алгоритм зіставлення відбитків пальців, який ідентифікує відбитки пальців, навіть якщо вони повернуті, перекладені або мають деформацію. Крім того, алгоритм зіставлення має спеціальний режим для зіставлення різномасштабних записів відбитків пальців, а також опціональне зіставлення дзеркальних відбитків пальців.

– Адаптивний алгоритм фільтрації зображення усуває шуми, розриви та застряглі виступи та надійно виділяє дрібниці навіть із найнижчої якості відбитків менш ніж за 1 секунду.

Механізм вилучення шаблонів обличчя та зіставлення MegaMatcher:

– Толерантність до положення обличчям забезпечує рівень зручності реєстрації. MegaMatcher дозволяє повертати голову на 360 градусів. Нахил голови може бути до 15 градусів у кожному напрямку від фронтального положення. Кут повороту голови може становити до 45 градусів у кожному напрямку від фронтального положення. Для отримання додаткової інформації див. технічні характеристики.

– Надійне розпізнавання обличчя забезпечує точну реєстрацію з камер, веб-камер і різноманітних відсканованих документів; обличчя можуть бути зареєстровані зі сканованих сторінок паспорта чи інших видів документів. Якщо у відео або на зображенні є кілька облич, їх можна зареєструвати та обробляти одночасно. За бажанням можна визначити стать людини, риси обличчя та основні емоції. Крім того, частково закриті обличчя (тобто особи в масках або респіраторах) можна розпізнати без окремої реєстрації.

– Розпізнавання ознак обличчя. MegaMatcher можна налаштувати для виявлення певних атрибутів під час виділення обличчя – усмішки, відкритого рота, закритих очей, окулярів, темних окулярів, бороди та вусів.

– Оцінка віку. MegaMatcher може додатково оцінити вік людини, аналізуючи виявлене обличчя на зображенні.

– Виявлення живості обличчя.

Звичайну систему ідентифікації обличчя можна обдурити, розмістивши фотографію перед камерою. MegaMatcher здатний запобігти таким порушенням безпеки, визначаючи, чи є обличчя у відеопотоці чи окремому кадрі «живим» чи фотографією. Виявлення живості може виконуватися в пасивному режимі, коли механізм оцінює певні риси обличчя, і в активному режимі, коли механізм оцінює реакцію користувача на виконання дій, таких як моргання або рухи головою.

Щоб отримати докладніші відомості, перегляньте рекомендації щодо визначення живості обличчя.

– Запис біометричного шаблону може містити кілька зразків обличчя однієї особи. Ці зразки можуть бути зареєстровані з різних джерел і в різний час, що дозволяє покращити роботу користувача під час зіставлення. Наприклад, особа може бути зареєстрована з окулярами та без окулярів або з різними типами окулярів; з бородою чи вусами та без них тощо.

Механізм вилучення голосових шаблонів і відповідності MegaMatcher:

– Залежна від тексту система підбору голосу визначає, чи відповідає зразок голосу шаблону, витягнутому з певної фрази. Під час реєстрації від особи, яка реєструється, запитується одна або кілька фраз. Пізніше цю людину можуть попросити вимовити певну фразу для перевірки. Цей метод забезпечує захист від використання таємно записаної випадкової фрази цієї особи.

– Двофакторна автентифікація за допомогою фрази-паролю виконується, коли людину просять сказати унікальну фразу (наприклад, фразу-пароль або відповідь на «секретне запитання», яке знає лише зареєстрована особа). Загальна безпека системи підвищується, оскільки перевіряються автентичність голосу та пароль.

– Незалежний від тексту механізм відповідності голосу використовує різні фрази для реєстрації та розпізнавання користувачів. Цей спосіб більш зручний, оскільки не вимагає від кожного користувача запам'ятовувати парольну фразу. Його можна поєднати з текстозалежним алгоритмом для швидшого незалежного від тексту пошуку з подальшою перевіркою фрази за допомогою більш надійного текстозалежного алгоритму.

– Автоматичне визначення голосової активності. Механізм здатний визначити, коли користувачі починають і закінчують говорити.

– Виявлення живості. Система може вимагати від кожного користувача зареєструвати набір унікальних фраз. Пізніше користувачеві буде запропоновано вимовити певну фразу із зареєстрованого набору. Таким чином система може

гарантувати перевірку живої людини (на відміну від самозванця, який використовує запис голосу).

– Кілька голосових записів з тією самою фразою можуть зберігатися для підвищення надійності розпізнавання мовця. Певні природні варіації голосу (наприклад, хрипкий голос) або зміни навколишнього середовища (наприклад, в офісі та на природі) можна зберегти в одному шаблоні.

Механізм вилучення шаблону ірису MegaMatcher і відповідності:

– Перевірена надійність NIST IREX. Механізм зіставлення райдужної оболонки MegaMatcher заснований на VeriEye, визнаному NIST одним із найнадійніших доступних алгоритмів розпізнавання райдужної оболонки.

– Швидке зіставлення. Швидкість зіставлення райдужної оболонки становить до 200 000 порівнянь за секунду на одному ПК. Для отримання додаткової інформації див. технічні характеристики.

– Надійне виявлення райдужної оболонки. Райдужка виявляється навіть за наявності перешкод для зображення, візуального шуму та/або різного рівня освітлення. Усуваються відблиски світла, засмічення повік і вій. Також приймаються зображення зі звуженими повіками або очима, які дивляться вбік.

– Автоматичне виявлення та корекція чергування забезпечує максимальну якість шаблонів характеристик райдужної оболонки із рухомих зображень райдужної оболонки.

– Правильна сегментація райдужної оболонки досягається, навіть якщо ідеальні кола не вдаються, центри внутрішніх і зовнішніх меж райдужної оболонки відрізняються, межі райдужної оболонки точно не є колами і навіть не еліпсами або межі райдужної оболонки здаються ідеальними колами.

– Визначення якості зображення райдужної оболонки. Оцінку якості зображення можна використовувати під час реєстрації райдужної оболонки, щоб переконатися, що лише шаблон райдужної оболонки найкращої якості зберігатиметься в базі даних. Кут повороту можна визначити за зображенням райдужної оболонки для прийняття подальших рішень щодо прийняття

зображення для реєстрації. Також до зарахування можуть бути відхилені райдужні оболонки, закриті косметичними (декоративними) контактними лінзами з художніми зображеннями або зміною кольору.

– Виявлення живості. Захоплену райдужну оболонку можна проаналізувати, чи є вона «живою» чи підробкою, щоб запобігти порушенню безпеки, якщо розмістити фотографію перед камерою або надіти контактні лінзи з підробленою текстурою райдужної оболонки.

– Автоматичне визначення положення діафрагми. Алгоритм здатний розділяти зображення лівої та правої райдужної оболонки.

Безкоштовний SDK перевірки відбитків пальців

Безкоштовний SDK для перевірки відбитків пальців дозволяє розробляти біометричні програми, які перевіряють відсканований відбиток пальця з попередньо збереженим відбитком пальця (співставлення 1 до 1). SDK містить біометричний механізм, який розроблений і оптимізований для перевірки відбитків пальців. Механізм сумісний із біометричною технологією VeriFinger SDK, яка пропонує набагато розширеніші можливості для розробників програмного забезпечення.

Безкоштовний SDK для перевірки відбитків пальців не дозволяє розробляти програми, які зчитують зображення відбитків пальців із файлів або виконують ідентифікацію за відбитками пальців (зіставлення 1 до всіх). Кількість збережених шаблонів відбитків пальців обмежена 10 записами в базі даних програми. VeriFinger SDK дозволяє розробляти широкий спектр біометричних програм, які ідентифікують відбитки пальців, взяті зі сканера відбитків пальців або файли зображень, із відбитками пальців, що зберігаються в базі даних (відповідність 1 до всіх). Крім того, VeriFinger SDK не має жодних обмежень щодо розміру бази даних відбитків пальців і дозволяє розробляти як автономні, так і мережеві програми, які використовують архітектуру клієнт-сервер.

VeriFinger SDK містить оболонку, яка дозволяє легко перейти від Free Fingerprint Verification SDK до VeriFinger SDK

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка додатків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмувані засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконий режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконі й багатовіконі додатки, а також додатки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до

якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої додатки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-додатки на базі об'єктно-орієнтованного підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей.

Одною з основних переваг роботи з MFC є можливість багаторазового використання того самого коду. В зв'язку з тим, що бібліотека містить багато елементів, загальних для всіх Windows-додатків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

Оскільки MFC являє собою набір класів, написаних мовою C++, тому програми, написані з використанням MFC, повинна бути в той же час програмами на C++. Для цього необхідно володіти відповідними знаннями. Для початку необхідно вміти створювати власні класи, розуміти принципи спадкування й уміти перевизначати віртуальні функції. Хоча програми, що використовують бібліотеку MFC, звичайно не містять занадто специфічних елементів з арсеналу C++, для їхнього написання проте потрібні солідні знання в даній області.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У бакалаврському проекті розробляється програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF. Тому розглянемо цей метод більш детально.

Завдання розпізнавання образів дотепер не вирішено у повному обсязі. Однак, у рамках істотних обмежень, є методи, що дозволяють наблизиться до її рішення.

Серед різних родинних методів, був обраний для розгляду метод Speeded Up Robust Features (SURF), оскільки він є одним з найефективніших і швидких сучасних алгоритмів. Крім того, SURF є розповсюдженим методом, його реалізації є в багатьох математичних бібліотеках.

Проблематика

Коли ми дивимося на навколишніх людей, предмети, природу, ми не усвідомлюємо який обсяг роботи проробляє наш мозок, що б обробити весь потік візуальної інформації. Нам не важко буде знайти знайому нам людину на фотографії, або відрізнити будинок від пам'ятника. Здавалося б, наші комп'ютери відмінно можуть зберігати величезні обсяги інформації, картинки, відео й аудіо файли. Що заважає їм з такою ж легкістю знайти біопараметричні ознаки. Цьому перешкоджає ряд моментів, які ми тут і перелічимо:

– Масштаб. Зображення мають різний масштаб. Предмети, які ми сприймаємо як однакові, насправді займають різну площу на різних зображеннях.

– Місце. Об'єкт, що цікавить нас, може перебувати в різних місцях зображення.

– Тло й перешкоди. Предмет, що ми сприймаємо як щось окреме, на зображені ніяк не виділений, і перебуває на тлі інших предметів. Крім того,

зображення не ідеально й може бути піддано всякого роду перекручуванням і перешкодам.

– Проекція, обертання й кут огляду. Зображення є лише двовимірною проекцією нашого тривимірного миру. Тому поворот об'єкта й зміна кута огляду кардинальним образом впливають на його двовимірну проекцію – зображення. Той самий об'єкт може давати зовсім різну картинку, залежно від повороту або відстані до нього.

Список можна було б продовжувати ще довго. Але ми не будемо цього робити.

Отже, надані два зображення, одне з них будемо вважати зразком, інше – сценою. Завдання зводиться до визначення факту наявності зразка на сцені, і до його локалізації. При цьому зразок на сцені може:

– мати інший масштаб;

– бути повернуть у площині зображення;

– бути в довільному місці сцени;

– може бути зашумлений, видний не повністю, частково закритий іншими предметами;

– може мати відмінну від зразка яскравість і контраст;

– його може не бути зовсім.

Можливі, звичайно, і інші варіанти. Однак ми будемо вирішувати тільки перераховані пункти. Наприклад, ми не будемо розглядати обертання об'єкта навколо довільної осі.

Шляхи рішення завдання

Не відволікаючись на різні підходи до рішення обкреслених вище проблем порівняння образів, виберемо один з них.

Найпростіше й тривіальне рішення полягає в наступному: візьмемо зразок у різних масштабах, повернемо його на всілякі кути, переберемо всілякі місця на сцені, і будемо всі ці шаблони попіксельно порівнювати зі сценою.

Рішення, як нескладно зрозуміти, гарне, але нереалізоване. І от чому. Нехай зразок і сцена мають типові розміри – порядку сотень пікселів по вертикалі й горизонталі. Порахувавши загальне число всіляких шаблонів, їхніх поворотів, масштабів і локалізації, а також помноживши на число операцій попіксельного порівняння, одержимо біля трильйона (10^{12}) операцій для пошуку й локалізації зразка на сцені.

Крім того, безпосереднє порівняння зразка зі сценою може дати поганий результат, через шуми, перекручування, заслонення, об'єкти тла.

Тому виділимо на зразку якісь ключові точки й невеликі ділянки навколо них. Ключовою точкою будемо вважати таку точку, що має якісь ознаки, що істотно відрізняють її від основної маси точок. Наприклад, це можуть бути краї ліній, невеликі кола, різкі перепади освітленості, кути й т.д. Припускаючи, що ключові точки присутні на зразку завжди, те можна пошук зразка звести до пошуку на сцені ключових точок зразка. А оскільки ключові точки сильно відрізняються від основної маси точок, те їхнє число буде істотно менше, ніж загальне число точок зразка.

У цілому, принцип вибору ключових точок не важливий. Головне що б їх було не занадто багато й вони були присутні на зображені зразка завжди.

Чим менше ділянка, тим менше на нього впливають великомасштабні перекручування. Так, якщо об'єкт у цілому, підданий ефекту перспективи (тобто ближній край об'єкта має більший видимий розмір, чим далекий), то для малої його ділянки явищем перспективи можна зневажити й замінити на зміну масштабу. Аналогічно, невеликий поворот об'єкта навколо деякої осі може сильно змінити картинку об'єкта в цілому, але малі ділянки зміняться мало-мало.

Крім того, якщо частина об'єкта виходить за край зображення або закрита, те невеликі ділянки навколо частини ключових точок будуть видні цілком, що також дозволяє їх легше ідентифікувати. А ще, якщо малі області лежать цілком усередині шуканого об'єкта, то на них не роблять ніякого впливу об'єкти тла.

З іншого боку, ділянка навколо ключової точки не повинна бути занадто малою. Дуже малі ділянки несуть занадто мало інформації про зображення й з більшою ймовірністю можуть випадково збігатися між собою.

Огляд методу SURF

SURF вирішує два завдання – пошук особливих точок зображення й створення їхніх дескрипторів, інваріантних до масштабу й обертання. Це значить, що опис ключової точки буде однаковий, навіть якщо зразок змінить розмір і буде повернений (тут і далі ми будемо говорити тільки про обертання в площині зображення). Крім того, сам пошук ключових точок теж повинен мати інваріантність. Так, що б повернений об'єкт сцени мав той же набір ключових точок, що й зразок.

Метод шукає особливі точки за допомогою матриці Гессе. Детермінант матриці Гессе (т.зв. гессіан) досягає екстремума в точках максимальної зміни градієнта яскравості. Він добре детектує плями, кути й краї ліній.

Гессіан інваріантний щодо обертання. Але не інваріантний масштабу. Тому SURF використовує різномасштабні фільтри для знаходження гессіанів.

Для кожної ключової точки вважається напрямок максимальної зміни яскравості (градієнт) і масштаб, узятий з масштабного коефіцієнта матриці Гессе.

Градієнт у точці обчислюється за допомогою фільтрів Хаара.

Після знаходження ключових точок, SURF формує їхні дескриптори. Дескриптор являє собою набір з 64 (або 128) чисел для кожної ключової точки. Ці числа відображають флуктуації градієнта навколо ключової точки. Оскільки ключова точка являє собою максимум гессіана, те це гарантує, що в околиці точки повинні бути ділянки з різними градієнтами. Таким чином, забезпечується дисперсія (розходження) дескрипторів для різних ключових точок.

Флуктуації градієнта околиць ключової точки вважаються щодо напрямку градієнта навколо точки в цілому (по всій околиці ключової точки). Таким чином, досягається інваріантність дескриптора щодо обертання. Розмір же області, на якій уважається дескриптор, визначається масштабом матриці Гессе, що

забезпечує інваріантність щодо масштабу. Флуктуації градієнта також зважають на допомогу фільтра Хаара.

Інтегральне подання

Для ефективного обчислення фільтрів Гессе й Хаара – використовується інтегральне подання зображень.

Якщо коротко, то інтегральне подання є матрицею, розмірність якої збігається з розмірністю вихідного зображення, а елементи розраховуються за формулою:

$$II(x,y) = \sum_{i=0,j=0}^{i\le x, j\le y} I(i,j), \tag{3.1}$$

де $I(i,j)$ – яскравість пікселів вихідного зображення.

Маючи інтегральну матрицю можна дуже швидко обчислювати суму яскравістей пікселів довільних прямокутних областей зображення, за формулою:

$$\text{SumOfRect(ABCD)} = II(A) + II(3) - II(B) - II(D),$$

де ABCD – прямокутник, що цікавить нас.

Обчислення матриці Гессе

Виявлення особливих точок в SURF засновано на обчислені детермінанта матриці Гессе (гессіана).

Матриця Гессе для двовимірної функції і її детермінант визначається в такий спосіб:

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$det(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 \tag{3.2}$$

.

Значення гессіана використовується для знаходження локального мінімуму або максимуму яскравості зображення. У цих точках значення гессіана досягає екстремума.

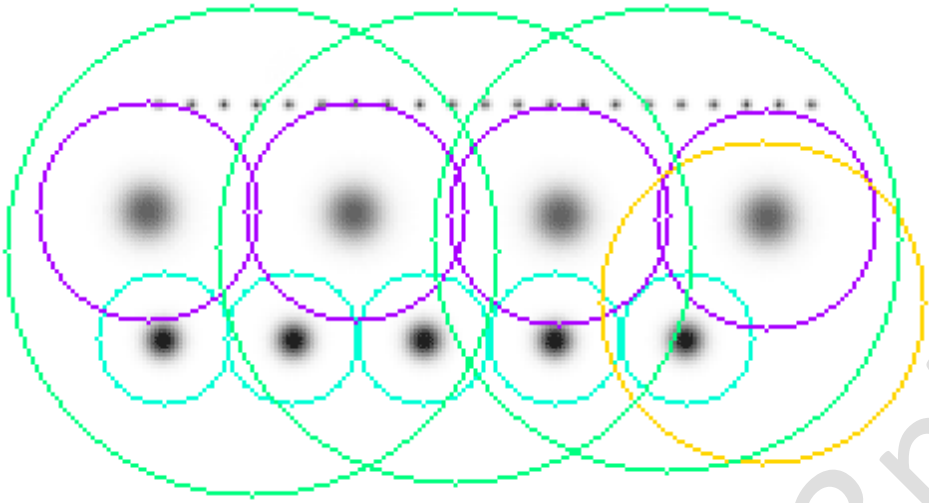

Рисунок 3.1 – Особливі точки (обкреслені кольоровими колами) являють собою локальні екстремуми яскравості зображення

Дрібні точки не розпізнані як особливі, через граничне відсікання по величині гессіана.

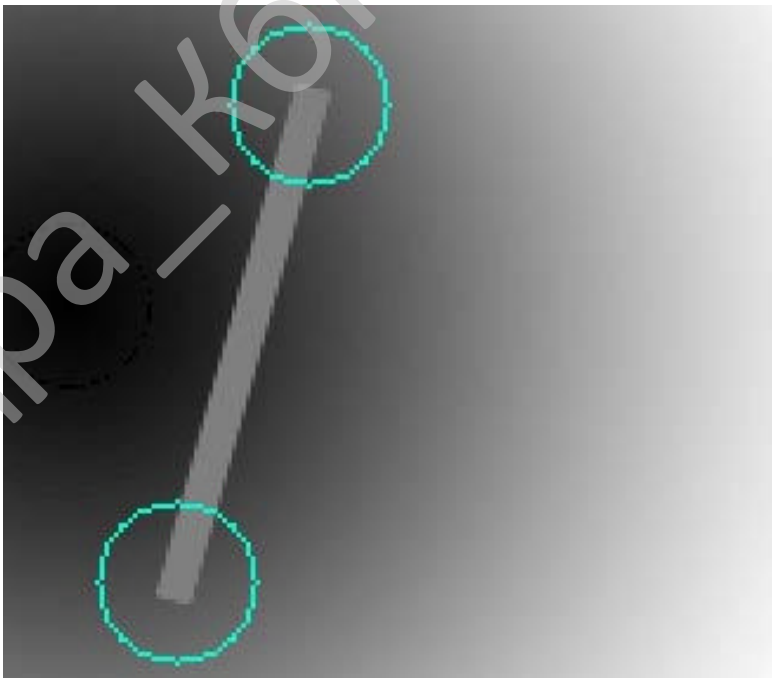

Рисунок 3.2 – Кінці відрізка, розпізнані як ключові точки, за допомогою матриці Гессе

Теоретично, обчислення матриці Гессе зводиться до знаходження Лапласіана Гауссіан. По суті, елементи матриці Гессе обчислюються як згортка (сума добутків) пікселів зображення на фільтри, зображені на рисунку 3.3.

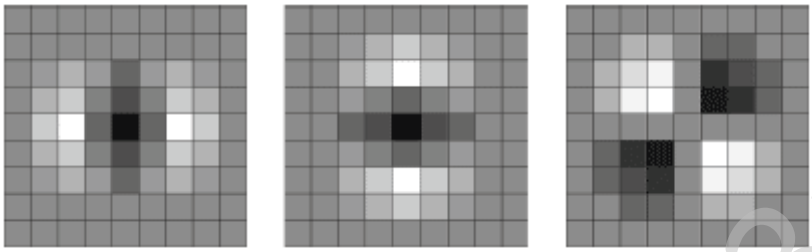

Рисунок 3.3 – Дискретизировані фільтри для знаходження чотирьох елементів матриці Гессе

На рисунку зображені дискретизировані фільтри для знаходження чотирьох елементів матриці Гессе (четвертий – збігається із третім, оскільки матриця Гессе симетрична). Фільтри мають просторовий масштаб 9x9 пікселів. Темні ділянки відповідають негативним значенням фільтра, світлі – позитивним.

Однак, SURF не використовує лапласіан гауссіани в тому виді, що зображений на рисунку. По-перше, за твердженням авторів, дискретизировані лапласіан гауссіани має досить великий розкид значення детермінанта, при обертані зразка (нагадаємо, що в ідеалі гессіан повинен бути інваріантний до обертання). Особливо детермінант «просідає» у районі повороту на 45 градусів. А по-друге, і цей головне, фільтр для лапласіана гауссіани має безперервний характер. Майже всі пікселі фільтра мають різні величини яскравості. А це не дозволяє ефективно використовувати такий потужний механізм розрахунку, як інтегральну матрицю зображення.

Тому SURF використовує бінарізовану апроксимацію лапласіана гауссіан (автори назвали його Fast-Hessian).

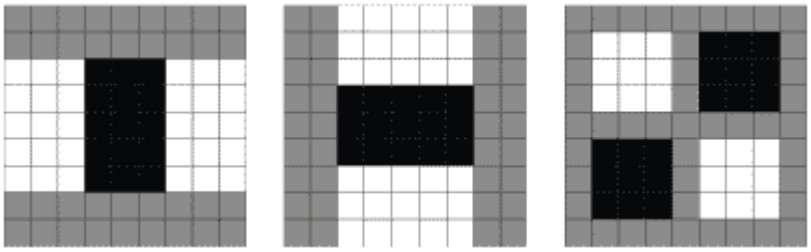

Рисунок 3.4 – На рисунку зображені фільтри, використовувані для знаходження матриці Гессе в SURF

Білі області відповідають значенню +1, чорні -2 (на третьому фільтрі -1), сірі – нульові. Просторовий масштаб – 9x9 пікселів.

Цей фільтр більше стійкий до обертання, і його можна ефективно обчислити за допомогою інтегральної матриці.

Таким чином, в SURF, гессіан обчислюється так:

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \quad (3.3)$$

,

де D_{xx}, D_{yy}, D_{xy} – згортки по фільтрах, зображеним на рисунку вгорі. Коефіцієнт 0.9 має теоретичне обґрунтування, і коректує наближений характер обчислень.

Отже, для знаходження особливих точок, SURF пробігається по пікселям зображення й шукає максимум гессіана. Спосіб знаходження локального максимуму гессіана ми розглянемо пізніше. У методі задається граничне значення гессіана. Якщо обчислене значення для пікселя вище порога – піксель розглядається як кандидат на ключову точку.

Отут ще корисно помітити наступне. Оскільки гессіан є похідній, і залежить тільки від перепаду яскравості, але не від абсолютного її рівня, то він інваріантний стосовно зрушення яскравості зображення. Таким чином, зміна рівня висвітлення зразка не впливає на виявлення ключових точок.

Крім того, властивості гессіана такі, що він досягає максимуму, як у точці білої плями на чорному тлі, так і чорної плями на білому тлі. Таким чином, метод виявляє й темні, і світлі особливості зображення.

Шкали

Як ми вже відзначали, гессіани не інваріантні щодо масштабу. Це значить, що для того самого пікселя, гессіан може мінятися при зміні масштабу фільтра. Рішення цієї проблеми тільки одне – перебирати різні масштаби фільтрів і по черзі їх застосовувати до даного пікселя.

З міркувань симетрії й дискретизації, розмір фільтра Fast-Hessian не може приймати довільні значення. Припустимі розміри цього фільтра такі (починаючи з мінімального): 9, 15, 21, 27 і так далі, із кроком 6. Однак, на практиці, поступово збільшувати розмір фільтра на 6 – не вигідно, тому що для великих масштабів крок 6 виявляється занадто дрібним, а фільтри – надлишковими. Тому (і по деяких інших причинах), SURF розбиває вся безліч масштабів на так звані октави. Кожна октава покриває певний інтервал масштабів, і має свій характерний розмір фільтра.

При цьому якби на октаву доводився тільки один фільтр, це було б занадто грубим наближенням. Крім того, ми б не могли знайти локальний максимум гессіана, серед різних масштабів, у різних октавах. Адже та сама точка може мати кілька локальних максимумів гессіана, у різних масштабах.

Якщо ми будемо шукати максимум серед всіх гессіанів, по всіх масштабах, то ми б знайшли тільки один з максимумів, у той час як їх може бути кілька. Один – в одному масштабі, іншої – в іншому.

Виходячи з перерахованого, октава містить не один фільтр, а чотири фільтри, які добре покривають характерний масштаб октави.

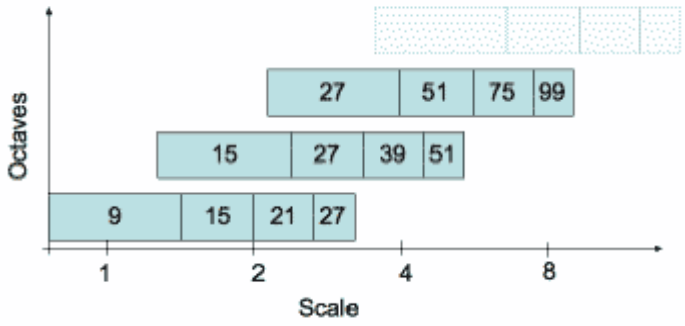

Рисунок 3.5 – Перші три октави методу SURF

Цифри в прямокутниках показують розмір фільтра Fast-Hessian. Логарифмічна шкала знизу – показує масштаби, що покриваються октавами.

Крок розміру фільтра в першій октаві – становить 6, у другий – 12, у третьої – 24 і так далі.

Як бачимо, октави значно перекриваються один одним. Це збільшує надійність знаходження локальних максимумів.

Виникає питання, а скільки властиво октав досить для покриття безлічі особливих точок різних масштабів. Теоретично, масштаби нескінчені, однак у реальних зображеннях, вони цілком кінцеві, і основна маса зосереджена в інтервалі від 1 до 10 (по даним авторів методу). Для покриття цього діапазону досить чотирьох октав. Плюс додається одна або дві октави для покриття більших масштабів. Разом, використовується 5-6 октав. Теоретично, цього цілком достатньо для покриття всіляких масштабів на зображені 1024x768 пікселів.

Знаходження локального максимуму гессіана

Для знаходження локального максимуму гессіана, використовується так званий метод сусідніх точок 3x3x3.

Його зміст зрозумілий з рисунка нижче.

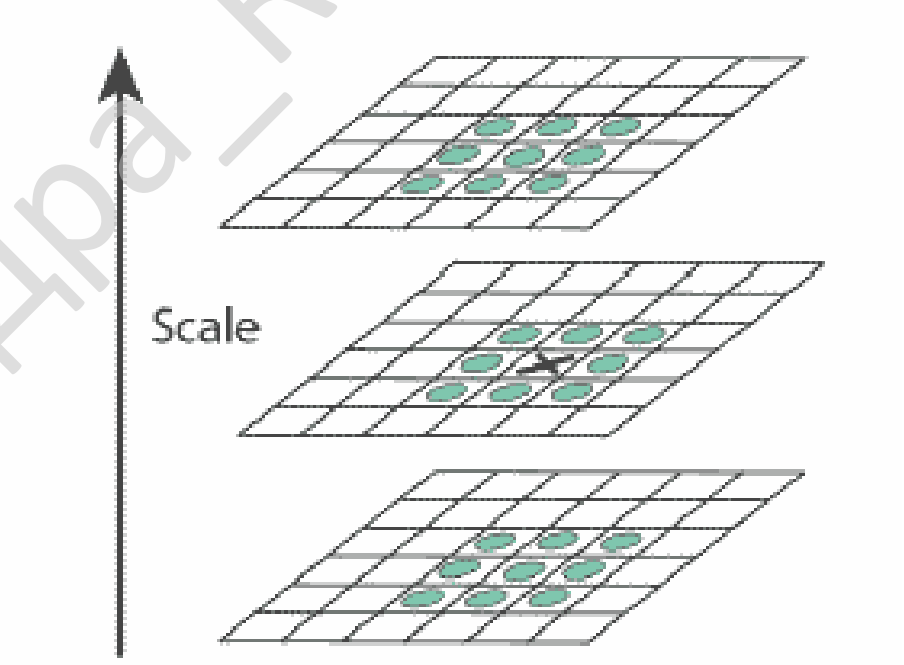

Рисунок 3.6 – Локальний максимуму гессіана

Піксель, позначений хрестиком уважається локальним максимумом, якщо його гессіан більше ніж у будь-якого його сусіда в його масштабі, а також більше кожного із сусідів масштабом менше й масштабом більше (усього 26 сусідів). Виходячи з такого визначення локального максимуму, зрозуміло, що октава повинна містити не менш трьох фільтрів, інакше ми не зможемо визначити факт знаходження локального максимуму гессіана усередині октави. Відзначимо ще такий момент. Фільтри октави вважаються не для всіх пікселів підряд. Перша октава вважається для кожного другого пікселя зображення. Друга – для кожного четвертого, третя – для кожного восьмого й так далі. Зміст зрозумілий – дві точки з відстанню 2 не можуть містити більше одного максимуму масштабу 2, 3 або більше високих масштабів. Тому нема рації перебирати всі точки зображення, для знаходження максимуму масштабу 3, наприклад.

Подвоєння кроку пікселів для октав дозволяє заощаджувати при розрахунку фільтрів. Як ви напевно вже помітили, розміри фільтрів в октавах повторюються. Так, наприклад, фільтр розміром 27 є присутнім у трьох октавах. Отож, при обчисленнях, цей фільтр буде вважатися тільки для першої октави. Друг і третя – просто використовують розрахунки першої октави. А подвоєння кроку пікселів гарантує, що точки в які потрібно вважати гессіан, уже були перелічені попередньою октавою.

Тому, незважаючи на те, що октава містить чотири фільтри, насправді кожна октава (крім першої) уважає тільки два характерних для неї розміру, два інших – завжди можна взяти з попередніх октав. Перша ж октава змушена вважати всі чотири своїх фільтри.

Отже, після знаходження максимального гессіана методом сусідніх точок 3x3x3, ми знайшли піксель у якому цей максимум досягається. Однак, оскільки, октава перебирає не всі точки зображення, те щирий максимум може не збігатися зі знайденим пікселем, а лежати десь поруч, у сусідніх пікселях.

Для знаходження точки щирого максимуму, використовується інтерполяція знайдених гессіанів куба 3x3x3 квадратичною функцією. Далі,

обчислюється похідна (методом кінцевих різниць сусідніх точок). Якщо вона близька до нуля – ми в точці щирого максимуму. Якщо похідна велика – зрушуємося убік її зменшення, і повторюємо ітерацію, доти поки похідна не стане менше заданого порога. Якщо в процесі ітерацій ми відходимо від початкової точки занадто далеко, то це вважається помилковим максимумом, і точка більше не вважається особою.

Знаходження орієнтації особливої точки

Для інваріантності обчислення дескрипторів особливої точки, які будуть розглянуті нижче, потрібно визначити переважну орієнтацію перепадів яскравості в особливій точці. Це поняття близько до поняття градієнта, але SURF використовує небагато інший алгоритм знаходження вектора орієнтації.

Спочатку, обчислюються точкові градієнти в пікселях, сусідніх з особливою точкою. Для розгляду беруться пікселі в окружності радіуса 6s навколо особливої точки. Де s – масштаб особливої точки. Для першої октави беруться точки з околиці радіусом 12.

Для обчислення градієнта, використовується фільтр Хаара. Розмір фільтра береться рівним 4s, де s – масштаб особливої точки. Вид фільтрів Хаара показаний на картинці:

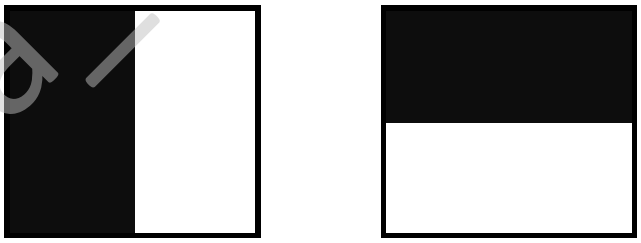

Рисунок 3.7 – Фільтри Хаара

Чорні області мають значення -1, білі +1. Фільтри Хаара дають точкове значення перепаду яскравості по осі X і Y відповідно. Оскільки фільтри Хаара мають прямокутну форму, їхні значення легко зважають на допомогу інтегральної матриці. Для розрахунку одного фільтра довільного розміру потрібно всього 6 операцій.

Значення вейвлета Хаара d і d для кожної точки множаться на вагу й запам'ятовуються в масиві. Вага визначається як значення гауссіани із центром в особливій точці й сигмою рівної 2s. Зважування на гауссіане необхідно для відсікання випадкових перешкод на далекі від особливої точки відстанях.

Далі, всі знайдені значення d і d, умовно наносяться у вигляді точок на площину, як показано на рисунку 3.8.

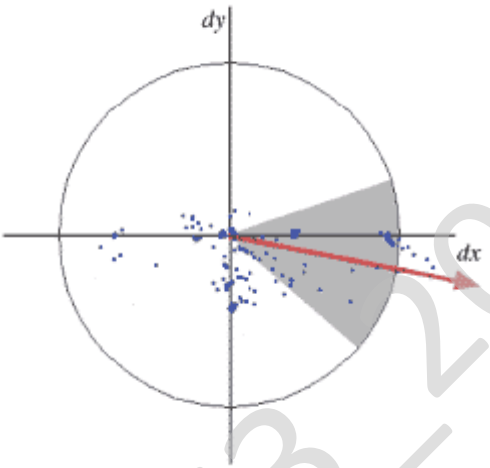

Рисунок 3.8 – Всі знайдені градієнти у вигляді точок у просторі dXd

Далі, береться кутове вікно (показане сірим на рисунку) розміром π/3, і обертається навколо центра координат. Вибирається таке положення вікна, при якому довжина сумарного вектора для точок, що потрапили у вікно, – максимальна. Обчислений у такий спосіб вектор нормується й приймається як пріоритетний напрямок в області особливої точки.

Маніпуляції з вікном потрібні для зменшення впливу шумових точок.

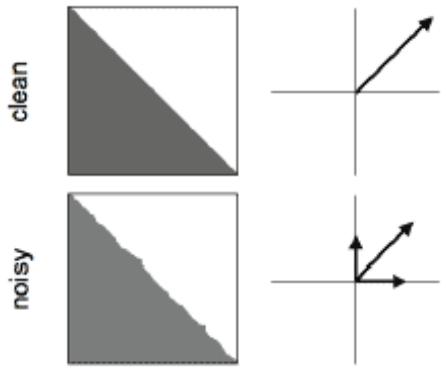

Рисунок 3.9 – Градієнт при ідеальному краї, і при краї із шумом

Як бачимо, шум дає додаткові градієнти в напрямках, що не збігаються з напрямком основного градієнта. Використання вікна дозволяє відітнути такі шумові точки, і більш точно обчислити щирий градієнт.

Відзначимо, що не завжди потрібна інваріантність дескрипторів щодо обертання. Метод SURF має модифікацію, у якій орієнтація особливих точок не розраховується. Така модифікація дозволяє надійно ідентифікувати точки, повернені не більше ніж на ±15 градусів.

Обчислення дескриптора особливої точки

Дескриптор являють собою масив з 64 (у розширеній версії 128) чисел, що дозволяють ідентифікувати особливу точку. Дескриптори однієї й тої ж особливої точки на зразку й на сцені повинні приблизно збігатися. Метод розрахунку дескриптора такий, що він не залежить від обертання й масштабу.

Для обчислення дескриптора, навколо особою точки формується прямокутна область, що має розмір 20s, де s – масштаб у якому була знайдена особлива точка. Для першої октави, область має розмір 40x40 пікселів. Квадрат орієнтується уздовж пріоритетного напрямку, обчисленого для особливої точки.

Дескриптор рахується як опис градієнта для 16 квадрантів навколо особою точки.

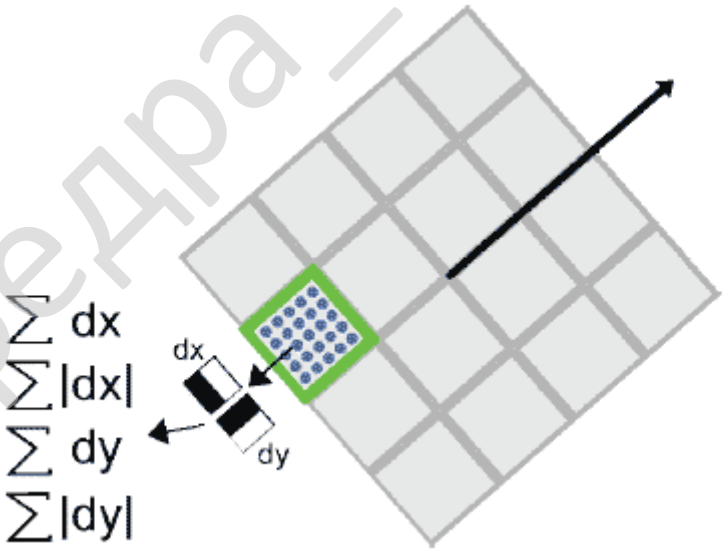

Рисунок 3.10 – Обчислення дескриптора особливої точки

Далі, квадрат розбивається на 16 більше дрібних квадрантів, як показано на рисунку. У кожному квадранті береться регулярна сітка 5x5 і для точки сітки шукається градієнт, за допомогою фільтра Хаара. Розмір фільтра Хаара береться рівним 2s, і для першої октави становить 4x4.

Слід зазначити, що при розрахунку фільтра Хаара, зображення не повертається, фільтр уважається у звичайних координатах зображення. А от отримані координати градієнта (d,d) повертаються на кут, що відповідає орієнтації квадрата.

Разом, для обчислення дескриптора особливої точки, потрібно обчислити 25 фільтрів Хаара, у кожному з 16 квадрантів. Разом, 400 фільтрів Хаара. З огляду на, що на фільтр потрібно 6 операцій, виходить, що дескриптор обійдеться мінімум в 2400 операцій.

Після знаходження 25 точкових градієнта квадранта, обчислюються чотири величини, які властиво і є компонентами дескриптора:

$$\sum dX, \sum |dX|, \sum dY, \sum |dY|.$$

Дві з них є просто сумарний градієнт по квадранту, а дві інших – сума модулів точкових градієнтів.

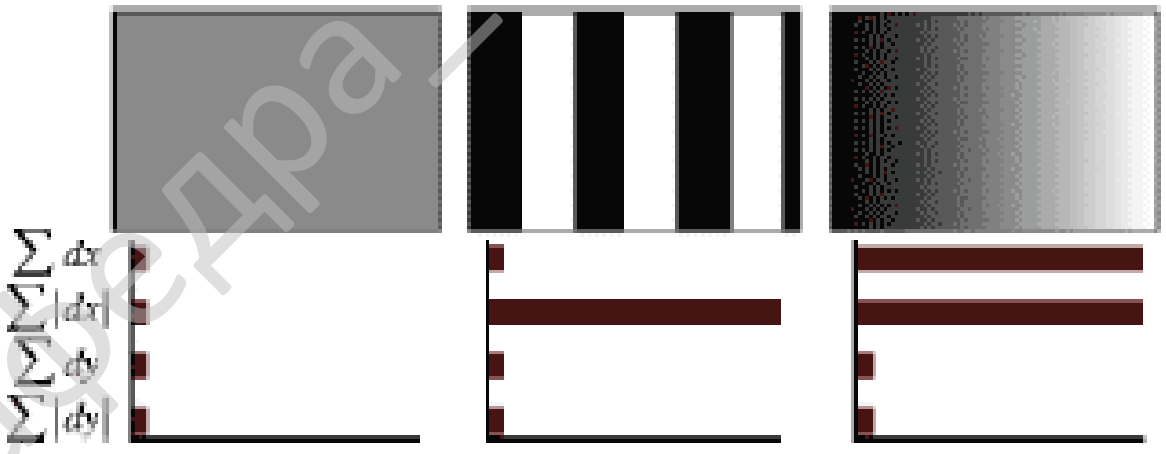

Рисунок 3.11 – Поводження дескриптора для різних ділянок зображень

Рисунок показує поводження дескриптора для різних зображень. Для рівномірних областей – всі значення близькі до нуля. Для повторюваних вертикальних смужок – всі величини, крім другої близькі до нуля. При збільшені яскравості в напрямку осі X, два перші компоненти мають більші значення.

Чотири компоненти на кожний квадрант, і 16 квадрантів, дають 64 компонента дескриптора для всієї області особливої точки. При занесені в масив, значення дескрипторів зважуються на гауссіану, із центром в особливій точці й із сигмою 3.3s. Це потрібно для більшої стійкості дескриптора до шумів у вилучені від особливої точки областях.

Плюс до дескриптора, для опису точки використовується знак сліду матриці Гессе, тобто величина sign(Dxx+Dyy). Для світлих точок на темному тлі, слід негативний, для темних точок на світлому тлі – позитивний. Таким чином, SURF розрізняє світлі й темні плями.

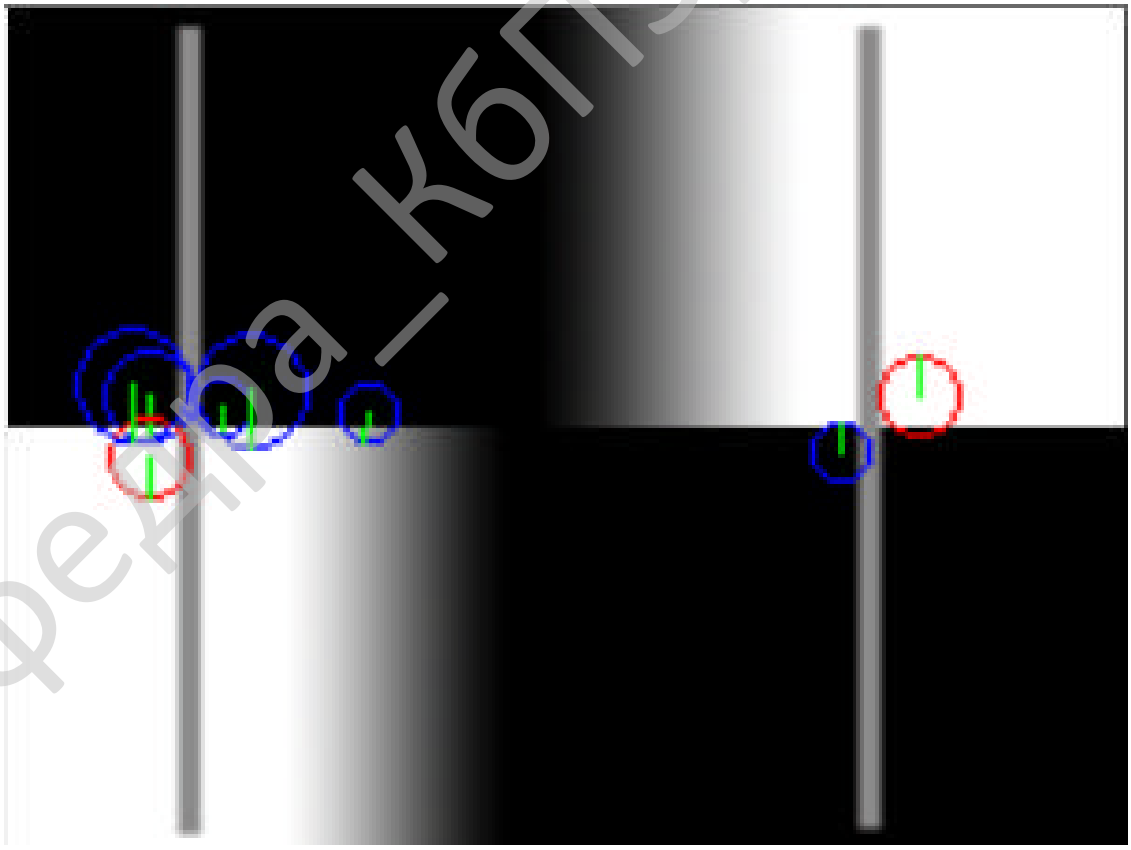

Рисунок 3.12 – Особливі точки зображення

Зелена лінія показує характерний напрямок для особливої точки. Синій колір окружності показує позитивний слід матриці Гессе, червоний – негативний слід.

Проведені тести показали, що метод поводиться цілком стабільно. Різке падіння відсотка розпізнавання при збільшені кута огляду на 45 градусів пов'язане з тим, що SURF не інваріантний до аффінних перетворень. Падіння при размитті й високого ступеня стиску JPEG пояснюється втратою інформації в зображені при цих перетвореннях. Гарний і стабільний результат спостерігається при обертані й зміні масштабу зображення. Непогано метод справляється й зі зміною яскравості зображення.

Недоліки методу

Потрібно відзначити, незважаючи на те, що SURF використовується для пошуку об'єктів на зображені, він сам працює не з об'єктами. SURF ніяк не виділяє об'єкт із тла. Він розглядає зображення як єдине ціле й шукає особливості цього зображення. При цьому особливості можуть бути як усередині об'єкта, так і на тлі, а також на точках границі об'єкта й тла. У зв'язку із цим, метод погано працює для об'єктів простої форми й без яскраво вираженої текстури. Усередині таких об'єктів, метод швидше за все не знайде особливих точок. Точки будуть знайдені або на границі об'єкта із тлом, або взагалі тільки на тлі. А це приведе до того, що об'єкт не зможе бути розпізнаний в іншому зображені, на іншому тлі.

3.2 Розробка структурної схеми

Біометричні сканери стають все більш досконалими. Ви навіть можете знайти біометричні дані в системах безпеки телефонів. Наприклад, технологія розпізнавання обличчя на iPhone X від Apple проектує 30 000 інфрачервоних точок на обличчя користувача для автентифікації користувача за шаблоном. За даними Apple, ймовірність помилкової ідентифікації з біометричними даними iPhone X становить один до мільйона.

Смартфон LG V30 поєднує в собі розпізнавання обличчя та голосу зі скануванням відбитків пальців і зберігає дані на телефоні для більшої безпеки. CrucialTec, виробник датчиків, пов'язує датчик серцевого ритму зі своїми сканерами відбитків пальців для двоетапної автентифікації. Це допомагає гарантувати, що клоновані відбитки пальців не можна використовувати для доступу до його систем.

Проблема полягає в тому, що біометричні сканери, включаючи системи розпізнавання обличчя, можна обдурити. Дослідники з Університету Північної Кароліни в Чапел-Хілл завантажили фотографії 20 добровольців із соціальних мереж і використали їх для створення 3D моделей їхніх облич. Дослідники успішно зламали чотири з п'яти перевірених систем безпеки.

Приклади клонування відбитків пальців є всюди. Один приклад із конференції з кібербезпеки Black Hat продемонстрував, що відбиток пальця можна надійно клонувати приблизно за 40 хвилин із матеріалу вартістю 10 доларів США, просто зробивши відбиток відбитка пальця у формувальному пластику чи воску для свічок.

Німецький комп'ютерний клуб Chaos підробив зчитувач відбитків пальців iPhone TouchID протягом двох днів після його випуску. Група просто сфотографувала відбиток пальця на скляній поверхні та використала його для розблокування iPhone 5s.

Біометрія – проблеми щодо ідентифікації та конфіденційності

Біометрична автентифікація зручна, але прихильники конфіденційності побоюються, що біометрична безпека підриває особисту конфіденційність. Викликає занепокоєння те, що особисті дані можуть збиратися легко і без згоди.

Розпізнавання облич є частиною повсякденного життя китайських міст, де воно використовується для звичайних покупок, а Лондон усіяний камерами відеоспостереження. Зараз Нью-Йорк, Чикаго та Київ підключають камери відеоспостереження у своїх містах до баз даних розпізнавання облич, щоб допомогти місцевій поліції боротися зі злочинністю. Розвиваючи технологію,

Університет Карнегі-Меллона розробляє камеру, яка може сканувати райдужну оболонку очей людей у натовпі з відстані 10 метрів.

У 2018 році в аеропорту Дубая було запроваджено розпізнавання облич, де мандрівників фотографують 80 камерами, коли вони проходять через тунель у віртуальному акваріумі.

Камери розпізнавання облич також працюють в інших аеропортах по всьому світу, в тому числі в Гельсінкі, Амстердамі, Міннеаполісі-Сент. Пол і Тампа. Усі ці дані мають десь зберігатися, що викликає побоювання постійного стеження та зловживання даними...

Питання безпеки біометричних даних

Більш актуальною проблемою є те, що бази даних особистої інформації є мішенню для хакерів. Наприклад, коли Управління управління персоналом США було зламано в 2015 році, кіберзлочинці втекли з відбитками пальців 5,6 мільйонів державних службовців, зробивши їх уразливими для крадіжки особистих даних.

Зберігання біометричних даних на пристрої, як-от TouchID або Face ID iPhone, вважається безпечнішим, ніж зберігання в постачальника послуг, навіть якщо дані зашифровані.

Цей ризик подібний до ризику бази даних паролів, у якій хакери можуть зламати систему та викрасти дані, які не захищені належним чином. Проте наслідки значно відрізняються. Якщо пароль зламано, його можна змінити. Біометричні дані за контрактом залишаються незмінними назавжди.

Способи захисту біометричної особистості

З ризиком для конфіденційності та безпеки біометричні системи повинні використовувати додаткові засоби захисту.

Неавторизований доступ стає складнішим, коли системи потребують кількох засобів автентифікації, таких як визначення життя (наприклад, блимання) і зіставлення закодованих зразків із користувачами в зашифрованих доменах.

Деякі системи безпеки також включають додаткові характеристики, такі як вік, стать і зріст, у біометричні дані, щоб перешкодити хакерам.

Хорошим прикладом є програма Aadhaar Управління унікальних ідентифікаційних номерів Індії. Започаткована в 2009 році програма багатоетапної автентифікації включає сканування райдужної оболонки ока, відбитки всіх 10 пальців і розпізнавання обличчя.

Ця інформація пов'язана з унікальною ідентифікаційною карткою, яка видається кожному з 1,2 мільярда жителів Індії. Невдовзі ця картка стане обов'язковою для всіх, хто отримує доступ до соціальних послуг в Індії.

Біометрія є хорошою заміною для імен користувачів у рамках стратегії двофакторної автентифікації. Це включає в себе:

– Щось, що ти є (біометрія).

– Щось у вас є (наприклад, апаратний маркер) або те, що ви знаєте (наприклад, пароль).

Двофакторна автентифікація є потужною комбінацією, особливо в умовах поширення пристроїв IoT. Завдяки посиленню захисту захищені інтернет-пристрої стають менш уразливими до витоку даних.

Крім того, використання менеджера паролів для зберігання будь-яких традиційних паролів може дати вам додатковий захист.

Висновки щодо біометрії

Таким чином, біометрія залишається все більш популярним способом перевірки особи для систем кібербезпеки.

Комбінований захист ваших фізичних або поведінкових підписів з іншими засобами автентифікації забезпечує один із найнадійніших відомих засобів захисту. На даний момент це принаймні краще, ніж використовувати символьний пароль як окрему перевірку.

Біометрична технологія пропонує дуже переконливі рішення для безпеки. Незважаючи на ризики, системи зручні та їх важко відтворити. Крім того, ці системи розвиватимуться ще дуже довго в майбутньому.

Для розробки програми були попередньо розроблені структурна схема роботи системи, структура формату передачі даних, функціональна схема роботи системи, діаграма процесів, а також блок-схеми алгоритму програми, розглянемо їх детально.

Створена автором система автентифікації за допомогою біопараметричних характеристик складається із двох частин – клієнтської частини на стороні користувача й серверної частини що перебуває віддалено в Інтернеті. Створена система універсальна. Застосовуючи стандартизовані бібліотеки Bioparametrs Aplication Programming Interface і використовуючи стандартизований формат передачі повідомлень користувач може використовувати будь-які біопараметричні сканери. При створені й тестуванні системи автор використовував два біопараметричні сканери, а саме – сканер відбитків пальців MorphoSmart 1300 Sagem для розпізнання папілярного рисунка пальця й планшетний сканер SignatureGem 1x5 розпізнавання підпису людини.

Клієнтська частина виконана у вигляді Active X додатка з застосуванням стандартизованої бібліотеки Bioparametrs Aplication Programming Interface 1.1 і застосовна з Internet Explorer версії 4.0 і вище, а також у будь-яких інших програмах, що підтримують роботу з Active X компонентами.

Серверна частина виконана у вигляді модуля COM і повністю сумісна з IIS сервером із застосуванням стандартної бази даних від корпорації Майкрософт – Microsoft Access з інтерфейсом ODBC.

Для подальшого розуміння роботи розробленої системи введемо наступну термінологію.

Навчання – процедура зняття серії відбитків того самого пальця або інших біопараметричних характеристик (підпис людини, форма руки, і т.ін.) з метою відбору деякої їхньої кількості з найбільше яскраво вираженими особливостями, чим ці особливості більше виділені тим краще якість зразка поняття прямо пов'язане із застосовуваними апаратними засобами зняття біопараметричних

характеристик. У деяких випадках зразок непридатний для розпізнавання через різні причини (наприклад у випадки зняття відбитків пальців забруднення).

Паспорт – еталонний зразок (перевірений 100% зразок) для ідентифікації особи.

Модель – математичний образ (пакет даних) відбитка пальця, одержуваного з біопараметричного сканера, а також біометричний ідентифікаційний запис і криптографічне додавання. Модель необхідний запис для розпізнавання користувача й надання йому прав.

Біометричний ідентифікаційний запис (BIR, BSP) – група даних з моделі відповідальних за біопараметричні характеристики (використовуваний сканер, якість зразка, опис тип даних і т.д.).

Криптографічне додавання – група даних з моделі відповідальних за автентифікацію.

Розпізнавання – процедура порівняння Моделі й Паспорти, результатом якої є вивід про їхню ідентичність і видачі коду перевірки.

Розглянемо докладніше особливості реалізації й взаємодії клієнтської частини із серверної зображеної на рисунку 3.13.

Розглянемо схему знизу нагору, при одержані з біопараметричного сканера даних на стороні клієнта розроблений Active X компонент використовуючи бібліотеку BIOAPI перевіряє якість отриманого зразка й формує біометричний ідентифікаційний запис (розділ 3.8) складається з 32 кілобайт інформації. Далі відбувається додавання криптографічного додавання (40 Кб) підтверджувального, що цей запис є записом користувача, формат запису стандартизований на території України. Після формування біометричного ідентифікаційного запису й криптографічного додавання відбувається остаточне формування моделі для перевірки користувача.

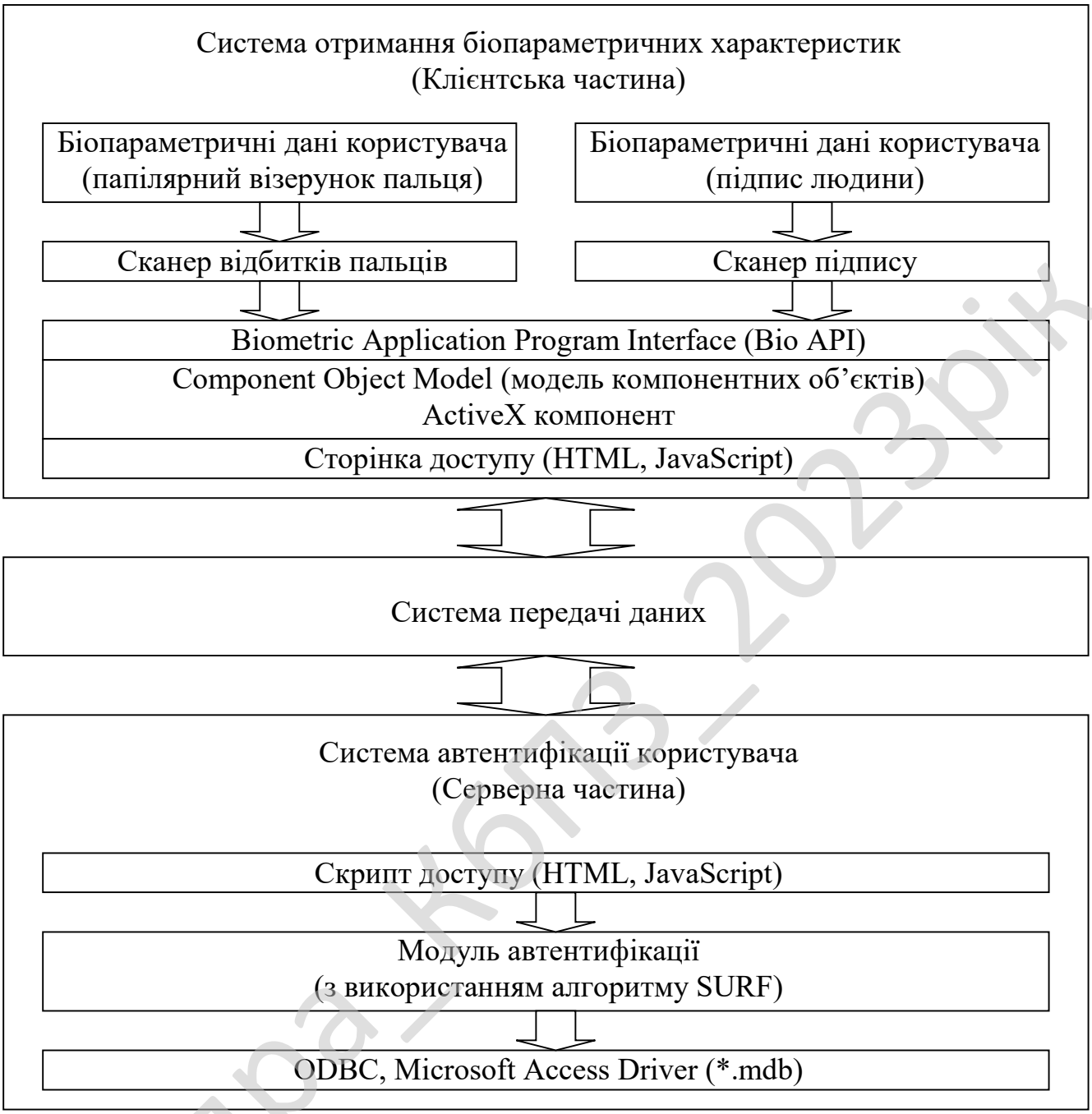

Рисунок 3.13 – Структурна схема роботи системи

Зі сторінки доступу модель передається через Інтернет на серверну частину системи де обробляється скриптом доступу й надходить в DCOM модуль автентифікації де відбувається перевірка на відповідність. При позитивному проходжені перевірки й добувані біометричного ідентифікаційного запису.

DCOM модуль витягає з Microsoft Access паспорт користувача. Відбувається біопараметричне зіставлення даних і повернення клієнтові коду результату.

Розглянемо необхідність застосування технологій компонента Active X і COM модулів при розробці клієнтської й серверної частини системи.

Як відоме написання Active компонент і COM на основі моделі компонентних об'єктів дозволяє вбудовувати розроблені програми в різні джерела, такі як HTML сторінки використовуючи JavaScrpt, PHP скріпти, Pearl скріпти й т.ін., що приводить до універсальності. Виходячи з поставленого завдання застосування даної технології доцільно. Знаючи всього CLSID Active X компонента, ми одержуємо волю вибору в застосуванні в різних джерелах. CLSID розробленого Active X об'єкта – "CLSID:7D6416 BB-4417-4B 92-B564-A39A474DC84E" (він унікальний).

Для функціонування серверної частини системи, побудованої із застосуванням COM модуля, не буде потрібно здобувати ніяких додаткових апаратних засобів. Це дуже зручно для проектів з невеликою клієнтською базою. У випадку більших вимог до швидкості й обсягу розпізнавання в передбачені технології нарощування продуктивності із застосуванням програмних засобів від Майкрософт. Безсумнівний плюс такого підходу – можливість поступового нарощування продуктивності системи в міру росту клієнтської бази, чим забезпечується оптимізація й мінімізація витрат на експлуатацію системи. Серверна частина через вибір COM модуля підтримує всі існуючі на даний момент технології масштабування й вирівнювання навантаження додатків від Майкрософт: Network Load Balancing, Component Load Balancing, Application Center, Object Pooling, JIT activation.

Структура формату передачі даних

Розглянемо формат передачі даних, які складаються з біометричного ідентифікаційного запису й українського криптографічного додавання, зображених на рисунку 3.14.

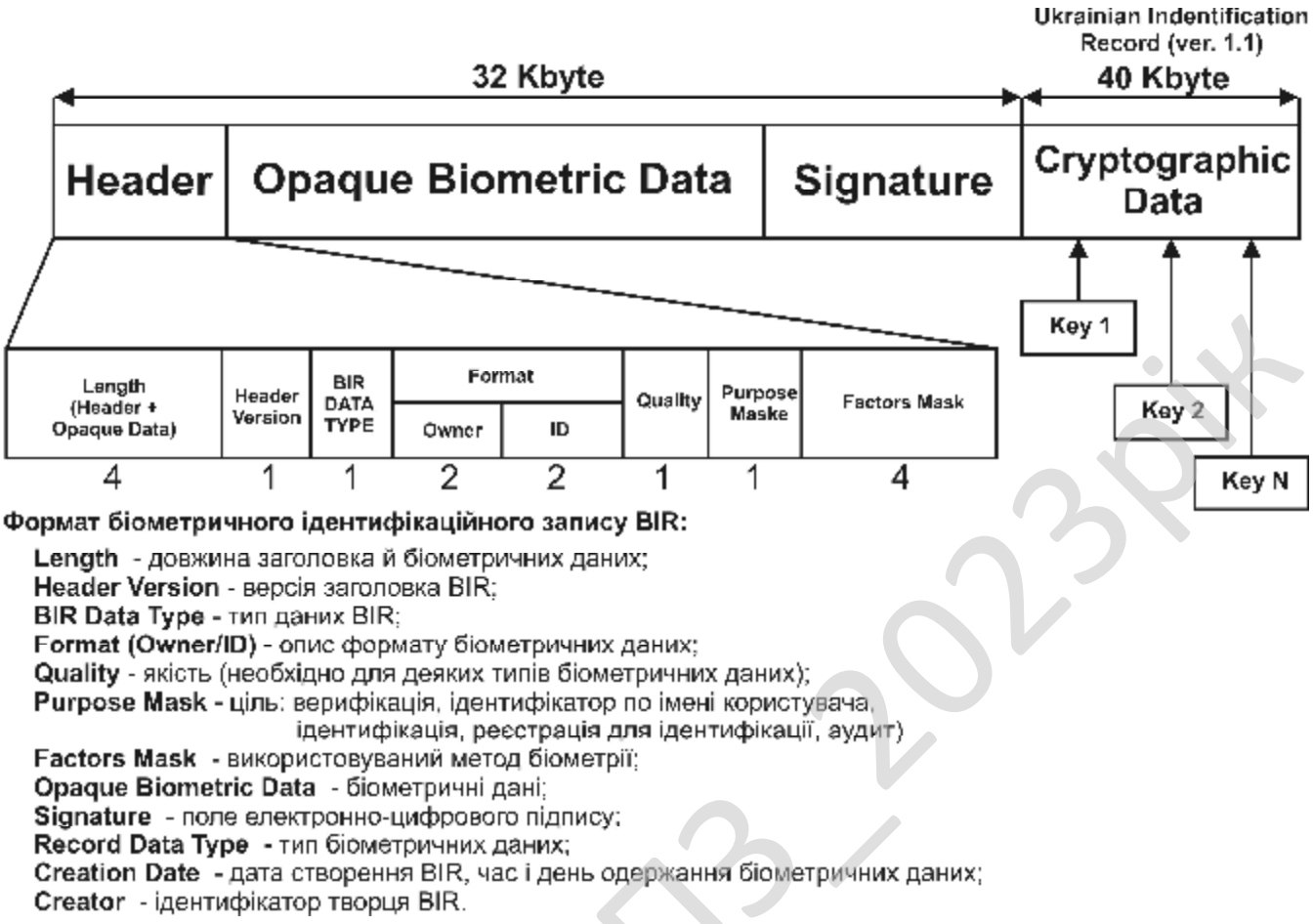

Рисунок 3.14 – Структура формату передачі даних

Основним терміном інтерфейсу BioAPI є біометричний ідентифікаційний запис – BIR (Biometric Identification Record), яку можна визначити як набір біометричних даних, з яким працюють додатки й провайдери послуг. У загальному BIR – це єдиний формат, пропонований для заміни й об'єднання самих різних форматів подання біометричних даних устаткуванням і програмним забезпеченням різних виробників.

Формат BIR, його структура й состав, затверджені Національним інститутом стандартів і технології США NIST, Росії, України й інших країн, а також затверджено Common Biometric Exchange File Format (CBEFF, узагальнений формат обміну біометричними даними). BIR складається із трьох секцій даних: заголовка, біометричних даних і електронного цифрового підпису. Далі розглянемо пояснення до полів BIR по специфікації BioAPI 1.1.

Сектор **Заголовок BIR** (Header) містить у собі наступні поля:

– **Length** – довжина, сума розміру заголовка й біометричних даних;

– **Header Version** – версія заголовка BIR;

– **BIR Data Type** – тип даних BIR. Насправді по стандарті CBEFF формування даного заголовка це поле носить інша назва, більш точно його визначальне: Опції захисту (Security options). У цьому полі вказується тип захисту біометричних даних (1 з 4 значень): тільки біометрія (немає захисту), криптографія, ЕЦП, криптографія й ЕЦП;

– **Format** (**Owner/ID**) – опис формату біометричних даних, представлених після заголовка BIR. Всі формати біометричних даних, підтримуючих BioAPI, реєструються в Міжнародній промисловій біометричній асоціації IBIA (International Biometric Industry Association).

Поле складається із двох підполів:

- Власник формату, тобто компанія, що зареєструвала даний формат подання біометричних даних, формат може бути декількох типів. У Додатку наведений список всіх зареєстрованих форматів подання біометричних даних.

- Ідентифікатор формату – номер використовуваного формату із зареєстрованих власником.

– **Quality** – якість, для деяких типів біометричних даних необхідно вказувати дане значення. Якість вказується в проміжку від 0 до 100. Якщо зазначено «-1» то якість не задана, якщо «-2», те в даному біометричному методі таке поняття як «якість» не використовується.

– **Purpose Mask** – ціль, з якої будуть використані біометричні дані. У даному полі вказується одне із шести значень:

a) верифікація (порівняння «один до одного», користувач називає своє ім'я й пред'являє ідентифікатор, по імені користувача шукається шаблон його ідентифікатора й вони рівняються);

b) ідентифікація (порівняння «один до багатьох», користувач пред'являє свій ідентифікатор, і в базі шаблонів шукається найбільш близький до нього);

c) реєстрація;

d) реєстрація тільки для верифікації;

e) реєстрація тільки для ідентифікації;

f) аудит.

– **Factors Mask** – використовуваний метод біометрії. У цей час зареєстровано 19 методів розпізнавання (можливе розширення даного списку при тверджені наступної версії формату):

– комбінація методів;

– форма особи;

– голос;

– відбиток пальця;

– сітківка ока;

– райдужна оболонка;

– геометрія кисті руки;

– динаміка підпису;

– динаміка клавіатурного набору;

– рух губ;

– термографія особи;

– термографія руки;

– хода;

– запах тіла;

– ДНК;

– форма вуха;

– геометрія пальця;

– геометрія долоні;

– рисунок вен на руці.

Сектор **Біометричні дані** (Opaque Biometric Data) – містить безпосередньо біометричні дані, використовувані для розпізнавання людини, відповідно до зареєстрованого формату, зазначеним у заголовку BIR у поле Format.

ЕЦП (Signature) – поле, у яке заноситься електроний цифровий підпис. Поле є опціональним. Якщо ЕЦП використовується, то підписуються разом і заголовок BIR, і біометричні дані. Крім цього в заголовку BIR існує також декілька опціональних полів, актуальних не для всіх біометричних методів розпізнавання:

– **Record Data Type** – тип біометричних даних, що втримуються. Передані в складі BIR біометричні дані можуть бути трьох типів: неопрацьовані, преоброблені, оброблені;

– **Creation Date** – дата створення BIR, час і день одержання біометричних даних;

– **Creator** – ідентифікатор творця BIR.

В інтерфейсі BioAPI визначені два рівні:

– верхній, визначальний інтерфейс клієнтського й серверного додатків, що викликає функції автентифікації;

– нижній, визначальний інтерфейс взаємодії із провайдером біометричних послуг (Biometric Service Provider), що виконують виклики верхнього рівня.

Верхній рівень (у версії 1.0 має назву «H», high) визначає три основних функції, необхідних додатку для проведення біометричному автентифікації:

1. **Enroll** (реєстрація). Вимір з біометричного пристрою, що зчитує, обробляються в придатну для використання форму, з якої формується шаблон, що повертається додатку.

2. **Verify** (верифікація, порівняння «один до одного»). Одна або більша кількість вимірів знімається з біометричного пристрою, обробляється в придатну для використання форму й потім рівняється з відповідним шаблоном. Результати порівняння вертаються додатку.

3. **Identify** (ідентифікації, порівняння «один до багатьох»). Одна або більша кількість вимірів знімається з біометричного пристрою, обробляється в придатну для використання форму й рівняється з набором шаблонів. Як результат

вертається список, що показує, наскільки близько виміру збігаються з найближчими кандидатами на ідентифікацію з набору шаблонів.

Нижній рівень, SPI (service provider interface) – визначає інтерфейс до провайдеру біометричних послуг (BSP – Biometric Service Provider), у якості якого можуть виступати практично будь-які підтримуючі цей інтерфейс біометричні системи, пристрої або програмні продукти. Функцією SPI є відображення «один до одного» викликів верхнього рівня у виклики до BSP.

3.3 Розробка функціональної схеми

На рисунку 3.15 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

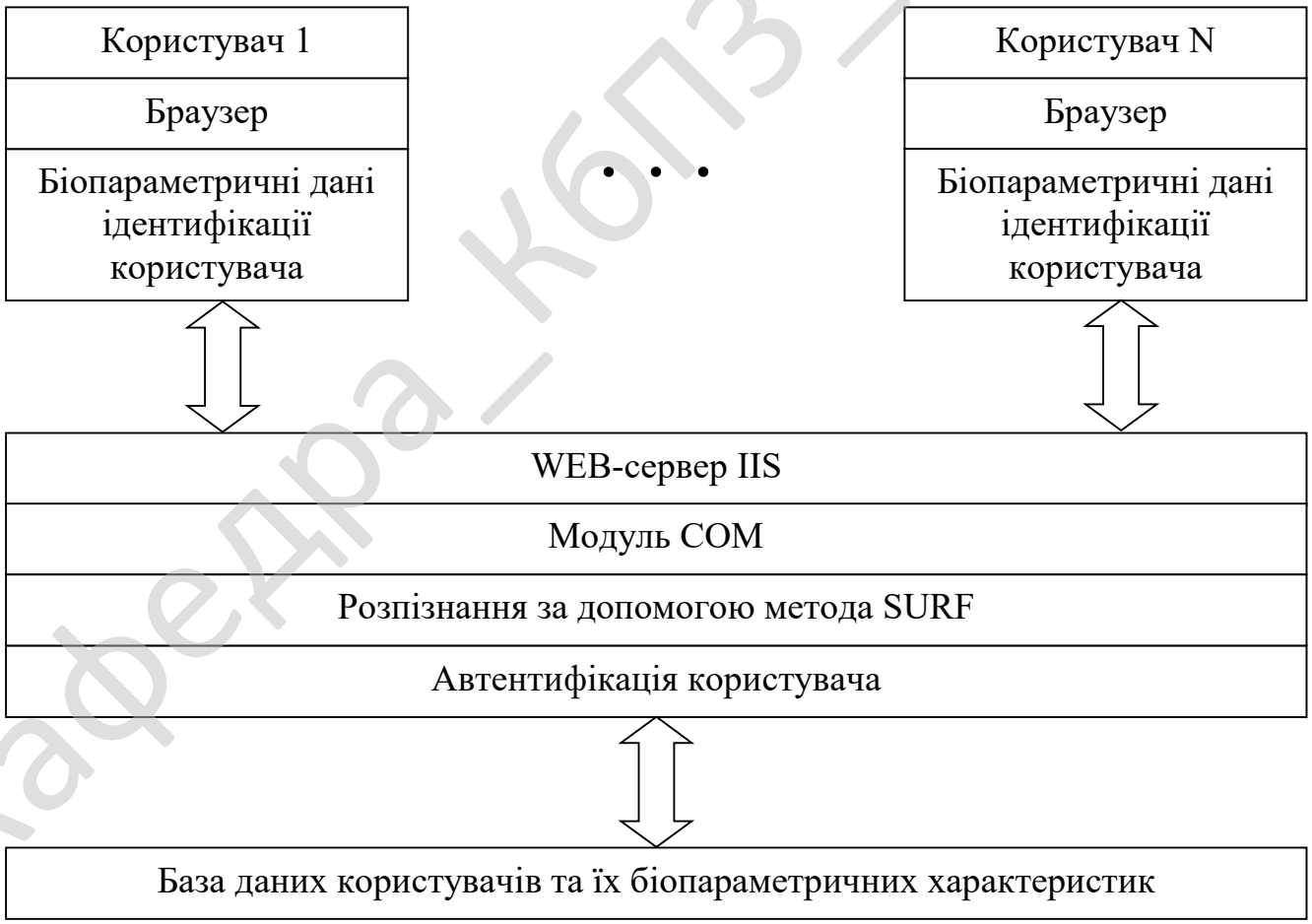

Рисунок 3.15 – Функціональна схема роботи системи

Функціональна схема – це схема, що описує саму суть роботи пристрою, програми, взаємодії й має більш узагальнений рівень, ніж структурна.

У функціональній схемі розглянута загальна взаємодія між клієнтом і сервером (схему варто переглядати зверху донизу).

Як зображено на функціональній схемі, на стороні клієнта формується модель, що складається з біометричного ідентифікаційного запису, криптографічного додавання й відправляється на сервер IIS – це Інформаційний Інтернет-Сервер компанії Microsoft; веб-сервер, який запускається в операційних системах Windows. Там відбувається перевірка криптографічного додавання за допомогою модуля COM. При проходженні перевірки відбувається добування з бази даних Microsoft Access еталонного паспорта й порівняння моделі з паспортом.

На даній схемі можна чітко собі представити загальну функціональну взаємодію й можливості розробленої системи.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.16. З рисунку видно, що процеси взаємодіють наступним чином.

Спершу завантажується процес виведення головного вікна програми. Він взаємодіє з процесом завантаження зображення, який, у свою чергу, взаємодіє з наступними процесами:

– Процес завантаження відбитку пальця.

– Процес завантаження підпису.

– Процес знаходження ключових точок.

Рисунок 3.16 – Діаграма взаємодії процесів

Процес знаходження ключових точок, у свою чергу взаємодіє з наступними процесами:

– Процес обчислення матриці Гессе.

– Процес створення дескрипторів точок.

– Процес малювання кіл, що показують місцезнаходження та масштаб точок.

– Процес малювання ліній, що показують напрям градієнта яскравості.

– Процес пошуку зображення, зі схожим набором ключових точок у базі даних.

Останній процес взаємодіє з процесом виведення знайдених зображень.

Процес виведення знайдених зображень взаємодіє з наступними процесами:

– Процес виведення значення схожості у процентах.

– Процес виведення інформації про зображення.

Процес виведення інформації про зображення взаємодіє з наступними процесами:

– Процес виведення наявних даних про людину.

– Процес виведення прізвища, імені та по-батькові людини, яка розпізнана за відбитком пальця, та підписом.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього користувач обирає завантажити зображення або ні.

Якщо він обирає завантажити зображення, тоді відбувається завантаження зображення з файлу.

Після цього користувач обирає, чи потрібно йому знайти ключові точки.

Якщо потрібно, тоді відбувається виконання наступних кроків:

– Обчислюється матриця Гессе (гессіан).

– Створюються дескриптори точок.

– Відбувається запис дескрипторів в окремий файл, прикріплений до файлу з зображенням.

– Відбувається виведення кіл, що показують місцезнаходження та масштаб точок.

– Виводяться лінії, що показують напрям градієнта яскравості.

Якщо користувач бажає зробити пошук схожого зображення у БД тоді запускається підпрограма пошуку у базі даних схожого зображення.

Якщо схоже зображення знайдено, тоді виконуються наступні дії:

– Виводиться знайдене зображення та значення схожості у процентах.

– Виводиться інформація про зображення.

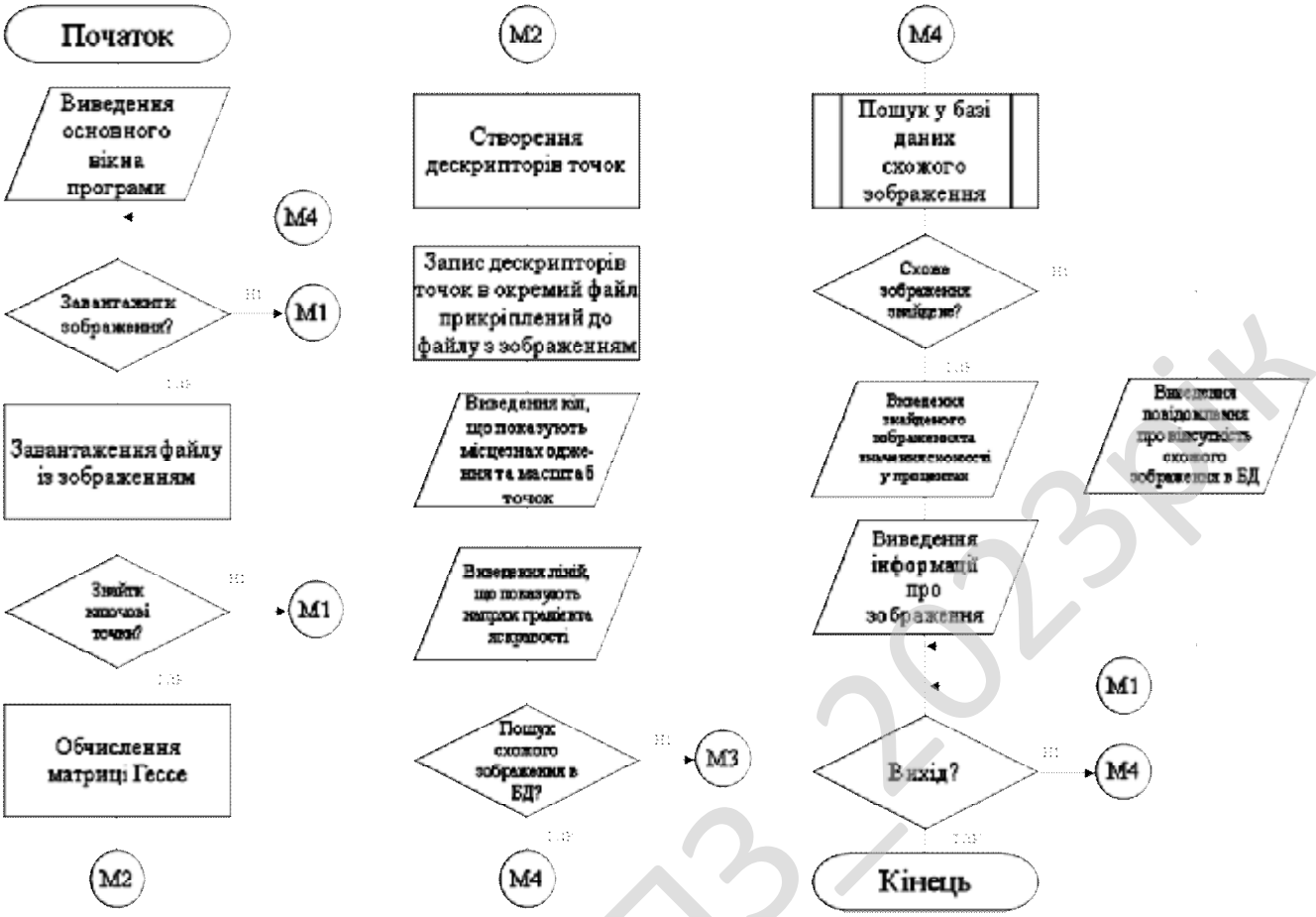

Рисунок 4.1 – Блок-схема основної програми

У противному випадку відбувається виведення повідомлення про відсутність схожого зображення у базі даних.

Після цього користувач визначає, працювати йому далі з програмою, або ні.

Наведемо частину коду, яка реалізує роботу з матрицею Гессе (гесіаном).

```
//! Конструктор без зображення
FastHessian::FastHessian(std::vector<Ipoint> &ipts,
                         const int octaves, const int intervals, const int
                         init_sample,
                         const float thresh)
                         : ipts(ipts), i_width(0), i_height(0)
{
  // Зберігаємо встановлені параметри
  saveParameters(octaves, intervals, init_sample, thresh);
}
```

```
//-------------------------------------------------------
//! Конструктор з зображенням
FastHessian::FastHessian(IplImage *img, std::vector<Ipoint> &ipts,
                         const int octaves, const int intervals,
                         const int init_sample,
                         const float thresh)
                         : ipts(ipts), i_width(0), i_height(0)
{
  // Зберігаємо встановлені параметри
  saveParameters(octaves, intervals, init_sample, thresh);
  // Встановлюємо поточне зображення
  setIntImage(img);
}
//-------------------------------------------------------
FastHessian::~FastHessian()
{
  for (unsigned int i = 0; i < responseMap.size(); ++i)
  {
    delete responseMap[i];
  }
}
//-------------------------------------------------------
//! Зберігаємо параметри
void FastHessian::saveParameters(const int octaves, const int intervals,
                                 const int init_sample, const float thresh)
{
  // Ініціалізуємо змінні з перевіреними граничними значеннями
  this->octaves =
    (octaves > 0 && octaves <= 4 ? octaves : OCTAVES);
  this->intervals =
    (intervals > 0 && intervals <= 4 ? intervals : INTERVALS);
  this->init_sample =
    (init_sample > 0 && init_sample <= 6 ? init_sample : INIT_SAMPLE);
  this->thresh = (thresh >= 0 ? thresh : THRES);
}
//-------------------------------------------------------
//! Встановлюємо або перевстановлюємо джерело цілочисельного зображення
void FastHessian::setIntImage(IplImage *img)
{
  // Змінюємо дежерело зображень
  this->img = img;
  i_height = img->height;
```

```
  i_width = img->width;
}
//-------------------------------------------------------
//! Знайходимо, що зображення змальовує і вписуємо у вектор особливостей
void FastHessian::getIpoints()
{
  // фільтруємо карту індексів
  static const int filter_map [OCTAVES][INTERVALS] = {{0,1,2,3}, {1,3,4,5},
{3,5,6,7}, {5,7,8,9}, {7,9,10,11}};
  // Очищуємо вектор існування ipts
  ipts.clear();
  // Будуємо карту відповідностей
  buildResponseMap();
  // Беремо шар відповідностей
  ResponseLayer *b, *m, *t;
  for (int o = 0; o < octaves; ++o) for (int i = 0; i <= 1; ++i)
  {
    b = responseMap.at(filter_map[o][i]);
    m = responseMap.at(filter_map[o][i+1]);
    t = responseMap.at(filter_map[o][i+2]);
    // цикл над шаром середньої відповідності в щільності самого  розкиданого шару
(завжди вищий), щоб знайти максимум через масштаб і простір
    for (int r = 0; r < t->height; ++r)
    {
      for (int c = 0; c < t->width; ++c)
      {
        if (isExtremum(r, c, t, m, b))
        {
          interpolateExtremum(r, c, t, m, b);
        }
      }
    }
  }
}
//-------------------------------------------------------
//! Будуємо карту відповідностей DoH
void FastHessian::buildResponseMap()
{
  // Розраховуємо відповідності для перших чотирьох октетів:
  // Oct1: 9,  15, 21, 27
  // Oct2: 15, 27, 39, 51
  // Oct3: 27, 51, 75, 99
```

```
// Oct4: 51, 99, 147,195
// Oct5: 99, 195,291,387
// Звільняємо пам’ять і очищуємо усі шари відповідності
for(unsigned int i = 0; i < responseMap.size(); ++i)
  delete responseMap[i];
responseMap.clear();
// Беремо атрибути зображення
int w = (i_width / init_sample);
int h = (i_height / init_sample);
int s = (init_sample);
// Розраховуємо апрксимаційний детермінант значень гессіана
if (octaves >= 1)
{
  responseMap.push_back(new ResponseLayer(w,   h,   s,   9));
  responseMap.push_back(new ResponseLayer(w,   h,   s,  15));
  responseMap.push_back(new ResponseLayer(w,   h,   s,  21));
  responseMap.push_back(new ResponseLayer(w,   h,   s,  27));
}

if (octaves >= 2)
{
  responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 39));
  responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 51));
}
if (octaves >= 3)
{
  responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 75));
  responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 99));
}
if (octaves >= 4)
{
  responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 147));
  responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 195));
}
if (octaves >= 5)
{
  responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 291));
  responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 387));
}
// Отримуємо відповідно зображенню
for (unsigned int i = 0; i < responseMap.size(); ++i)
{
```

```
    buildResponseLayer(responseMap[i]);
  }
}
//-------------------------------------------------------
//! Обчислюємо відповідний DoH для забезпечуваного шару
void FastHessian::buildResponseLayer(ResponseLayer *rl)
{
  float *responses = rl->responses;          // збереження відповідностей
  unsigned char *laplacian = rl->laplacian; // збереження знаку лапласіана
  int step = rl->step;                       // розмір шагу для цього фільтру
  int b = (rl->filter - 1) / 2 + 1;          // границя для цього фільтру
  int l = rl->filter / 3;                    // частка для цього фільтру(розмір
фільтру / 3)
  int w = rl->filter;                        // розмір фільтру
  float inverse_area = 1.f/(w*w);            // чинник нормалізації
  float Dxx, Dyy, Dxy;
  for(int r, c, ar = 0, index = 0; ar < rl->height; ++ar)
  {
    for(int ac = 0; ac < rl->width; ++ac, index++)
    {
      // беремо координати зображення
      r = ar * step;
      c = ac * step;
      // Розраховуємо відповідні компоненти
      Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1, w)
          - BoxIntegral(img, r - l + 1, c - l / 2, 2*l - 1, l)*3;
      Dyy = BoxIntegral(img, r - b, c - l + 1, w, 2*l - 1)
          - BoxIntegral(img, r - l / 2, c - l + 1, l, 2*l - 1)*3;
      Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
            + BoxIntegral(img, r + 1, c - l, l, l)
            - BoxIntegral(img, r - l, c - l, l, l)
            - BoxIntegral(img, r + 1, c + 1, l, l);
      // Нормалізуємо фільтр відповідностей відносно розміру
      Dxx *= inverse_area;
      Dyy *= inverse_area;
      Dxy *= inverse_area;

      // Беремо детермінант відповідності гесціана & знак лапласіана
      responses[index] = (Dxx * Dyy - 0.81f * Dxy * Dxy);
      laplacian[index] = (Dxx + Dyy >= 0 ? 1 : 0);
#ifdef RL_DEBUG
      // створюємо список координат зображень для кожної відповідності
```

```
      rl->coords.push_back(std::make_pair<int,int>(r,c));
#endif
    }
  }
}

//-------------------------------------------------------
//! Не функція Максимального Придушення
int FastHessian::isExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  // визначаємо кордони
  int layerBorder = (t->filter + 1) / (2 * t->step);
  if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder || c >=
t->width - layerBorder)
    return 0;
  // перевіряємо точку-кандидат посередині шара
  float candidate = m->getResponse(r, c, t);
  if (candidate < thresh)
    return 0;
  for (int rr = -1; rr <=1; ++rr)
  {
    for (int cc = -1; cc <=1; ++cc)
    {
      // якщо будь-яка відповідність у 3x3x3 - це не більший максимум кандидата
      if (
        t->getResponse(r+rr, c+cc) >= candidate ||
        ((rr != 0 && cc != 0) && m->getResponse(r+rr, c+cc, t) >= candidate) ||
        b->getResponse(r+rr, c+cc, t) >= candidate
        )
        return 0;
    }
  }
  return 1;
}
//-------------------------------------------------------
//! Інтерполюємо простір масштабу екстремума до точності підпікселя, щоб
сформувати особливість зображення.
void FastHessian::interpolateExtremum(int r, int c, ResponseLayer *t,
ResponseLayer *m, ResponseLayer *b)
{
  // беремо шаг дистанції між фільтрами
```

```
  // перевіряємо проміжний фільтр який є середнім між вищим та нижчим
  int filterStep = (m->filter - b->filter);
  assert(filterStep > 0 && t->filter - m->filter == m->filter - b->filter);

  // Беремо відгалуження до фактичного розташування екстремуму
  double xi = 0, xr = 0, xc = 0;
  interpolateStep(r, c, t, m, b, &xi, &xr, &xc );
  // Якщо точка досить близька до фактичного екстремуму
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f )
  {
    Ipoint ipt;
    ipt.x = static_cast<float>((c + xc) * t->step);
    ipt.y = static_cast<float>((r + xr) * t->step);
    ipt.scale = static_cast<float>((0.1333f) * (m->filter + xi * filterStep));
    ipt.laplacian = static_cast<int>(m->getLaplacian(r,c,t));
    ipts.push_back(ipt);
  }
}
//-------------------------------------------------------
//! Виконуємо один крок інтерполяції екстремуму.
void FastHessian::interpolateStep(int r, int c, ResponseLayer *t,
                                  ResponseLayer *m, ResponseLayer *b,
                                  double* xi, double* xr, double* xc )
{
  CvMat* dD, * H, * H_inv, X;
  double x[3] = { 0 };
  dD = deriv3D( r, c, t, m, b );
  H = hessian3D( r, c, t, m, b );
  H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  cvInvert( H, H_inv, CV_SVD );
  cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
  cvGEMM( H_inv, dD, -1, NULL, 0, &X, 0 );
  cvReleaseMat( &dD );
  cvReleaseMat( &H );
  cvReleaseMat( &H_inv );
  *xi = x[2];
  *xr = x[1];
  *xc = x[0];
}
//-------------------------------------------------------
//! Обчислюємо часткові похідні слова в x, y, і масштаб пікселя.
```

```
CvMat* FastHessian::deriv3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* dI;
  double dx, dy, ds;
  dx = (m->getResponse(r, c + 1, t) - m->getResponse(r, c - 1, t)) / 2.0;
  dy = (m->getResponse(r + 1, c, t) - m->getResponse(r - 1, c, t)) / 2.0;
  ds = (t->getResponse(r, c) - b->getResponse(r, c, t)) / 2.0;

  dI = cvCreateMat( 3, 1, CV_64FC1 );
  cvmSet( dI, 0, 0, dx );
  cvmSet( dI, 1, 0, dy );
  cvmSet( dI, 2, 0, ds );
  return dI;
}
//-------------------------------------------------------
//! Обчислюємо 3D матрицю гессіана для пікселя.
CvMat* FastHessian::hessian3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* H;
  double v, dxx, dyy, dss, dxy, dxs, dys;
  v = m->getResponse(r, c, t);
  dxx = m->getResponse(r, c + 1, t) + m->getResponse(r, c - 1, t) - 2 * v;
  dyy = m->getResponse(r + 1, c, t) + m->getResponse(r - 1, c, t) - 2 * v;
  dss = t->getResponse(r, c) + b->getResponse(r, c, t) - 2 * v;
  dxy = ( m->getResponse(r + 1, c + 1, t) - m->getResponse(r + 1, c - 1, t) -
          m->getResponse(r - 1, c + 1, t) + m->getResponse(r - 1, c - 1, t) ) /
4.0;
  dxs = ( t->getResponse(r, c + 1) - t->getResponse(r, c - 1) -
          b->getResponse(r, c + 1, t) + b->getResponse(r, c - 1, t) ) / 4.0;
  dys = ( t->getResponse(r + 1, c) - t->getResponse(r - 1, c) -
          b->getResponse(r + 1, c, t) + b->getResponse(r - 1, c, t) ) / 4.0;
  H = cvCreateMat( 3, 3, CV_64FC1 );
  cvmSet( H, 0, 0, dxx );
  cvmSet( H, 0, 1, dxy );
  cvmSet( H, 0, 2, dxs );
  cvmSet( H, 1, 0, dxy );
  cvmSet( H, 1, 1, dyy );
  cvmSet( H, 1, 2, dys );
  cvmSet( H, 2, 0, dxs );
  cvmSet( H, 2, 1, dys );
```

```
  cvmSet( H, 2, 2, dss );
  return H;
}
```

На рисунку 4.2 зображено блок-схему алгоритму роботи підпрограми пошуку. Вона працює наступним чином.

Спершу визначається, за яким критерієм буде відбуватися ідентифікація користувача, тобто за відбитком пальця, або за підписом.

Якщо користувач обирає, за відбитком пальця, тоді відбувається виконання наступних дій.

Спершу відбувається читання дескрипторів ключових точок зображення шаблону.

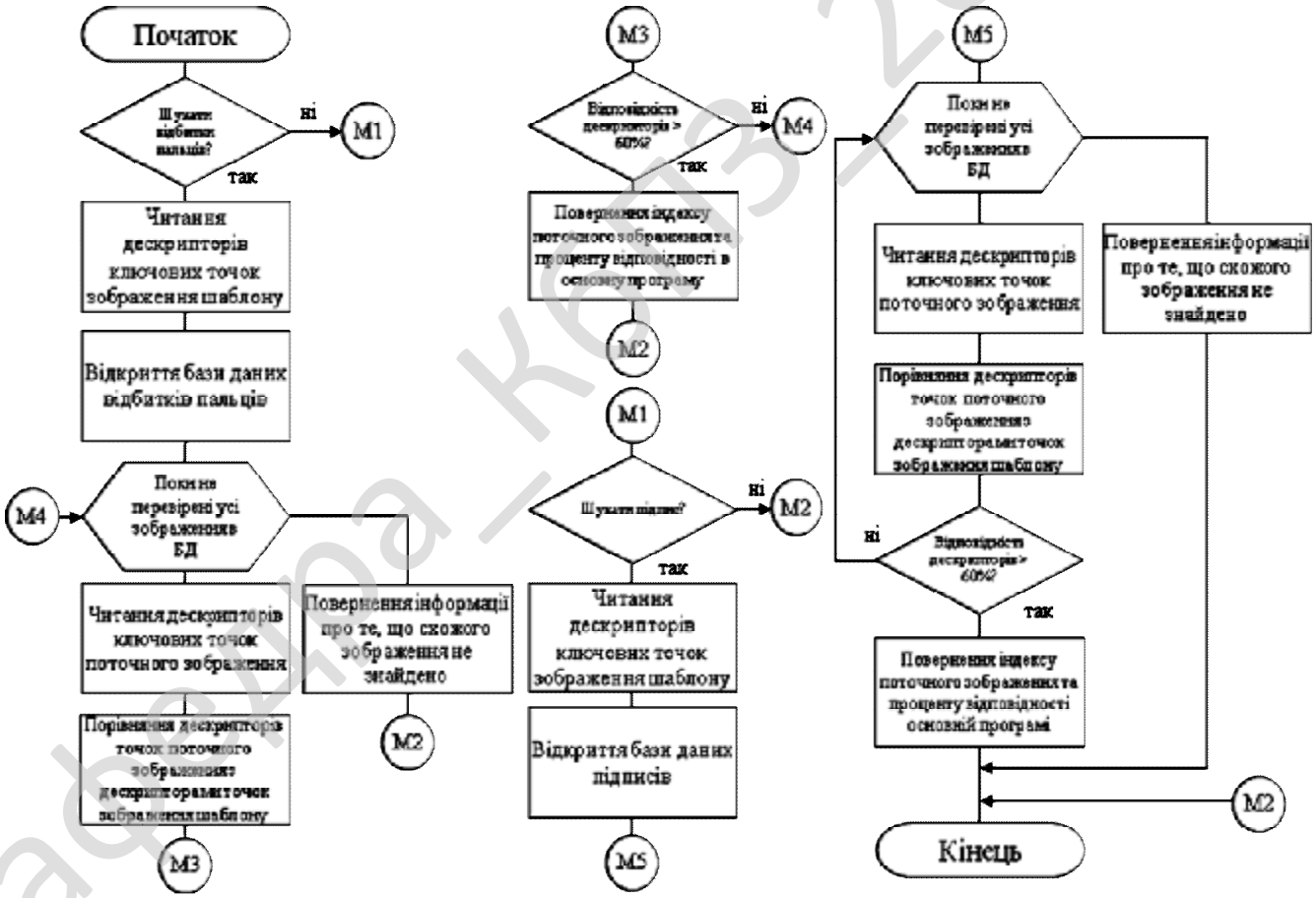

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми пошуку

Після цього відбувається відкриття бази даних відбитків пальців.

Далі, поки не перевірені усі зображення у базі даних виконується цикл, у якому відбуваються наступні дії:

– Читання дескрипторів ключових точок поточного зображення.

– Порівняння дескрипторів точок поточного зображення з дескрипторами точок зображення шаблону.

– Якщо відповідність дескрипторів більше 60%, тоді відбувається повернення індексу поточного зображення та проценту відповідності в основну програму, й підпрограма закінчує свою роботу. У противному випадку, цикл продовжує свою роботу.

Після закінчення роботи циклу, якщо не знайдено співпадань, виводиться інформація про те, що схожого зображення не знайдено, й підпрограмма закінчує свою роботу.

Якщо користувач обирає, за підписом, тоді відбувається виконання наступних дій.

Спершу відбувається читання дескрипторів ключових точок зображення шаблону.

Після цього відбувається відкриття бази даних підписів користувачів системи.

Далі, поки не перевірені усі зображення у базі даних виконується цикл, у якому відбуваються наступні дії:

– Читання дескрипторів ключових точок поточного зображення.

– Порівняння дескрипторів точок поточного зображення з дескрипторами точок зображення шаблону.

– Якщо відповідність дескрипторів більше 60%, тоді відбувається повернення індексу поточного зображення та проценту відповідності в основну програму, й підпрограма закінчує свою роботу. У противному випадку, цикл продовжує свою роботу.

Після закінчення роботи циклу, якщо не знайдено співпадань, виводиться інформація про те, що схожого зображення не знайдено, й підпрограма закінчує свою роботу.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм SEED – у криптографії симетричний блоковий криптоалгоритм на основі Мережі Фейстеля, розроблений Корейським агентством інформаційної безпеки (Korean Information Security Agency, KISA) в 1998 році. В алгоритмі використовується 128-бітний блок і ключ довжиною 128 біт. Алгоритм одержав широке поширення й використовується фінансовими й банківськими структурами, виробничими підприємствами й бюджетними установами Південної Кореї, оскільки 40-бітний SSL не забезпечує на даний момент мінімально необхідного рівня безпеки. Агентством по захисту інформації специфіковане використання шифру SEED у протоколах TLS і S/MIME. У той же час, алгоритм SEED не реалізований у більшості сучасних браузерів і інтернет-додатків, що утрудняє його використання в даній сфері поза межами Південної Кореї.

SEED являє собою Мережа Фейстеля з 16 раундами, 128-бітовими блоками й 128-бітовим ключем. Алгоритм використовує дві 8×8 таблиці підстановки, які, як такі з Safer, виведені з дискретного зведення в ступінь (у цьому випадку, x^{247} і x^{251} – плюс деякі «несумісні операції»). Це є деякою подібністю с MISTY1 у рекурсивності його структури: 128-бітовий повний шифр – мережа Фейстеля з F-функцією, що впливає на 64-бітові половини, у той час як сама F-функція – Мережа Фейстеля, складена з G-функції, що впливає на 32-розрядні половини. Однак рекурсія не простягнеться далі, тому що G-функція – не Мережа Фейстеля. В G-функції 32-розрядне слово розглядають як чотири 8-бітових байта, кожний з яких проходить через одну або іншу таблицю підстановки, потім поєднується в помірковано комплексному наборі булевих функцій таким чином, що кожний біт виводу залежить від 3 з 4 вхідних байтів.

SEED має складний ключовий розклад, генеруючи тридцять два 32-розрядних додаткових символу, використовуючи G-функції на серіях обертань вихідного неопрацьованого ключа, комбінованого зі спеціальними раундовими константами (як в TEA) від «Золотого співвідношення» (англ. Golden ratio).

Згідно з дослідженнями KISA, алгоритм SEED «надійно протистоїть відомим атакам».

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Система, досить проста в експлуатації, тому наведемо ряд скріншотів, для того, щоб зрозуміти, як працює розроблений, у ході виконання бакалаврського проектування програмний продукт.

На рисунку 5.1 зображено головне вікно програми (обчислення та візуалізація дескрипторів ключових точок для зображення відбитку пальця).

Рисунок 5.1 – Головне вікно програми (обчислення та візуалізація дескрипторів ключових точок для зображення відбитку пальця)

З рисунку видно, що інтерфейс користувача складається з використання наступних кнопок:

– Відкрити файл.

– Обчислити ключові точки.

– Знайти в базі даних.

– Про програму.

Також виводиться вікно, у якому наведено зображення, яке потрібно розпізнати, та знайти подібне у відповідній базі даних.

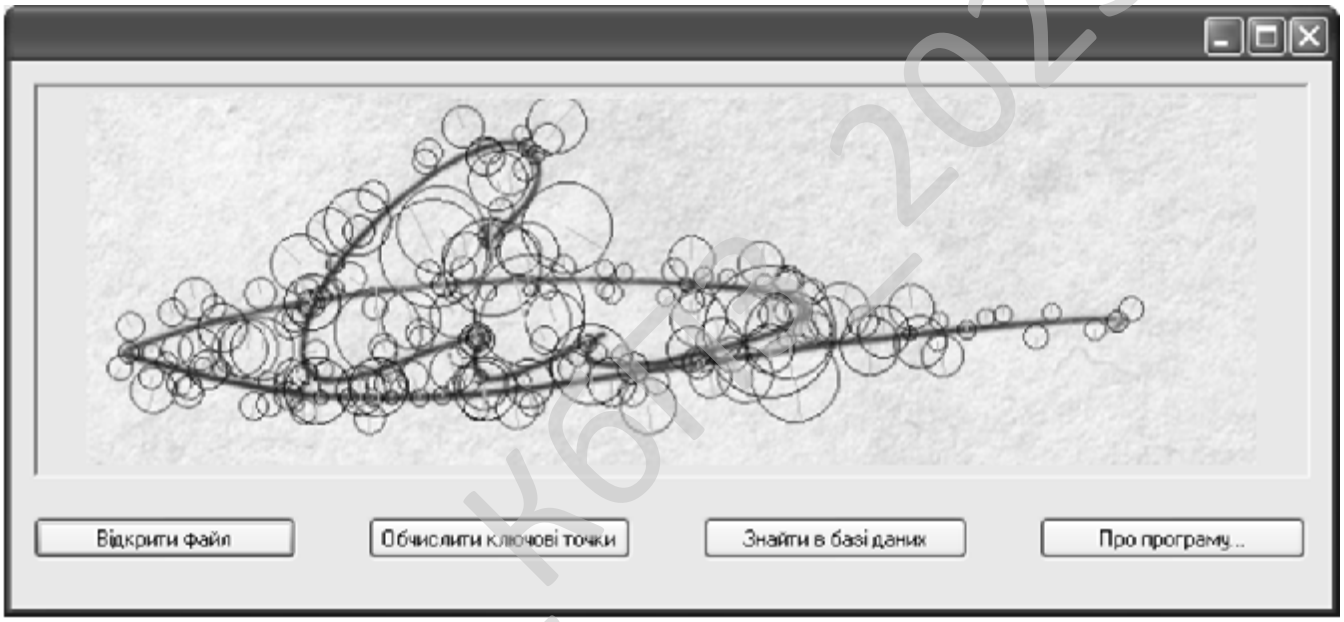

Рисунок 5.2 – Головне вікно програми (обчислення та візуалізація дескрипторів ключових точок для зображення підпису)

На рисунку 5.1 наведено обчислення та візуалізація дескрипторів ключових точок для зображення відбитку пальця.

На рисунку 5.2 наведено обчислення та візуалізація дескрипторів ключових точок для зображення підпису користувача.

На рисунку 5.3 наведено обчислення та візуалізація дескрипторів ключових точок для абстрактного зображення.

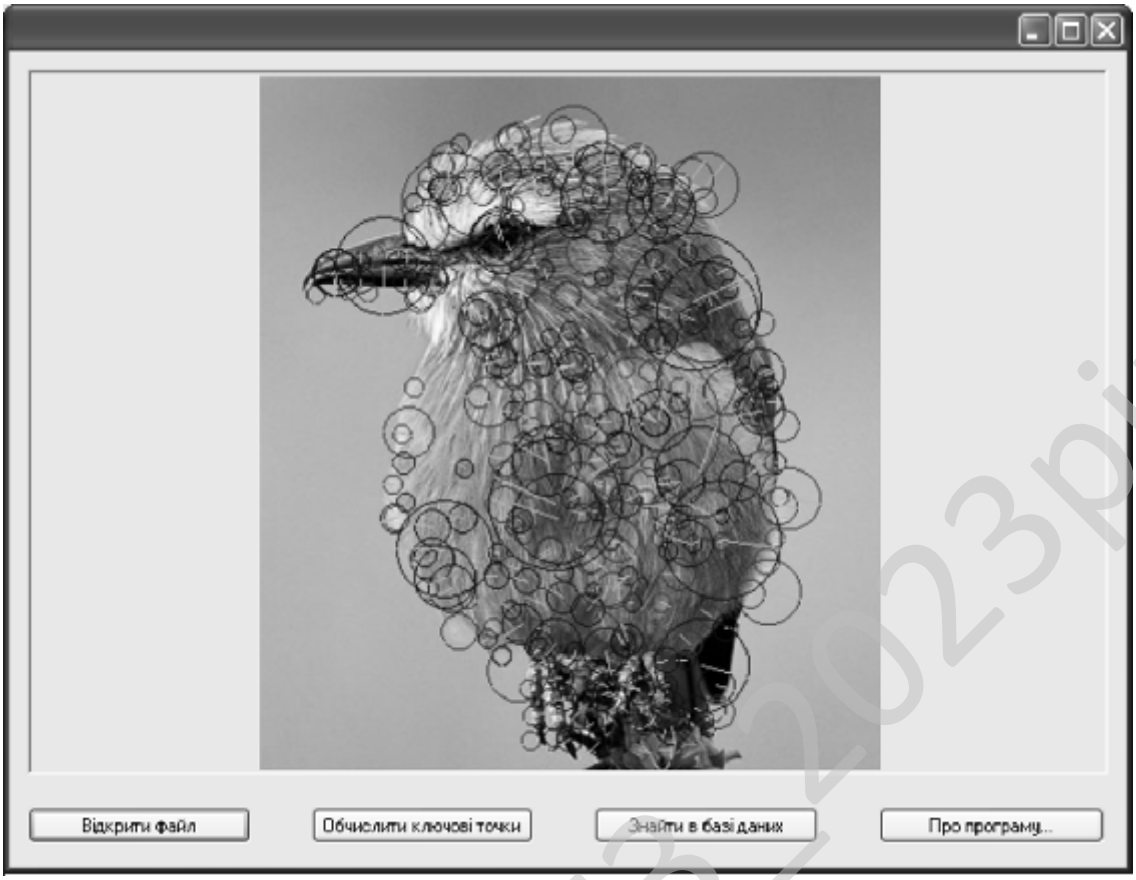

Рисунок 5.3 – Головне вікно програми (обчислення та візуалізація дескрипторів ключових точок для абстрактного зображення)

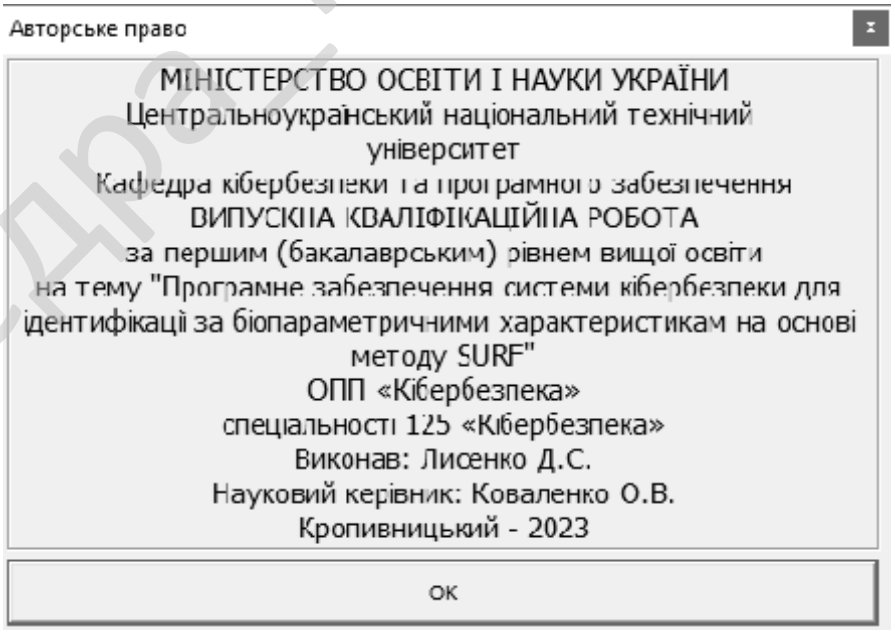

Рисунок 5.4 – Довідка

На рисунку 5.4 зображено вікно довідки, з якого стають зрозумілими наступні дані:

– Виконавець бакалаврського проекту.

– Керівник бакалаврського проекту.

– Тема бакалаврського проекту.

– Місце виконання бакалаврського проекту.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для ідентифікації за біопараметричними характеристикам на основі методу SURF.

– Досліджена система для ідентифікації за біопараметричними характеристикам на основі методу SURF.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для ідентифікації за біопараметричними характеристикам на основі методу SURF.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи кібербезпеки Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм SEED.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ANSI B10.8. Finger Minutiae Extraction and Format Standard for One-to-One Matching
2. NISTIR 6529, ISO/IEC 19785. Common Biometric Exchange File Format (CBEFF)
3. ANSI X9.84. Biometric Information Management and Security
4. ISO/IEC 7816-11. Personal Verification Through Biometric Methods
5. ISO/IEC 19784. BioAPI 2.0
6. ANSI/INCITS 358. BioAPI 1.1
7. ISO/IEC 19794-2. Finger Minutiae Data
8. ISO/IEC 19794-4. Finger Image Data
9. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». CEUR Workshop Proceedings Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe (Scopus).
10. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». CEUR Workshop Proceedings Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe (Scopus).
11. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

12. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

13. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

14. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

15. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

16. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

17. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

18. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

19. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

20. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". – Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

21. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

22. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

23. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

24. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

25. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

26. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

27. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

28. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

29. Коваленко О.В. Управління ризиками розроблення програмного забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

30. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

31. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

32. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

33. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

34. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

35. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

36. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

37. Коваленко О.В. Математичні моделі технології тестування DOM XSS вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

38. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін'єкцій / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

39. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

40. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

41. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea

informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

42. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

43. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

44. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

45. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

46. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

47. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

48. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

49. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

50. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

51. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

52. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов //

Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

53. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції "Проблеми кібербезпеки інформаційно-телекомунікаційних систем". м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

54. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

55. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

56. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

57. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-

практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

58. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

59. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

60. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

61. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

62. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

63. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

64. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

65. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

66. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

67. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 13-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 86 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Коваленко О.В.

Програмне забезпечення системи кібербезпеки для ідентифікації за біопараметричними характеристикам на основі методу SURF

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2023 року

main.cpp – головний файл програми

```
#include "surflib.h"
#include "kmeans.h"
#include <ctime>
#include <iostream>

//-------------------------------------------------------
/* У програмі, для розпізнання біопараметричних характеристик, використовується
метод SURF.  Для цього необхідно лише 1 звернення до функції, щоб запустити
описані особливості SURF!
// Визначемо параметри ПРОЦЕДУРИ, як:
//  - 1, вказати шлях до статичного зображення
//  - 2, захопити відео та зображення від вебкамери
//  - 3, визначити знаходження об'єкту в зображення (працює при динамічному
виконанні програми)
//  - 4, показати переміщення особи (працює при динамічному виконанні програми)
//  - 5, показати зміни між статичними зображеннями
*/


#define PROCEDURE 2

//-------------------------------------------------------

int mainImage(void)
{
  // Оголошення Ipoints та інших змінних
  IpVec ipts;
  IplImage *img=cvLoadImage("imgs/sf.jpg");

  // Визначення та описання потрібних ключових точок у зображенні
  clock_t start = clock();
  surfDetDes(img, ipts, false, 5, 4, 2, 0.0004f);
  clock_t end = clock();

  std::cout<< "Програма розпізнання біопараметричних характеристик методом SURF
знайшла: " << ipts.size() << " особливі точки" << std::endl;
  std::cout<< "Програма розпізнання біопараметричних характеристик методом SURF
виконується: " << float(end - start) / CLOCKS_PER_SEC  << " секунд" <<
std::endl;

  // Відображення знайдених особливих точок
  drawIpoints(img, ipts);

  // Виведення результату на екран
  showImage(img);

  return 0;
}

//-------------------------------------------------------

int mainVideo(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Ініціалізація пристрою відеозапису
  //cv::VideoWriter vw("c:\\out.avi",
CV_FOURCC('D','I','V','X'),10,cvSize(320,240),1);
  //vw << img;

  // Створюємо вікно
```

```
  cvNamedWindow("Програма розпізнання біопараметричних характеристик методом
SURF", CV_WINDOW_AUTOSIZE );

  // Оголошення Ipoints та інших змінних
  IpVec ipts;
  IplImage *img=NULL;

  // Прокручуємо головну картинку
  while( 1 )
  {
    // Вирізаємо фрейм з джерела картинки
    img = cvQueryFrame(capture);

    // Отримуємо точки для методу SURF
    surfDetDes(img, ipts, false, 4, 4, 2, 0.004f);

    // Відображення знайдених особливих точок
    drawIpoints(img, ipts);

    // Виводимо фігуру FPS
    drawFPS(img);

    // Виведення результату на екран
    cvShowImage("Програма розпізнання біопараметричних характеристик методом
SURF", img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання біопараметричних характеристик методом
SURF" );
  return 0;
}


//-------------------------------------------------------


int mainMatch(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Оголошення Ipoints та інших змінних
  IpPairVec matches;
  IpVec ipts, ref_ipts;

  // Описуємо пошук необхідних  точок на  відеокадрі
  // Це описано у рядку IplImage *img = cvLoadImage("imgs/object.jpg");
  // де object.jpg потрібний нам кадр з відео
  IplImage *img = cvLoadImage("imgs/object.jpg");
  if (img == NULL) error("Потрібно завантажити довідкове зображення для того,
щоб управляти відповідністю процедури");
  CvPoint src_corners[4] = {{0,0}, {img->width,0}, {img->width, img->height},
{0, img->height}};
  CvPoint dst_corners[4];

  // Витягуємо довідковий об’єкт Ipoints
  surfDetDes(img, ref_ipts, false, 3, 4, 3, 0.004f);
  drawIpoints(img, ref_ipts);
  showImage(img);

  // Створюємо вікно
  cvNamedWindow("Програма розпізнання біопараметричних характеристик методом
SURF", CV_WINDOW_AUTOSIZE );
```

```
  // Прокручуємо головну картинку
  while( true )
  {
    // Вирізаємо фрейм з джерела картинки
    img = cvQueryFrame(capture);

    // Визначаємо та описуємо особливі точки у фреймі
    surfDetDes(img, ipts, false, 3, 4, 3, 0.004f);

    // Рисуємо відповідний вектор
    getMatches(ipts,ref_ipts,matches);

    // Цей виклик знаходить, де об'єктні кути мають бути у фреймі
    if (translateCorners(matches, src_corners, dst_corners))
    {
      // Рисуємо фігуру вокруг об'єкту
      for(int i = 0; i < 4; i++ )
      {
        CvPoint r1 = dst_corners[i%4];
        CvPoint r2 = dst_corners[(i+1)%4];
        cvLine( img, cvPoint(r1.x, r1.y),
          cvPoint(r2.x, r2.y), cvScalar(255,255,255), 3 );
      }

      for (unsigned int i = 0; i < matches.size(); ++i)
        drawIpoint(img, matches[i].first);
    }

    // Виводимо фігуру FPS
    drawFPS(img);

    // Виведення результату на екран
    cvShowImage("Програма розпізнання біопараметричних характеристик методом
SURF", img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання біопараметричних характеристик методом
SURF" );
  return 0;
}


//--------------------------------------------------


int mainMotionPoints(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("No Capture");

  // Створюємо вікно
  cvNamedWindow("Програма розпізнання біопараметричних характеристик методом
SURF", CV_WINDOW_AUTOSIZE );

  // Оголошення Ipoints та інших змінних
  IpVec ipts, old_ipts, motion;
  IpPairVec matches;
  IplImage *img;

  // Прокручуємо головну картинку
  while( 1 )
  {
    // Вирізаємо фрейм з джерела картинки
```

```
    img = cvQueryFrame(capture);

    // Визначення та описання потрібних ключових точок у зображенні
    old_ipts = ipts;
    surfDetDes(img, ipts, true, 3, 4, 2, 0.0004f);

    // Рисуємо відповідний вектор
    getMatches(ipts,old_ipts,matches);
    for (unsigned int i = 0; i < matches.size(); ++i)
    {
      const float & dx = matches[i].first.dx;
      const float & dy = matches[i].first.dy;
      float speed = sqrt(dx*dx+dy*dy);
      if (speed > 5 && speed < 30)
        drawIpoint(img, matches[i].first, 3);
    }

    // Виведення результату на екран
    cvShowImage("Програма розпізнання біопараметричних характеристик методом
SURF", img);

    // Якщо нажата клавіша ESC перериваємо прокручування
    if( (cvWaitKey(10) & 255) == 27 ) break;
  }

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання біопараметричних характеристик методом
SURF" );
  return 0;
}


//--------------------------------------------------------

int mainStaticMatch()
{
  IplImage *img1, *img2;
  img1 = cvLoadImage("imgs/img1.jpg");
  img2 = cvLoadImage("imgs/img2.jpg");

  IpVec ipts1, ipts2;
  surfDetDes(img1,ipts1,false,4,4,2,0.0001f);
  surfDetDes(img2,ipts2,false,4,4,2,0.0001f);

  IpPairVec matches;
  getMatches(ipts1,ipts2,matches);

  for (unsigned int i = 0; i < matches.size(); ++i)
  {
    drawPoint(img1,matches[i].first);
    drawPoint(img2,matches[i].second);

    const int & w = img1->width;

cvLine(img1,cvPoint(matches[i].first.x,matches[i].first.y),cvPoint(matches[i].se
cond.x+w,matches[i].second.y), cvScalar(255,255,255),1);
    cvLine(img2,cvPoint(matches[i].first.x-
w,matches[i].first.y),cvPoint(matches[i].second.x,matches[i].second.y),
cvScalar(255,255,255),1);
  }

  std::cout<< "Відповідає: " << matches.size();

  cvNamedWindow("1", CV_WINDOW_AUTOSIZE );
  cvNamedWindow("2", CV_WINDOW_AUTOSIZE );
  cvShowImage("1", img1);
  cvShowImage("2",img2);
  cvWaitKey(0);
```

```
  return 0;
}

//-------------------------------------------------------

int mainKmeans(void)
{
  IplImage *img = cvLoadImage("imgs/img1.jpg");
  IpVec ipts;
  Kmeans km;

  // Беремо Ipoints
  surfDetDes(img,ipts,true,3,4,2,0.0006f);

  for (int repeat = 0; repeat < 10; ++repeat)
  {

    IplImage *img = cvLoadImage("imgs/img1.jpg");
    km.Run(&ipts, 5, true);
    drawPoints(img, km.clusters);

    for (unsigned int i = 0; i < ipts.size(); ++i)
    {
      cvLine(img, cvPoint(ipts[i].x,ipts[i].y),
cvPoint(km.clusters[ipts[i].clusterIndex].x
,km.clusters[ipts[i].clusterIndex].y),cvScalar(255,255,255));
    }

    showImage(img);
  }

  return 0;
}

//-------------------------------------------------------

int main(void)
{
  if (PROCEDURE == 1) return mainImage();
  if (PROCEDURE == 2) return mainVideo();
  if (PROCEDURE == 3) return mainMatch();
  if (PROCEDURE == 4) return mainMotionPoints();
  if (PROCEDURE == 5) return mainStaticMatch();
  if (PROCEDURE == 6) return mainKmeans();
}
```

surf.cpp – реалізація алгоритму SURF

```
/*************************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*************************************************************/

#include "utils.h"

#include "surf.h"

//-----------------------------------------------------------
//! SURF константи (їх не потрібно вводити під час виконання програми)
const float pi = 3.14159f;

const double gauss25 [7][7] = {

0.02350693969273,0.01849121369071,0.01239503121241,0.00708015417522,0.0034462810
1733,0.00142945847484,0.00050524879060,

0.02169964028389,0.01706954162243,0.01144205592615,0.00653580605408,0.0031813183
4134,0.00131955648461,0.00046640341759,

0.01706954162243,0.01342737701584,0.00900063997939,0.00514124713667,0.0025025136
4222,0.00103799989504,0.00036688592278,

0.01144205592615,0.00900063997939,0.00603330940534,0.00344628101733,0.0016774850
5986,0.00069579213743,0.00024593098864,

0.00653580605408,0.00514124713667,0.00344628101733,0.00196854695367,0.0009581946
7066,0.00039744277546,0.00014047800980,

0.00318131834134,0.00250251364222,0.00167748505986,0.00095819467066,0.0004664034
1759,0.00019345616757,0.00006837798818,

0.00131955648461,0.00103799989504,0.00069579213743,0.00039744277546,0.0001934561
6757,0.00008024231247,0.00002836202103
};

const double gauss33 [11][11] = {

0.014614763,0.013958917,0.012162744,0.00966788,0.00701053,0.004637568,0.00279865
7,0.001540738,0.000773799,0.000354525,0.000148179,

0.013958917,0.013332502,0.011616933,0.009234028,0.006695928,0.004429455,0.002673
066,0.001471597,0.000739074,0.000338616,0.000141529,

0.012162744,0.011616933,0.010122116,0.008045833,0.005834325,0.003859491,0.002329
107,0.001282238,0.000643973,0.000295044,0.000123318,

0.00966788,0.009234028,0.008045833,0.006395444,0.004637568,0.003067819,0.0018513
53,0.001019221,0.000511879,0.000234524,9.80224E-05,

0.00701053,0.006695928,0.005834325,0.004637568,0.003362869,0.002224587,0.0013424
83,0.000739074,0.000371182,0.000170062,7.10796E-05,

0.004637568,0.004429455,0.003859491,0.003067819,0.002224587,0.001471597,0.000888
072,0.000488908,0.000245542,0.000112498,4.70202E-05,

0.002798657,0.002673066,0.002329107,0.001851353,0.001342483,0.000888072,0.000535
929,0.000295044,0.000148179,6.78899E-05,2.83755E-05,

0.001540738,0.001471597,0.001282238,0.001019221,0.000739074,0.000488908,0.000295
044,0.00016243,8.15765E-05,3.73753E-05,1.56215E-05,

0.000773799,0.000739074,0.000643973,0.000511879,0.000371182,0.000245542,0.000148
179,8.15765E-05,4.09698E-05,1.87708E-05,7.84553E-06,
```

```
0.000354525,0.000338616,0.000295044,0.000234524,0.000170062,0.000112498,6.78899E
-05,3.73753E-05,1.87708E-05,8.60008E-06,3.59452E-06,
  0.000148179,0.000141529,0.000123318,9.80224E-05,7.10796E-05,4.70202E-
05,2.83755E-05,1.56215E-05,7.84553E-06,3.59452E-06,1.50238E-06
};

//---------------------------------------------------------

//---------------------------------------------------------

//! Конструктор
Surf::Surf(IplImage *img, IpVec &ipts)
: ipts(ipts)
{
  this->img = img;
}

//---------------------------------------------------------

//! Опишемо усі особливості у векторі, що поставляється
void Surf::getDescriptors(bool upright)
{
  // Перевіряємо Ipoints на опис
  if (!ipts.size()) return;

  // Беремо розмір вектору для фіксованих меж циклу
  int ipts_size = (int)ipts.size();

  if (upright)
  {
    // U-SURF цикл тільки отримує дескриптори
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Витягуємо  вертикальні (тобто інваріант не обертання) дескриптори
      getDescriptor(true);
    }
  }
  else
  {
    // Головний SURF-64 цикл визначення орієнтації та отримання дескрипторів
    for (int i = 0; i < ipts_size; ++i)
    {
      // Встановлюємо Ipoint на опис
      index = i;

      // Призначаємо орієнтацію і витягуємо дескриптори інваріанту обертання
      getOrientation();
      getDescriptor(false);
    }
  }
}

//---------------------------------------------------------

//! Призначаємо поставленняIpoint на орієнтацію
void Surf::getOrientation()
{
  Ipoint *ipt = &ipts[index];
  float gauss = 0.f, scale = ipt->scale;
  const int s = fRound(scale), r = fRound(ipt->y), c = fRound(ipt->x);
  std::vector<float> resX(109), resY(109), Ang(109);
  const int id[] = {6,5,4,3,2,1,0,1,2,3,4,5,6};

  int idx = 0;
  // розраховуємо відповідні для точок Хаара в межах радіусу 6*масштаб
```

```
  for(int i = -6; i <= 6; ++i)
  {
    for(int j = -6; j <= 6; ++j)
    {
      if(i*i + j*j < 36)
      {
        gauss = static_cast<float>(gauss25[id[i+6]][id[j+6]]);
        resX[idx] = gauss * haarX(r+j*s, c+i*s, 4*s);
        resY[idx] = gauss * haarY(r+j*s, c+i*s, 4*s);
        Ang[idx] = getAngle(resX[idx], resY[idx]);
        ++idx;
      }
    }
  }

  // розраховуємо основний напрямок
  float sumX=0.f, sumY=0.f;
  float max=0.f, orientation = 0.f;
  float ang1=0.f, ang2=0.f;

  // цикл слайдів pi/3 вікно біля точки, яка може бути
  for(ang1 = 0; ang1 < 2*pi;  ang1+=0.15f) {
    ang2 = ( ang1+pi/3.0f > 2*pi ? ang1-5.0f*pi/3.0f : ang1+pi/3.0f);
    sumX = sumY = 0.f;
    for(unsigned int k = 0; k < Ang.size(); ++k)
    {
      // беремо angle з x-axis для точки прикладу
      const float & ang = Ang[k];

      // визначаємо чи є точка в межах вікна
      if (ang1 < ang2 && ang1 < ang && ang < ang2)
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
      else if (ang2 < ang1 &&
        ((ang > 0 && ang < ang2) || (ang > ang1 && ang < 2*pi) ))
      {
        sumX+=resX[k];
        sumY+=resY[k];
      }
    }

    // якщо вектор робив від цього вікна довше, ніж усі попередні вектори потім
це формує новий домінуючий напрям
    if (sumX*sumX + sumY*sumY > max)
    {
      // запам'ятовуємо найбільшу орієнтацію
      max = sumX*sumX + sumY*sumY;
      orientation = getAngle(sumX, sumY);
    }
  }

  // призначаємо орієнтацію домінуючого відповідному вектору
  ipt->orientation = orientation;
}

//-------------------------------------------------------

//! Беремо модифікований дескріптор.

void Surf::getDescriptor(bool bUpright)
{
  int y, x, sample_x, sample_y, count=0;
  int i = 0, ix = 0, j = 0, jx = 0, xs = 0, ys = 0;
  float scale, *desc, dx, dy, mdx, mdy, co, si;
  float gauss_s1 = 0.f, gauss_s2 = 0.f;
  float rx = 0.f, ry = 0.f, rrx = 0.f, rry = 0.f, len = 0.f;
  float cx = -0.5f, cy = 0.f; //Підобласть зосереджується для 4x4 блока гауса
```

```
Ipoint *ipt = &ipts[index];
scale = ipt->scale;
x = fRound(ipt->x);
y = fRound(ipt->y);
desc = ipt->descriptor;

if (bUpright)
{
  co = 1;
  si = 0;
}
else
{
  co = cos(ipt->orientation);
  si = sin(ipt->orientation);
}

i = -8;

//Розраховуємо дескриптор для цієї особливої точки
while(i < 12)
{
  j = -8;
  i = i-4;

  cx += 1.f;
  cy = -0.5f;

  while(j < 12)
  {
    dx=dy=mdx=mdy=0.f;
    cy += 1.f;

    j = j - 4;

    ix = i + 5;
    jx = j + 5;

    xs = fRound(x + ( -jx*scale*si + ix*scale*co));
    ys = fRound(y + ( jx*scale*co + ix*scale*si));

    for (int k = i; k < i + 9; ++k)
    {
      for (int l = j; l < j + 9; ++l)
      {
        //Беремо координати визначеної точки та повертаємо їх
        sample_x = fRound(x + (-l*scale*si + k*scale*co));
        sample_y = fRound(y + ( l*scale*co + k*scale*si));

        //Беремо the gaussian weighted x and y responses
        gauss_s1 = gaussian(xs-sample_x,ys-sample_y,2.5f*scale);
        rx = haarX(sample_y, sample_x, 2*fRound(scale));
        ry = haarY(sample_y, sample_x, 2*fRound(scale));

        //Беремо блок Гусса x та y відповідно на вісь обертання
        rrx = gauss_s1*(-rx*si + ry*co);
        rry = gauss_s1*(rx*co + ry*si);

        dx += rrx;
        dy += rry;
        mdx += fabs(rrx);
        mdy += fabs(rry);

      }
    }

    //Додаємо значення до дескриптора вектора
    gauss_s2 = gaussian(cx-2.0f,cy-2.0f,1.5f);
```

```
      desc[count++] = dx*gauss_s2;
      desc[count++] = dy*gauss_s2;
      desc[count++] = mdx*gauss_s2;
      desc[count++] = mdy*gauss_s2;

      len += (dx*dx + dy*dy + mdx*mdx + mdy*mdy) * gauss_s2*gauss_s2;

      j += 9;
    }
    i += 9;
  }

  //конвертуємо до Unit Vector
  len = sqrt(len);
  for(int i = 0; i < 64; ++i)
    desc[i] /= len;

}


//-------------------------------------------------------

//! Розраховуємо значення в  2d гаусciані в x,y
inline float Surf::gaussian(int x, int y, float sig)
{
  return (1.0f/(2.0f*pi*sig*sig)) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

//! Розраховуємо значення в  2d гаусciані в x,y
inline float Surf::gaussian(float x, float y, float sig)
{
  return 1.0f/(2.0f*pi*sig*sig) * exp( -(x*x+y*y)/(2.0f*sig*sig));
}

//-------------------------------------------------------

//! Розраховуємо вейвлет Хаара в  x напрямку
inline float Surf::haarX(int row, int column, int s)
{
  return BoxIntegral(img, row-s/2, column, s, s/2)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s, s/2);
}

//-------------------------------------------------------

//! Розраховуємо вейвлет Хаара в  y напрямку
inline float Surf::haarY(int row, int column, int s)
{
  return BoxIntegral(img, row, column-s/2, s/2, s)
    -1 * BoxIntegral(img, row-s/2, column-s/2, s/2, s);
}

//-------------------------------------------------------

//! Беремо вугол з +ve x-axis для вектора  (X Y)
float Surf::getAngle(float X, float Y)
{
  if(X > 0 && Y >= 0)
    return atan(Y/X);

  if(X < 0 && Y >= 0)
    return pi - atan(-Y/X);

  if(X < 0 && Y < 0)
    return pi + atan(Y/X);
```

```
  if(X > 0 && Y < 0)
    return 2*pi - atan(-Y/X);

  return 0;
}
```

surf.h – файл заголовків

```
/*************************************************************
* --- Програма розпізнання біопараметричних характеристик методом SURF --- *
*************************************************************/

#ifndef SURF_H
#define SURF_H

#include <cv.h>
#include "ipoint.h"
#include "integral.h"

#include <vector>

class Surf {

public:

//! Стандартний констуктор(img є цілочислене зображення)
Surf(IplImage *img, std::vector<Ipoint> &ipts);

//! Опишемо усі особливості у векторі, що поставляється
void getDescriptors(bool bUpright = false);

private:

//---------------- Private Functions -----------------//

//! Призначаємо поточнеIpoint на орієнтацію
void getOrientation();

//! Беремо дескриптор.
void getDescriptor(bool bUpright = false);

//! Розраховуємо значення в  2d гаусціані в x,y
inline float gaussian(int x, int y, float sig);
inline float gaussian(float x, float y, float sig);

//! Розраховуємо вейвлет Хаара в  x and y directions
inline float haarX(int row, int column, int size);
inline float haarY(int row, int column, int size);

//! Беремо the angle з +ve x-axis of the vector given by [X Y]
float getAngle(float X, float Y);


//---------------- Private Variables -----------------//

//! Цілочисельне зображення  де Ipoints визначений
IplImage *img;

//! Ipoints вектор
IpVec &ipts;

//! Індексуємо поточнеIpoint в вектор
int index;
};

 #endif
```

utils.cpp – опис підпрограми, яка реалізує утиліту

```
/*********************************************************
* --- Програма розпізнання біопараметричних характеристик методом SURF --- *
*
* *
*********************************************************/

#include <highgui.h>
#include <iostream>
#include <fstream>
#include <time.h>

#include "utils.h"

using namespace std;

//--------------------------------------------------------

static const int NCOLOURS = 8;
static const CvScalar COLOURS [] = {cvScalar(255,0,0), cvScalar(0,255,0),
cvScalar(0,0,255), cvScalar(255,255,0),
cvScalar(0,255,255), cvScalar(255,0,255),
cvScalar(255,255,255), cvScalar(0,0,0)};

//--------------------------------------------------------

//! Виводимо повідомлення про помилку, та завершуємо програму
void error(const char *msg)
{
cout << "\nError: " << msg;
getchar();
exit(0);
}

//--------------------------------------------------------

//! Показуємо зображення й чекаємо натискання клавіши
void showImage(const IplImage *img)
{
cvNamedWindow("Surf", CV_WINDOW_AUTOSIZE);
cvShowImage("Surf", img);
cvWaitKey(0);
}

//--------------------------------------------------------

//! Показуємо зображення у головному вікні й чекаємо натискання клавіши
void showImage(char *title,const IplImage *img)
{
cvNamedWindow(title, CV_WINDOW_AUTOSIZE);
cvShowImage(title, img);
cvWaitKey(0);
}

//--------------------------------------------------------

// Конвертуємо зображення по одному каналу32F
IplImage *getGray(const IplImage *img)
{
// Перевіряємо, ми поставляли ненульовий img покажчик
if (!img) error("Не в змозі створити зображення у градаціях сірого кольору.
Немає зображення, що поставляється");

IplImage* gray8, * gray32;

gray32 = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
```

```
if( img->nChannels == 1 )
gray8 = (IplImage *) cvClone( img );
else {
gray8 = cvCreateImage( cvGetSize(img), IPL_DEPTH_8U, 1 );
cvCvtColor( img, gray8, CV_BGR2GRAY );
}

cvConvertScale( gray8, gray32, 1.0 / 255.0, 0 );

cvReleaseImage( &gray8 );
return gray32;
}

//-------------------------------------------------------

//! Виводимо всі Ipoints у забезпеченому векторі
void drawIpoints(IplImage *img, vector<Ipoint> &ipts, int tailSize)
{
Ipoint *ipt;
float s, o;
int r1, c1, r2, c2, lap;

for(unsigned int i = 0; i < ipts.size(); i++)
{
ipt = &ipts.at(i);
s = (2.5f * ipt->scale);
o = ipt->orientation;
lap = ipt->laplacian;
r1 = fRound(ipt->y);
c1 = fRound(ipt->x);
c2 = fRound(s * cos(o)) + c1;
r2 = fRound(s * sin(o)) + r1;

if (o) // Зелена лінія вказує орієнтацію
cvLine(img, cvPoint(c1, r1), cvPoint(c2, r2), cvScalar(0, 255, 0));
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(c1,r1), 1, cvScalar(0, 255, 0),-1);

if (lap == 1)
{ // Блакитні круги вказують темні краплі на світлих фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(255, 0, 0),1);
}
else if (lap == 0)
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 0, 255),1);
}
else if (lap == 9)
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 255, 0),1);
}

// Виводимо рух з  ipoint dx та dy
if (tailSize)
{
cvLine(img, cvPoint(c1,r1),
cvPoint(int(c1+ipt->dx*tailSize), int(r1+ipt->dy*tailSize)),
cvScalar(255,255,255), 1);
}
}
}

//-------------------------------------------------------

//! Виводимо єдину особливість на зображенні
void drawIpoint(IplImage *img, Ipoint &ipt, int tailSize)
{
float s, o;
int r1, c1, r2, c2, lap;
```

```
s = (2.5f * ipt.scale);
o = ipt.orientation;
lap = ipt.laplacian;
r1 = fRound(ipt.y);
c1 = fRound(ipt.x);

// Зелена лінія вказує орієнтацію
if (o) // Зелена лінія вказує орієнтацію
{
c2 = fRound(s * cos(o)) + c1;
r2 = fRound(s * sin(o)) + r1;
cvLine(img, cvPoint(c1, r1), cvPoint(c2, r2), cvScalar(0, 255, 0));
}
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(c1,r1), 1, cvScalar(0, 255, 0),-1);

if (lap >= 0)
{ // Блакитні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(255, 0, 0),1);
}
else
{ // Червоні круги вказують світлі краплі на темних фонах
cvCircle(img, cvPoint(c1,r1), fRound(s), cvScalar(0, 0, 255),1);
}

// Виводимо рух з  ipoint dx and dy
if (tailSize)
{
cvLine(img, cvPoint(c1,r1),
cvPoint(int(c1+ipt.dx*tailSize), int(r1+ipt.dy*tailSize)),
cvScalar(255,255,255), 1);
}
}

//-------------------------------------------------------

//! Виводимо єдину особливість на зображенні
void drawPoint(IplImage *img, Ipoint &ipt)
{
float s, o;
int r1, c1;

s = 3;
o = ipt.orientation;
r1 = fRound(ipt.y);
c1 = fRound(ipt.x);

cvCircle(img, cvPoint(c1,r1), fRound(s), COLOURS[ipt.clusterIndex%NCOLOURS], -
1);
cvCircle(img, cvPoint(c1,r1), fRound(s+1),
COLOURS[(ipt.clusterIndex+1)%NCOLOURS], 2);

}

//-------------------------------------------------------

//! Виводимо єдину особливість на зображенні
void drawPoints(IplImage *img, vector<Ipoint> &ipts)
{
float s, o;
int r1, c1;

for(unsigned int i = 0; i < ipts.size(); i++)
{
s = 3;
o = ipts[i].orientation;
r1 = fRound(ipts[i].y);
c1 = fRound(ipts[i].x);
```

```
cvCircle(img, cvPoint(c1,r1), fRound(s), COLOURS[ipts[i].clusterIndex%NCOLOURS],
-1);
cvCircle(img, cvPoint(c1,r1), fRound(s+1),
COLOURS[(ipts[i].clusterIndex+1)%NCOLOURS], 2);
}
}

//-------------------------------------------------------

//! Виводимо дескрипторні вікна навкруги Ipoints у забезпеченому векторі
void drawWindows(IplImage *img, vector<Ipoint> &ipts)
{
Ipoint *ipt;
float s, o, cd, sd;
int x, y;
CvPoint2D32f src[4];

for(unsigned int i = 0; i < ipts.size(); i++)
{
ipt = &ipts.at(i);
s = (10 * ipt->scale);
o = ipt->orientation;
y = fRound(ipt->y);
x = fRound(ipt->x);
cd = cos(o);
sd = sin(o);

src[0].x=sd*s+cd*s+x; src[0].y=-cd*s+sd*s+y;
src[1].x=sd*s+cd*-s+x; src[1].y=-cd*s+sd*-s+y;
src[2].x=sd*-s+cd*-s+x; src[2].y=-cd*-s+sd*-s+y;
src[3].x=sd*-s+cd*s+x; src[3].y=-cd*-s+sd*s+y;

if (o) // Виводимо лінію орієнтації
cvLine(img, cvPoint(x, y),
cvPoint(fRound(s*cd + x), fRound(s*sd + y)), cvScalar(0, 255, 0),1);
else // Зелена точка, якщо, користуючись вертикальною версією
cvCircle(img, cvPoint(x,y), 1, cvScalar(0, 255, 0),-1);


// Виводимо квадрат навколо точки
cvLine(img, cvPoint(fRound(src[0].x), fRound(src[0].y)),
cvPoint(fRound(src[1].x), fRound(src[1].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[1].x), fRound(src[1].y)),
cvPoint(fRound(src[2].x), fRound(src[2].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[2].x), fRound(src[2].y)),
cvPoint(fRound(src[3].x), fRound(src[3].y)), cvScalar(255, 0, 0),2);
cvLine(img, cvPoint(fRound(src[3].x), fRound(src[3].y)),
cvPoint(fRound(src[0].x), fRound(src[0].y)), cvScalar(255, 0, 0),2);

}
}

//-------------------------------------------------------

// Виводимо фігуру FPS у зображенні (вимагає щонайменше 2 виклики)
void drawFPS(IplImage *img)
{
static int counter = 0;
static clock_t t;
static float fps;
char fps_text[20];
CvFont font;
cvInitFont(&font,CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC, 1.0,1.0,0,2);

// додаємо fps зображення (кожні 10 фреймів)
if (counter > 10)
{
fps = (10.0f/(clock()-t) * CLOCKS_PER_SEC);
t=clock();
```

```
counter = 0;
}

// Інкрементуємо лічильник
++counter;

// Беремо зображення з рядка
sprintf(fps_text,"FPS: %.2f",fps);

// Виводимо рядок на зображенні
cvPutText (img,fps_text,cvPoint(10,25), &font, cvScalar(255,255,0));
}

//--------------------------------------------------------

//! Зберігаємо SURF зображення до файлу
void saveSurf(char *filename, vector<Ipoint> &ipts)
{
ofstream outfile(filename);

// виводимо довжину дескриптора
outfile << "64\n";
outfile << ipts.size() << "\n";

// створюємо лінію виведення: координати x y
for(unsigned int i=0; i < ipts.size(); i++)
{
outfile << ipts.at(i).scale << " ";
outfile << ipts.at(i).x << " ";
outfile << ipts.at(i).y << " ";
outfile << ipts.at(i).orientation << " ";
outfile << ipts.at(i).laplacian << " ";
outfile << ipts.at(i).scale << " ";
for(int j=0; j<64; j++)
outfile << ipts.at(i).descriptor[j] << " ";

outfile << "\n";
}

outfile.close();
}

//--------------------------------------------------------

//! Завантажуємо SURF зображення з файлу
void loadSurf(char *filename, vector<Ipoint> &ipts)
{
int descriptorLength, count;
ifstream infile(filename);

// очищуємо ipts перший вектор
ipts.clear();

// читаємо дескриптор довжини/числа ipoints
infile >> descriptorLength;
infile >> count;

// для кожної ipoint
for (int i = 0; i < count; i++)
{
Ipoint ipt;

// читаємо значення
infile >> ipt.scale;
infile >> ipt.x;
infile >> ipt.y;
infile >> ipt.orientation;
infile >> ipt.laplacian;
infile >> ipt.scale;
```

```
// читаємо дескриптор компонент
for (int j = 0; j < 64; j++)
infile >> ipt.descriptor[j];

ipts.push_back(ipt);

}
}

//-------------------------------------------------------

//-------------------------------------------------------
```

utils.h – заголовочний файл

```
/*********************************************************
* --- Програма розпізнання біопараметричних характеристик методом SURF --- *
*********************************************************/

#ifndef UTILS_H
#define UTILS_H

#include <cv.h>
#include "ipoint.h"

#include <vector>


//! Виводимо повідомлення про помилку, та завершуємо програму
void error(const char *msg);

//! Показуємо зображення й чекаємо натискання клавіши
void showImage(const IplImage *img);

//! Показуємо зображення у головному вікні й чекаємо натискання клавіши
void showImage(char *title,const IplImage *img);

// Конвертуємо зображення по одному каналу32F
IplImage* getGray(const IplImage *img);

//! Виводимо єдину особливість на зображенні
void drawIpoint(IplImage *img, Ipoint &ipt, int tailSize = 0);

//! Виводимо всі Ipoints у забезпеченому векторі
void drawIpoints(IplImage *img, std::vector<Ipoint> &ipts, int tailSize = 0);

//! Виводимо дескрипторні вікна навкругиIpoints у забезпеченому векторі
void drawWindows(IplImage *img, std::vector<Ipoint> &ipts);

// Виводимо фігуру FPS on the image (вимагає щонайменше 2 викликів)
void drawFPS(IplImage *img);

//! Виводимо точку в позиції на зображенні
void drawPoint(IplImage *img, Ipoint &ipt);

//! Виводимо точку для всіх зображень
void drawPoints(IplImage *img, std::vector<Ipoint> &ipts);

//! Зберігаємо SURF зображення до файлу
void saveSurf(char *filename, std::vector<Ipoint> &ipts);

//! Завантажуємо SURF зображення з файлу
void loadSurf(char *filename, std::vector<Ipoint> &ipts);

//! Round float to nearest integer
inline int fRound(float flt)
{
return (int) floor(flt+0.5f);
}

#endif
```

ipoint.cpp – підпрограма визначення базових точок

```
/*************************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*
*
*************************************************************/

#include <cv.h>
#include <vector>

#include "ipoint.h"

//! Формуємо  IpPairVec з відповідних ipts
void getMatches(IpVec &ipts1, IpVec &ipts2, IpPairVec &matches)
{
  float dist, d1, d2;
  Ipoint *match;

  matches.clear();

  for(unsigned int i = 0; i < ipts1.size(); i++)
  {
    d1 = d2 = FLT_MAX;

    for(unsigned int j = 0; j < ipts2.size(); j++)
    {
      dist = ipts1[i] - ipts2[j];

      if(dist<d1) // якщо це зображення відповідає краще, ніж поточно краще
всього
      {
        d2 = d1;
        d1 = dist;
        match = &ipts2[j];
      }
      else if(dist<d2) // це зображення відповідає краще, ніж по-друге краще
всього
      {
        d2 = dist;
      }
    }

    // Якщо розташування d1:d2 ratio < 0.65 ipoints
    if(d1/d2 < 0.65)
    {
      // Запам'ятовуємо зміни у позиції
      ipts1[i].dx = match->x - ipts1[i].x;
      ipts1[i].dy = match->y - ipts1[i].y;
      matches.push_back(std::make_pair(ipts1[i], *match));
    }
  }
}

//
// Ця функція користується CV_RANSAC (
//

//-------------------------------------------------------

//! Шукаємо гомографію між визначеними точками, та перетворюємо src_corners до
dst_corners
int translateCorners(IpPairVec &matches, const CvPoint src_corners[4], CvPoint
dst_corners[4])
{
#ifndef WINDOWS
  double h[9];
```

```
  CvMat _h = cvMat(3, 3, CV_64F, h);
  std::vector<CvPoint2D32f> pt1, pt2;
  CvMat _pt1, _pt2;

  int n = (int)matches.size();
  if( n < 4 ) return 0;

  // Встановлюємо вектор корректного розміру
  pt1.resize(n);
  pt2.resize(n);

  // Копіюємо Ipoints з поточного вектора до cvPoint векторів
  for(int i = 0; i < n; i++ )
  {
    pt1[i] = cvPoint2D32f(matches[i].second.x, matches[i].second.y);
    pt2[i] = cvPoint2D32f(matches[i].first.x, matches[i].first.y);
  }
  _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
  _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );

  // Шукаємо гомографію (перетворення) між двома наборами точок
  if(!cvFindHomography(&_pt1, &_pt2, &_h, CV_RANSAC, 5))  //
    return 0;

  // перетворюємо src_corners до dst_corners використовуючи  гомографію
(перетворення)
  for(int i = 0; i < 4; i++ )
  {
    double x = src_corners[i].x, y = src_corners[i].y;
    double Z = 1./(h[6]*x + h[7]*y + h[8]);
    double X = (h[0]*x + h[1]*y + h[2])*Z;
    double Y = (h[3]*x + h[4]*y + h[5])*Z;
    dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));
  }
#endif
  return 1;
}
```

ipoint.h – заголовочний файл

```
/*********************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*
*                                                        *
*********************************************************/

#ifndef IPOINT_H
#define IPOINT_H

#include <vector>
#include <math.h>

//-------------------------------------------------------

class Ipoint; // Передопис
typedef std::vector<Ipoint> IpVec;
typedef std::vector<std::pair<Ipoint, Ipoint> > IpPairVec;

//-------------------------------------------------------

//! Ipoint операції
void getMatches(IpVec &ipts1, IpVec &ipts2, IpPairVec &matches);
int translateCorners(IpPairVec &matches, const CvPoint src_corners[4], CvPoint
dst_corners[4]);

//-------------------------------------------------------

class Ipoint {

public:

  //! деструктор
  ~Ipoint() {};

  //! конструктор
  Ipoint() : orientation(0) {};

  //! Визначає відстань простору дескрипторів між  Ipoints
  float operator-(const Ipoint &rhs)
  {
    float sum=0.f;
    for(int i=0; i < 64; ++i)
      sum += (this->descriptor[i] - rhs.descriptor[i])*(this->descriptor[i] -
rhs.descriptor[i]);
    return sqrt(sum);
  };

  //! Координати визначених базових точок
  float x, y;

  //! Визначає градацію
  float scale;

  //! Орієнтація мала розміри з +ve x-axis
  float orientation;

  //! Використовуємо лапласіан для швидких відповідних цілей
  int laplacian;

  //! Вектор дескрипторів компонентів
  float descriptor[64];

  //! Місце для зрушених точок
  float dx, dy;
```

```
  //! Використовуємо запам’ятовування індексу
  int clusterIndex;
};

//---------------------------------------------------------


#endif
```

integral.cpp – робота з зображенням

```
/*************************************************************
* --- Програма розпізнання біопараметричних характеристик методом SURF --- *
* *
*************************************************************/

#include "utils.h"

#include "integral.h"

//! розраховуємо цілочисельне зображення . Переймає на себе початкове
зображення, щоб бути 32-бітною float точкою. Повертає IplImage 32-бітну float
форму.
IplImage *Integral(IplImage *source)
{
// конвертує зображення в один канал 32f
IplImage *img = getGray(source);
IplImage *int_img = cvCreateImage(cvGetSize(img), IPL_DEPTH_32F, 1);

// встановлюємо змінні, бо дані мають доступ
int height = img->height;
int width = img->width;
int step = img->widthStep/sizeof(float);
float *data = (float *) img->imageData;
float *i_data = (float *) int_img->imageData;

// тільки перша колонка
float rs = 0.0f;
for(int j=0; j<width; j++)
{
rs += data[j];
i_data[j] = rs;
}

// осередки, що залишилися, - сума вище і вліво
for(int i=1; i<height; ++i)
{
rs = 0.0f;
for(int j=0; j<width; ++j)
{
rs += data[i*step+j];
i_data[i*step+j] = rs + i_data[(i-1)*step+j];
}
}

// звільняємо сіре зображення
cvReleaseImage(&img);

// повертаємо цілочисельне зображення
return int_img;
}
```

integral.h – заголовочний файл

```
/*************************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*
*                                                            *
*************************************************************/

#ifndef INTEGRAL_H
#define INTEGRAL_H

#include <algorithm>  // запит для std::min/max

// невизначений VS макрос
#ifdef min
  #undef min
#endif

#ifdef max
  #undef max
#endif

#include <cv.h>

//! розраховуємо цілочисельне зображення  з image img.
IplImage *Integral(IplImage *img);


//! Обчислюємо суму пікселів в межах прямокутника, вказаного верхнім лівим,
запускаємо координату і розмір
inline float BoxIntegral(IplImage *img, int row, int col, int rows, int cols)
{
  float *data = (float *) img->imageData;
  int step = img->widthStep/sizeof(float);

  // Віднімання для рядків/колонок.
  int r1 = std::min(row,          img->height) - 1;
  int c1 = std::min(col,          img->width)  - 1;
  int r2 = std::min(row + rows,   img->height) - 1;
  int c2 = std::min(col + cols,   img->width)  - 1;

  float A(0.0f), B(0.0f), C(0.0f), D(0.0f);
  if (r1 >= 0 && c1 >= 0) A = data[r1 * step + c1];
  if (r1 >= 0 && c2 >= 0) B = data[r1 * step + c2];
  if (r2 >= 0 && c1 >= 0) C = data[r2 * step + c1];
  if (r2 >= 0 && c2 >= 0) D = data[r2 * step + c2];

  return std::max(0.f, A - B - C + D);
}

#endif
```

fasthessian.cpp – реалізація гессіана

```
/*************************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*                                                            *
*************************************************************/

#include "integral.h"
#include "ipoint.h"
#include "utils.h"

#include <vector>

#include "responselayer.h"
#include "fasthessian.h"



using namespace std;

//-------------------------------------------------------

//! Конструктор без зображення
FastHessian::FastHessian(std::vector<Ipoint> &ipts,
                         const int octaves, const int intervals, const int
init_sample,
                         const float thresh)
                         : ipts(ipts), i_width(0), i_height(0)
{
  // Зберігаємо встановлені параметри
  saveParameters(octaves, intervals, init_sample, thresh);
}

//-------------------------------------------------------

//! Конструктор з зображенням
FastHessian::FastHessian(IplImage *img, std::vector<Ipoint> &ipts,
                         const int octaves, const int intervals, const int
init_sample,
                         const float thresh)
                         : ipts(ipts), i_width(0), i_height(0)
{
  // Зберігаємо встановлені параметри
  saveParameters(octaves, intervals, init_sample, thresh);

  // Встановлюємо поточне зображення
  setIntImage(img);
}

//-------------------------------------------------------

FastHessian::~FastHessian()
{
  for (unsigned int i = 0; i < responseMap.size(); ++i)
  {
    delete responseMap[i];
  }
}

//-------------------------------------------------------

//! Зберігаємо параметри
void FastHessian::saveParameters(const int octaves, const int intervals,
                                 const int init_sample, const float thresh)
{
  // Ініціалізуємо змінні з перевіреними граничними значеннями
  this->octaves =
```

```
    (octaves > 0 && octaves <= 4 ? octaves : OCTAVES);
  this->intervals =
    (intervals > 0 && intervals <= 4 ? intervals : INTERVALS);
  this->init_sample =
    (init_sample > 0 && init_sample <= 6 ? init_sample : INIT_SAMPLE);
  this->thresh = (thresh >= 0 ? thresh : THRES);
}


//-------------------------------------------------------

//! Встановлюємо або перевстановлюємо джерело Цілочисельного зображення
void FastHessian::setIntImage(IplImage *img)
{
  // Змінюємо дежерело зображень
  this->img = img;

  i_height = img->height;
  i_width = img->width;
}

//-------------------------------------------------------

//! Знайходимо, що зображення змальовує і вписуємо у вектор особливостей
void FastHessian::getIpoints()
{
  // фільтруємо карту індексів
  static const int filter_map [OCTAVES][INTERVALS] = {{0,1,2,3}, {1,3,4,5},
{3,5,6,7}, {5,7,8,9}, {7,9,10,11}};

  // Очищуємо вектор існування ipts
  ipts.clear();

  // Будуємо карту відповідностей
  buildResponseMap();

  // Беремо шар відповідностей
  ResponseLayer *b, *m, *t;
  for (int o = 0; o < octaves; ++o) for (int i = 0; i <= 1; ++i)
  {
    b = responseMap.at(filter_map[o][i]);
    m = responseMap.at(filter_map[o][i+1]);
    t = responseMap.at(filter_map[o][i+2]);

    // цикл над шаром середньої відповідності в щільності самого  розкиданого
шару (завжди вищий), щоб знайти максимум через масштаб і простір
    for (int r = 0; r < t->height; ++r)
    {
      for (int c = 0; c < t->width; ++c)
      {
        if (isExtremum(r, c, t, m, b))
        {
          interpolateExtremum(r, c, t, m, b);
        }
      }
    }
  }
}

//-------------------------------------------------------

//! Будуємо карту відповідностей DoH
void FastHessian::buildResponseMap()
{
  // Розраховуємо відповідності для перших чотирьох октетів:
  // Oct1: 9,  15, 21, 27
  // Oct2: 15, 27, 39, 51
  // Oct3: 27, 51, 75, 99
  // Oct4: 51, 99, 147,195
```

```
  // Oct5: 99, 195,291,387

  // Звільняємо пам'ять і очищуємо усі шари відповідності
  for(unsigned int i = 0; i < responseMap.size(); ++i)
    delete responseMap[i];
  responseMap.clear();

  // Беремо атрибути зображення
  int w = (i_width / init_sample);
  int h = (i_height / init_sample);
  int s = (init_sample);

  // Розраховуємо апрксимаційний детермінант значень гессіана
  if (octaves >= 1)
  {
    responseMap.push_back(new ResponseLayer(w,   h,   s,   9));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   15));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   21));
    responseMap.push_back(new ResponseLayer(w,   h,   s,   27));
  }

  if (octaves >= 2)
  {
    responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 39));
    responseMap.push_back(new ResponseLayer(w/2, h/2, s*2, 51));
  }

  if (octaves >= 3)
  {
    responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 75));
    responseMap.push_back(new ResponseLayer(w/4, h/4, s*4, 99));
  }

  if (octaves >= 4)
  {
    responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 147));
    responseMap.push_back(new ResponseLayer(w/8, h/8, s*8, 195));
  }

  if (octaves >= 5)
  {
    responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 291));
    responseMap.push_back(new ResponseLayer(w/16, h/16, s*16, 387));
  }

  // Отримуємо відповідно зображенню
  for (unsigned int i = 0; i < responseMap.size(); ++i)
  {
    buildResponseLayer(responseMap[i]);
  }
}

//-------------------------------------------------------

//! Обчислюємо відповідний DoH для забезпечуваного шару
void FastHessian::buildResponseLayer(ResponseLayer *rl)
{
  float *responses = rl->responses;          // збереження відповідностей
  unsigned char *laplacian = rl->laplacian; // збереження знаку лапласіана
  int step = rl->step;                       // розмір шагу для цього фільтру
  int b = (rl->filter - 1) / 2 + 1;          // границя для цього фільтру
  int l = rl->filter / 3;                    // частка для цього фільтру(розмір
фільтру / 3)
  int w = rl->filter;                        // розмір фільтру
  float inverse_area = 1.f/(w*w);            // чинник нормалізації
  float Dxx, Dyy, Dxy;

  for(int r, c, ar = 0, index = 0; ar < rl->height; ++ar)
  {
```

```
    for(int ac = 0; ac < rl->width; ++ac, index++)
    {
      // беремо координати зображення
      r = ar * step;
      c = ac * step;

      // Розраховуємо відповідні компоненти
      Dxx = BoxIntegral(img, r - l + 1, c - b, 2*l - 1, w)
          - BoxIntegral(img, r - l + 1, c - l / 2, 2*l - 1, l)*3;
      Dyy = BoxIntegral(img, r - b, c - l + 1, w, 2*l - 1)
          - BoxIntegral(img, r - l / 2, c - l + 1, l, 2*l - 1)*3;
      Dxy = + BoxIntegral(img, r - l, c + 1, l, l)
            + BoxIntegral(img, r + 1, c - l, l, l)
            - BoxIntegral(img, r - l, c - l, l, l)
            - BoxIntegral(img, r + 1, c + 1, l, l);

      // Нормалізуємо фільтр відповідностей відносно розміру
      Dxx *= inverse_area;
      Dyy *= inverse_area;
      Dxy *= inverse_area;

      // Беремо детремінант відповідності гессіана & знак лапласіана
      responses[index] = (Dxx * Dyy - 0.81f * Dxy * Dxy);
      laplacian[index] = (Dxx + Dyy >= 0 ? 1 : 0);

#ifdef RL_DEBUG
      // створюємо список координат зображень для кожної відповідності
      rl->coords.push_back(std::make_pair<int,int>(r,c));
#endif
    }
  }
}

//-------------------------------------------------------

//! Не функція Максимального Придушення
int FastHessian::isExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  // визначаємо кордони
  int layerBorder = (t->filter + 1) / (2 * t->step);
  if (r <= layerBorder || r >= t->height - layerBorder || c <= layerBorder || c
>= t->width - layerBorder)
    return 0;

  // перевіряємо точку-кандидат посередині шара
  float candidate = m->getResponse(r, c, t);
  if (candidate < thresh)
    return 0;

  for (int rr = -1; rr <=1; ++rr)
  {
    for (int cc = -1; cc <=1; ++cc)
    {
      // якщо будь-яка відповідність у 3x3x3 - це не більший максимум кандидата
      if (
        t->getResponse(r+rr, c+cc) >= candidate ||
        ((rr != 0 && cc != 0) && m->getResponse(r+rr, c+cc, t) >= candidate) ||
        b->getResponse(r+rr, c+cc, t) >= candidate
        )
        return 0;
    }
  }

  return 1;
}

//-------------------------------------------------------
```

```
//! Інтерполюємо простір масштабу екстремума до точності підпікселя, щоб
сформувати особливість зображення.
void FastHessian::interpolateExtremum(int r, int c, ResponseLayer *t,
ResponseLayer *m, ResponseLayer *b)
{
  // беремо шаг дистанції між фільтрами
  // перевіряємо проміжний фільтр який є середнім між вищи та нижчим
  int filterStep = (m->filter - b->filter);
  assert(filterStep > 0 && t->filter - m->filter == m->filter - b->filter);

  // Беремо відгалуження до фактичного розташування екстремуму
  double xi = 0, xr = 0, xc = 0;
  interpolateStep(r, c, t, m, b, &xi, &xr, &xc );

  // Якщо точка досить близька до фактичного екстремуму
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f )
  {
    Ipoint ipt;
    ipt.x = static_cast<float>((c + xc) * t->step);
    ipt.y = static_cast<float>((r + xr) * t->step);
    ipt.scale = static_cast<float>((0.1333f) * (m->filter + xi * filterStep));
    ipt.laplacian = static_cast<int>(m->getLaplacian(r,c,t));
    ipts.push_back(ipt);
  }
}

//-------------------------------------------------------

//! Виконуємо один крок інтерполяції екстремуму.
void FastHessian::interpolateStep(int r, int c, ResponseLayer *t, ResponseLayer
*m, ResponseLayer *b,
                                  double* xi, double* xr, double* xc )
{
  CvMat* dD, * H, * H_inv, X;
  double x[3] = { 0 };

  dD = deriv3D( r, c, t, m, b );
  H = hessian3D( r, c, t, m, b );
  H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  cvInvert( H, H_inv, CV_SVD );
  cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
  cvGEMM( H_inv, dD, -1, NULL, 0, &X, 0 );

  cvReleaseMat( &dD );
  cvReleaseMat( &H );
  cvReleaseMat( &H_inv );

  *xi = x[2];
  *xr = x[1];
  *xc = x[0];
}

//-------------------------------------------------------

//! Обчислюємо часткові похідні слова в x, y, і масштаб пікселя.
CvMat* FastHessian::deriv3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* dI;
  double dx, dy, ds;

  dx = (m->getResponse(r, c + 1, t) - m->getResponse(r, c - 1, t)) / 2.0;
  dy = (m->getResponse(r + 1, c, t) - m->getResponse(r - 1, c, t)) / 2.0;
  ds = (t->getResponse(r, c) - b->getResponse(r, c, t)) / 2.0;

  dI = cvCreateMat( 3, 1, CV_64FC1 );
  cvmSet( dI, 0, 0, dx );
  cvmSet( dI, 1, 0, dy );
  cvmSet( dI, 2, 0, ds );
```

```
  return dI;
}

//-------------------------------------------------------

//! Обчислюємо 3D матрицю гессіана для пікселя.
CvMat* FastHessian::hessian3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* H;
  double v, dxx, dyy, dss, dxy, dxs, dys;

  v = m->getResponse(r, c, t);
  dxx = m->getResponse(r, c + 1, t) + m->getResponse(r, c - 1, t) - 2 * v;
  dyy = m->getResponse(r + 1, c, t) + m->getResponse(r - 1, c, t) - 2 * v;
  dss = t->getResponse(r, c) + b->getResponse(r, c, t) - 2 * v;
  dxy = ( m->getResponse(r + 1, c + 1, t) - m->getResponse(r + 1, c - 1, t) -
          m->getResponse(r - 1, c + 1, t) + m->getResponse(r - 1, c - 1, t) ) /
4.0;
  dxs = ( t->getResponse(r, c + 1) - t->getResponse(r, c - 1) -
          b->getResponse(r, c + 1, t) + b->getResponse(r, c - 1, t) ) / 4.0;
  dys = ( t->getResponse(r + 1, c) - t->getResponse(r - 1, c) -
          b->getResponse(r + 1, c, t) + b->getResponse(r - 1, c, t) ) / 4.0;

  H = cvCreateMat( 3, 3, CV_64FC1 );
  cvmSet( H, 0, 0, dxx );
  cvmSet( H, 0, 1, dxy );
  cvmSet( H, 0, 2, dxs );
  cvmSet( H, 1, 0, dxy );
  cvmSet( H, 1, 1, dyy );
  cvmSet( H, 1, 2, dys );
  cvmSet( H, 2, 0, dxs );
  cvmSet( H, 2, 1, dys );
  cvmSet( H, 2, 2, dss );

  return H;
}

//-------------------------------------------------------
```

fasthessian.h – заголовочний файл

```
/*************************************************************
* --- Програма розпізнання біопараметричних характеристик методом SURF --- *
* *
*************************************************************/

#ifndef FASTHESSIAN_H
#define FASTHESSIAN_H

#include <cv.h>
#include "ipoint.h"

#include <vector>

class ResponseLayer;
static const int OCTAVES = 5;
static const int INTERVALS = 4;
static const float THRES = 0.0004f;
static const int INIT_SAMPLE = 2;


class FastHessian {

public:

//! Конструктор без зображення
FastHessian(std::vector<Ipoint> &ipts,
const int octaves = OCTAVES,
const int intervals = INTERVALS,
const int init_sample = INIT_SAMPLE,
const float thres = THRES);

//! Конструктор з зображенням
FastHessian(IplImage *img,
std::vector<Ipoint> &ipts,
const int octaves = OCTAVES,
const int intervals = INTERVALS,
const int init_sample = INIT_SAMPLE,
const float thres = THRES);

//! Деструктор
~FastHessian();

//! Зберігаємо параметри
void saveParameters(const int octaves,
const int intervals,
const int init_sample,
const float thres);

//! Встановлюємо або перевстановлюємо джерело цілочиселього зображення
void setIntImage(IplImage *img);

//! Знайходимо, що зображення змальовує і вписуємо у вектор особливостей
void getIpoints();

private:

//---------------- Private Functions -----------------//

//! Будуємо карту відповідностей DoH
void buildResponseMap();

//! Обчислюємо відповідний DoH для забезпечуваного шару
void buildResponseLayer(ResponseLayer *r);

//! 3x3x3 тест на екстремум
```

```
int isExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);

//! Функція інтерполяції – адаптовується для SIFT додатку
void interpolateExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b);
void interpolateStep(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b,
double* xi, double* xr, double* xc );
CvMat* deriv3D(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);
CvMat* hessian3D(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);

//---------------- Private Variables -----------------//

//! Точка цілочисельного зображення , та її атрибути
IplImage *img;
int i_width, i_height;

//! Посилаємося до вектора особливостей проходів за межами
std::vector<Ipoint> &ipts;

//! Стек відповідностей детермінанту значення гессіана
std::vector<ResponseLayer *> responseMap;

//! Число октетів
int octaves;

//! Число інтервалів між октетами
int intervals;

//! Початковий пробний крок для виявлення Ipoint
int init_sample;

//! Порогове значення для  відповідностей кіл
float thresh;
};


#endif
```

responselayer.h – заголовочний файл ля шару відповідностей

```
/*********************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*********************************************************/


//#define RL_DEBUG  // не треба коментувати для тесту шару відповідностей

class ResponseLayer
{
public:

  int width, height, step, filter;
  float *responses;
  unsigned char *laplacian;

  ResponseLayer(int width, int height, int step, int filter)
  {
    assert(width > 0 && height > 0);

    this->width = width;
    this->height = height;
    this->step = step;
    this->filter = filter;

    responses = new float[width*height];
    laplacian = new unsigned char[width*height];

    memset(responses,0,sizeof(float)*width*height);
    memset(laplacian,0,sizeof(unsigned char)*width*height);
  }

  ~ResponseLayer()
  {
    if (responses) delete [] responses;
    if (laplacian) delete [] laplacian;
  }

  inline unsigned char getLaplacian(unsigned int row, unsigned int column)
  {
    return laplacian[row * width + column];
  }

  inline unsigned char getLaplacian(unsigned int row, unsigned int column,
ResponseLayer *src)
  {
    int scale = this->width / src->width;

    #ifdef RL_DEBUG
    assert(src->getCoords(row, column) == this->getCoords(scale * row, scale *
column));
    #endif

    return laplacian[(scale * row) * width + (scale * column)];
  }

  inline float getResponse(unsigned int row, unsigned int column)
  {
    return responses[row * width + column];
  }

  inline float getResponse(unsigned int row, unsigned int column, ResponseLayer
*src)
  {
    int scale = this->width / src->width;
```

```
    #ifdef RL_DEBUG
    assert(src->getCoords(row, column) == this->getCoords(scale * row, scale *
column));
    #endif

    return responses[(scale * row) * width + (scale * column)];
  }

#ifdef RL_DEBUG
  std::vector<std::pair<int, int>> coords;

  inline std::pair<int,int> getCoords(unsigned int row, unsigned int column)
  {
    return coords[row * width + column];
  }

  inline std::pair<int,int> getCoords(unsigned int row, unsigned int column,
ResponseLayer *src)
  {
    int scale = this->width / src->width;
    return coords[(scale * row) * width + (scale * column)];
  }
#endif
};
```

kmeans.h – заголовочний файл

```
/*************************************************************
*  --- Програма розпізнання біопараметричних характеристик методом SURF ---
*
*************************************************************/

#include "ipoint.h"

#include <vector>
#include <time.h>
#include <stdlib.h>

//-------------------------------------------------------------
//
//-------------------------------------------------------------

class Kmeans {

public:

  //! Деструктор
  ~Kmeans() {};

  //! Конструктор
  Kmeans() {};

  //!
  void Run(IpVec *ipts, int clusters, bool init = false);

  //! Встановлюємо ipts до використання
  void SetIpoints(IpVec *ipts);

  //! Випадково поширюйте ``n'' кластерів
  void InitRandomClusters(int n);

  //! Призначемо Ipoints кластерам
  bool AssignToClusters();

  //! Розраховуємо нові центри кластерів
  void RepositionClusters();

  //! Функція вимірює відстань між  2 ipoints
  float Distance(Ipoint &ip1, Ipoint &ip2);

  //! Запам'ятовуємо вектор ipoints для цього руху
  IpVec *ipts;

  //! Запам'ятовуємо вектор центрів кластерів
  IpVec clusters;

};

//-------------------------------------------------------------

void Kmeans::Run(IpVec *ipts, int clusters, bool init)
{
  if (!ipts->size()) return;

  SetIpoints(ipts);

  if (init) InitRandomClusters(clusters);

  while (AssignToClusters());
  {
    RepositionClusters();
  }
```

```
}

//---------------------------------------------------------

void Kmeans::SetIpoints(IpVec *ipts)
{
  this->ipts = ipts;
}

//---------------------------------------------------------

void Kmeans::InitRandomClusters(int n)
{
  // очищуємо вектор кластеру
  clusters.clear();

  // Запускаємо генератор випадкових чисел
  srand((int)time(NULL));

  // додаємо 'n' випадкових ipoints до списку кластерів й ініціалізуємо центри
  for (int i = 0; i < n; ++i)
  {
    clusters.push_back(ipts->at(rand() % ipts->size()));
  }
}

//---------------------------------------------------------

bool Kmeans::AssignToClusters()
{
  bool Updated = false;

  // цикл над усіма Ipoints і призначаємо кожну найближчому кластеру
  for (unsigned int i = 0; i < ipts->size(); ++i)
  {
    float bestDist = FLT_MAX;
    int oldIndex = ipts->at(i).clusterIndex;

    for (unsigned int j = 0; j < clusters.size(); ++j)
    {
      float currentDist = Distance(ipts->at(i), clusters[j]);
      if (currentDist < bestDist)
      {
        bestDist = currentDist;
        ipts->at(i).clusterIndex = j;
      }
    }

    // визначаємо чи змінила точка кластер
    if (ipts->at(i).clusterIndex != oldIndex) Updated = true;
  }

  return Updated;
}

//---------------------------------------------------------

void Kmeans::RepositionClusters()
{
  float x, y, dx, dy, count;

  for (unsigned int i = 0; i < clusters.size(); ++i)
  {
    x = y = dx = dy = 0;
    count = 1;

    for (unsigned int j = 0; j < ipts->size(); ++j)
    {
      if (ipts->at(j).clusterIndex == i)
```

```
      {
        Ipoint ip = ipts->at(j);
        x += ip.x;
        y += ip.y;
        dx += ip.dx;
        dy += ip.dy;
        ++count;
      }
    }

    clusters[i].x = x/count;
    clusters[i].y = y/count;
    clusters[i].dx = dx/count;
    clusters[i].dy = dy/count;
  }
}

//-------------------------------------------------------

float Kmeans::Distance(Ipoint &ip1, Ipoint &ip2)
{
  return sqrt(pow(ip1.x - ip2.x, 2)
            + pow(ip1.y - ip2.y, 2)
            /*+ pow(ip1.dx - ip2.dx, 2)
            + pow(ip1.dy - ip2.dy, 2)*/);
}

//-------------------------------------------------------
```