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
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Пашева М.П.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2025 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 2025 року

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

Пашевій Марії Павлівні
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 57-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI*

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 2025 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	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Пашева М.П.
(прізвище та ініціали)

АНОТАЦІЯ

Пашева М.П. Програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Метою розробки є програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Результат роботи – програмна реалізація системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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 з ОС Windows 10/11.

Програму розроблено в середовищі Visual C++.

Ключові слова: кібербезпека, багатофакторна біометрична ідентифікація, BioAPI

ABSTRACT

Pasheva M.P. Software for a cybersecurity system for multifactor biometric identification using the BioAPI interface. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a cybersecurity system for multifactor biometric identification using the BioAPI interface.

The purpose of the development is to develop software for a cybersecurity system for multifactor biometric identification using the BioAPI interface.

The result of the work is a software implementation of a cybersecurity system for multifactor biometric identification using the BioAPI interface.

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 are provided.

The program can be used on PCs with Windows 10/11.

The program was developed in Visual C++.

Keywords: cybersecurity, multifactor biometric identification, BioAPI

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 є найбільш підходящим засобом ідентифікації та автентифікації осіб надійним і швидким способом за допомогою унікальних біологічних характеристик. Біометрія дозволяє ідентифікувати та автентифікувати особу на основі впізнаваних, перевірених, унікальних і конкретних даних.

Біометрична автентифікація порівнює дані про характеристики людини з біометричним «шаблоном» цієї особи, щоб визначити схожість.

– Спочатку зберігається еталонна модель.

– Потім збережені дані порівнюються з біометричними даними людини, які підлягають автентифікації.

У цьому режимі запитання: « Ви справді містер чи місіс X? »

Біометрична ідентифікація полягає у встановленні особистості людини.

– Мета полягає в тому, щоб отримати елемент біометричних даних цієї особи. Це може бути фотографія обличчя, запис голосу або відбиток пальця.

– Потім ці дані порівнюються з біометричними даними кількох інших осіб, які зберігаються в базі даних.

У цьому режимі запитання просте: « Хто ти? »

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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

– Огляд існуючих систем для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

– Дослідження системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

– Програмна реалізація системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI, 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

Система призначена для реалізації програмного забезпечення багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Зіткнувшись із шахрайством з документами та крадіжкою особистих даних, тероризмом і кіберзлочинністю, а також глобальними змінами законодавства, впроваджуються нові рішення біометричної безпеки.

Біометрію можна визначити як найбільш практичний засіб ідентифікації та ідентифікації людей надійним і швидким способом за допомогою унікальних біологічних характеристик.

Звичайно, підвищення популярності серед громадськості, величезне підвищення точності, багата пропозиція та падіння цін на датчики, IP-камери та програмне забезпечення полегшують встановлення біометричних систем.

Сьогодні багато програм використовують цю технологію.

Біометрична ідентифікація

Біометричні ідентифікатори

Існує два види біометрії:

– Фізіологічні вимірювання. Вони можуть бути як морфологічними, так і біологічними. Морфологічні ідентифікатори в основному складаються з відбитків пальців, форми руки, візерунка вени пальця, ока (райдужної оболонки та сітківки) та форми обличчя. Для біологічного аналізу медичні бригади та судово-медичні експерти можуть використовувати ДНК, кров, слину або сечу.

– Поведінкові вимірювання. Найпоширенішими є: розпізнавання голосу, характерна динаміка (швидкість руху пера, прискорення, тиск, що чиниться, нахил), динаміка натискання клавіш, те, як ми використовуємо предмети, хода, звук кроків, жести тощо.

Використовувані методи є предметом постійних досліджень і розробок і постійно вдосконалюються.

Однак різні види вимірювань не мають однакового рівня надійності.

Фізіологічні вимірювання зазвичай забезпечують стабільність протягом життя людини.

Наприклад, вони не схильні до стресу, на відміну від ідентифікації за допомогою поведінкових вимірювань.

1.2 Область застосування

Областю застосування системи є біометрична ідентифікація. Історично склалося так, що додатки, що використовують біометричні дані, ініціювали органи влади для контролю доступу військових та ідентифікації злочинців або цивільних осіб відповідно до жорстко регламентованої правової та технічної бази.

Сьогодні сектори, включаючи банківську справу, роздрібну торгівлю та мобільну комерцію, демонструють справжній апетит до переваг біометрії.

Найважливіше те, що за останні сім років обізнаність і визнання підвищилися, оскільки мільйони користувачів смартфонів розблоковують свої телефони за допомогою відбитка пальця або обличчя.

Але що такого особливого в біометрії?

Знову ж таки, біометричні системи чудові там, де ідентифікація та автентифікація є критичними.

Давайте швидко розглянемо найбільш типові випадки використання біометричних технологій:

1. Правоохоронні органи та громадська безпека (ідентифікація злочинців/підозрюваних).

2. Військові (ідентифікація противника/союзника).

3. Прикордонний, подорожній та міграційний контроль (ідентифікація мандрівника/мігранта/пасажира).

4. Цивільна ідентифікація (ідентифікація громадянина/резидента/виборця).

5. Охорона здоров’я та субсидії (ідентифікація пацієнта/бенефіціара/ медичного працівника).

6. Фізичний і логічний доступ (ідентифікація власника/користувача/ працівника/підрядника/партнера).

7. Комерційні програми (ідентифікація споживача/клієнта).

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI, 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

Система реєстрації особи MegaMatcher

MegaMatcher Identity Registration System (IDRS) забезпечує швидкий, точний і надійний збір біометричних даних. Настроюване рішення може спростити процес реєстрації та забезпечити безпечний і ефективний збір даних для багатьох різних сценаріїв. MegaMatcher IDRS можна використовувати як окреме рішення або в поєднанні з MegaMatcher Identity Management System (IDMS) і MegaMatcher Automated Biometric Identification System (ABIS).

Основні функції та можливості:

– Захоплення обличчя, сумісне з ICAO. Знімає високоякісні зображення обличчя, які відповідають міжнародним стандартам для документів, що посвідчують особу. Це забезпечує надійний збір біометричних даних.

– Адаптивне захоплення відбитків пальців. Для гнучкого захоплення відбитків пальців MegaMatcher IDRS інтегрується з широким спектром сканерів, підтримуючи понад 120 моделей пристроїв.

– Запобігання шахрайству. Система включає оцінку віку, біометричну перевірку живучості, запобігання багаторазовій реєстрації, а також виявлення втручання оператора.

– Безпека даних. MegaMatcher IDRS використовує шифрування під час зберігання конфіденційних біометричних і біографічних даних. У разі втрати апаратного забезпечення конфіденційні біометричні та біографічні дані неможливо відновити, тому дані залишаються захищеними.

– Сумісність із комплектами біометричної реєстрації MegaMatcher IDRS

працює з різними пристроями (сканерами, камерами, принтерами) для безпроблемної реєстрації. Він підтримує операційні системи Windows, Linux і Android.

Додатки

Система реєстрації ідентифікаційних даних MegaMatcher може бути налаштована для різних сценаріїв і пропонує універсальні рішення для додатків різного масштабу:

– Керування виборцями: MegaMatcher IDRS може забезпечити біометричну реєстрацію виборців (BVR) для створення безпечного та надійного реєстру виборців шляхом збору біометричних даних (таких як відбитки пальців, фотографії обличчя та райдужної оболонки ока) і біографічних даних (таких як ім'я, дата народження, стать, адреса тощо) під час реєстрації виборців.

– Реєстри національних ідентифікаційних номерів: система може використовуватися державними установами для реєстрації громадян для різноманітних цілей, таких як видача національних посвідчень особи, водійських прав або паспортів.

– Організації: MegaMatcher IDRS можна використовувати як реєстраційне рішення для компаній, громад та інших організацій, що задовольняє конкретні потреби, як-от адаптація співробітників, реєстрація відвідувачів або реєстрація одержувачів допомоги.

Функціональні можливості

Реєстрація особи

MegaMatcher IDRS спрощує процес реєстрації особи, забезпечуючи точний і безпечний збір даних, надаючи:

– Інтуїтивно зрозумілий досвід користувача. MegaMatcher IDRS пропонує зручний, покроковий майстер, який проводить людей через процес реєстрації, забезпечуючи плавну та ефективну роботу.

– Персоналізований збір біографічних даних. Система дозволяє збирати індивідуальні біографічні дані для кожної людини, сприяючи цілеспрямованому

спілкуванню та посилюючи заходи безпеки.

– Перевірки узгодженості даних. Кожній особі присвоюється унікальний ідентифікатор, створюючи чіткий зв’язок між зібраними даними та відповідною особою. Система також може виконувати такі перевірки:

- Біометричне виявлення живості – спеціалізований алгоритм може визначити, чи є обличчя у відеопотоці чи окремому кадрі «живим», таким чином запобігаючи спробам видати себе за фотографію іншої людини, показану перед камерою.
- Оцінка віку – вік людини визначається за наданим зображенням обличчя, щоб запобігти реєстрації неповнолітніх.
- Виявлення кількох реєстрацій – надані біометричні дані можна порівняти з усіма попередніми реєстраціями, щоб запобігти спробам видавання себе за іншу особу
- Виявлення втручання оператора – усі оператори також реєструються в системі, щоб перевірити, чи їхні біометричні дані не використовуються для реєстрації інших осіб.

Управління даними

MegaMatcher IDRS забезпечує ефективне управління даними та безпечний контроль даних:

– Керування користувачами. Система пропонує функції контролю та керування доступом і дозволами користувачів, забезпечуючи безпеку даних.

– Підтримка онлайн і офлайн режимів. Ця гнучкість дозволяє збирати дані в різних середовищах навіть без підключення до Інтернету.

– Експорт даних і звітність. MegaMatcher IDRS забезпечує легку та ефективну передачу даних через знімні диски або підключення до Інтернету. Він також генерує статистичні дані та звіти, що дозволяє аналізувати та контролювати ідентифікаційні дані.

Рисунок 2.1 – Інтерфейс користувача MegaMatcher IDRS

Сумісність з комплектами біометричної реєстрації

MegaMatcher IDRS можна використовувати в комплектах біометричної реєстрації як повне рішення для безперебійного, безпечного та простого процесу

реєстрації особи. Системну сумісність можна узгодити з декількома сканерами, камерами, принтерами або блокнотами для підпису. Це дає змогу програмному забезпеченню MegaMatcher IDRS об'єднувати різні функції реєстраційного набору в один єдиний інтерфейс користувача та спрощувати процес реєстрації. Завдяки технологічній гнучкості рішення можна змінювати для кожного унікального процесу реєстрації або функцій біометричного набору.

Біометрична система відвідуваності

Біометрична система відвідуваності – це вдосконалене програмне забезпечення для вимірювання часу, призначене для оптимізації відстеження відвідуваності та покращення управління робочою силою. Використовуючи біометричну технологію, ця система забезпечує точну та безпечну перевірку відвідуваності співробітників за допомогою відбитків пальців або розпізнавання обличчя. Усунувши можливість штампування друзів, підприємства можуть значно покращити цілісність своїх процесів обліку часу. Програмне забезпечення пропонує звітування даних у реальному часі, що дозволяє менеджерам відстежувати тенденції відвідуваності, відстежувати відпрацьовані години та ефективно керувати понаднормовими. Крім того, він легко інтегрується з системами нарахування заробітної плати, спрощуючи процес нарахування заробітної плати та зменшуючи адміністративне навантаження. Завдяки зручному для користувача інтерфейсу та параметрам, які можна налаштувати, система біометричної відвідуваності дозволяє організаціям впроваджувати політику відвідуваності, яка відповідає їхнім конкретним потребам. Використовуючи цю технологію, компанії можуть підвищити підзвітність, підвищити продуктивність працівників і отримати цінну інформацію про динаміку робочої сили.

Особливості біометричної системи відвідуваності:

Програмне забезпечення годинника часу.

– Відстеження лікарняних.

– Управління заробітною платою.

– База даних співробітників.

– Відстеження активності.

– GPS.

– Відстеження відвідуваності.

– Портал самообслуговування.

– Розрахунок понаднормової роботи.

– Мобільний доступ.

– Відстеження часу проекту.

– Панель активності.

– Профілі співробітників.

– Відстеження відпустки/відпустки.

– Звітність/Аналітика.

– Управління працівниками.

– Відстеження оплачуваних товарів.

– Відстеження робочої станції.

– Онлайн годинник часу.

– Запити на відпустку.

– Управління календарем.

– Управління розкладом.

– Онлайн перфокарта.

– Облік робочого часу співробітників.

– Портал співробітників.

– Біометричне розпізнавання.

– Планування співробітників.

– Контроль процесу затвердження.

– Автоматизоване планування.

– Управління відповідністю.

– Геозонування.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програми мною була використана мова програмування Visual C++. 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 програмуванні 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 багатовіконний 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 одновіконні й багатовіконні 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-орієнтованого 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. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

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 багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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 якості автентифікаційної інформації в цьому випадку беруться в увагу оригінальні й невід'ємні характеристики людини. Найбільше часто використовуються наступні з них:

1. Відбитки пальців. Відомо, що вони унікальні для кожної людини, причому не міняються протягом життя. Для сканування відбитків пальців застосовується найдешевше встаткування (у порівняні з іншими методами біопараметричної автентифікації), крім того, даний метод звичний для користувачів і не викликає яких-небудь побоювань. Однак вважається, що недорогі сканери відбитків пальців можна обдурити спеціально виготовленим штучним пальцем.

2. Рисунок райдужної оболонки ока. Це на сьогодні найбільш точний метод біопараметричної автентифікації. Але багато користувачів бояться процесу сканування райдужної оболонки, та й устаткування для сканування є дорогим. До того ж даний спосіб викликає дорікання з боку правозахисників. Вони говорять, що око людини несе багато інформації про стан його здоров'я, про зловживання спиртними напоями, наркотиками й т.д. Є побоювання, що цю інформацію про користувачів (побічну для процесу автентифікації) настроєна відповідним чином система може зберігати, після чого можливо її використання їм на шкоду.

3. Риси особи. Дана технологія розпізнавання вважається дуже перспективною, оскільки саме по рисах особи довідаються один одного люди. На жаль, системи, що реалізують даний метод, поки не блищать точністю.

Незважаючи на наявність на ринку альтернативних засобів автентифікації, може трапитися так, що без імен користувачів і паролів обійтися буде неможливо. От кілька рекомендацій для таких випадків:

– Web-сайти, які вимагають введення імен користувачів і паролів, відвідуйте лише в тому випадку, якщо в них реалізована технологія Secure Sockets Layer (SSL).

– Вибирайте надійні паролі довжиною не менш восьми символів. Вони повинні складатися із символів верхнього й нижнього регістрів, чисел і знаків пунктуації.

– Для кожної системи використовуйте особливий пароль (і, якщо можливо, особливе ім'я користувача). Якщо ваші облікові дані для однієї системи стануть надбанням зловмисника, він не зможе використовувати їх на інших системах.

– Якщо для кожного облікового запису ви будете застосовувати окреме ім'я користувача й пароль, у вас нагромадиться великий обсяг різних облікових даних. У цьому випадку рекомендується придбати недорогий засіб автентифікації за допомогою біометричних даних, наприклад пристрій зчитування відбитків пальців, що дозволить зберігати кожний набір імен користувача й паролів і автоматично реєструватися на Web-сайтах, пред'явивши біометричні дані.

– Не зберігайте на Web-сайтах відомості про свою кредитну карту або інші що дозволяє ідентифікувати вас інформацію.

Різні системи контрольованого забезпечення доступу можна розділити на три класи відповідно до того, що людина повинна пред'являти системі:

– те, що вона знає;

– те, чим вона володіє;

– те, що є частиною її самої.

Перший клас використовує різного роду шифри, що набираються людиною (наприклад, PIN-коди, криптографічні коди й т.п.).

Другий клас використовує шифри, передані за допомогою фізичних носіїв інформації (пластикові карти з магнітною смугою, електроні таблетки "touch memory", електроні token-пристрої, proximity-карти й т.д.).

Третій – біопараметричний клас принципово відрізняється тим, що ідентифікації піддається властиво особистість людини – його індивідуальні характеристики (рисунок папілярного візерунка, райдужна оболонка ока й т.д.).

Біопараметричні системи доступу є досить зручними й дружелюбними для користувачів. У відмінності від паролів і носіїв інформації для систем контролю доступу (ключів, карт, токенів), які можуть бути загублені, украдені, скопійовані, вони засновані на біопараметричних параметрах (відбитки пальців, форма руки, особи, рисунок райдужної оболонки ока,...), які завжди з нами й проблема їхньої схоронності, як правило, вирішується автоматично. Втратити їх набагато складніше. У ряді систем, що ще не одержали масового поширення, досягнута висока якість "біопараметричних ключів/замків". Треба думати, що через якийсь час вони проникнуть у сферу побуту.

Біопараметричні технології дуже швидко розвиваються останім часом, і ринок цих пристроїв уже зараз становить біля одного мільярда доларів. За прогнозами International Biometric Group через чотири роки цей ринок складе більше 4 млрд. доларів. Дотепер всі прогнози цієї організації трохи перевиконувалися реальним розвитком ринку. У світі працює більше 400 компаній, які пов'язані з розробками й реалізацією біопараметричної продукції.

Застосування біопараметричних систем

У цивільному й поліцейському застосуванні біопараметрики (у частині дактилоскопії) є певні розходження. Так як криміналістам треба ідентифікувати особистість часто по невеликих частинах відбитків пальців, вони повинні зберігати у своїх базах повні відбитки всіх пальців. Очевидно, що джерела інформації й "споживачі" тут не прагнуть пред'являти свої відбитки пальців.

У випадку ж застосування біопараметрики в системах контролю доступу, навпроти, кожна людина намагається якнайкраще й вірніше пред'явити свій відбиток пальця, щоб його швидше й краще розпізнали. У силу цього дуже часто обмежуються зіставленням щодо невеликої центральної області відбитка пальця, розміром іноді до 10х10 мм. У даний момент у рамках роботи комітету SC37 JTC1 ISO-IEC саме обговорюється питання про стандартизацію його розміру.

При прийнятті цього рішення варто пам'ятати, що чим менше ділянка пальця, по якому виробляється ідентифікація, тим менш точна ця ідентифікація. При дуже маленьких ділянках, скажемо 6х6 мм, імовірність помилки велика й досить швидко убуває в міру збільшення розміру скануємої ділянки відбитка пальців. Після досягнення розмірів 12х18 мм із подальшим збільшенням розмірів імовірність помилки зменшується вже значно слабкіше, а після того, як починають використовувати зону сканування 20х25 мм, їхня зміна не істотна.

Ще одна важлива відмінність поліцейських систем від систем контролю доступу полягає у важливості такого параметра, як швидкодія алгоритму розпізнавання. У криміналістиці треба пізнати пропонований відбиток, зрівнявши його з відбитками всієї або обґрунтовано обмеженої бази даних. Тут швидкодія дуже важлива.

Всі розвинені країни готуються до впровадження біопараметричних паспортів, і Україна не є виключенням. Активно йде робота з вироблення біопараметричних стандартів для забезпечення реальної роботи з біопараметричними паспортами на всій земній кулі.

Практично все для технічного рішення цього питання вже є. Більше складним є рішення соціологічних, психологічних і юридичних питань. Із цих питань створена спеціальна група вже згаданого міжнародного комітету зі стандартизації в області біопараметрики. Найімовірніше біопараметричні паспорти будуть містити в собі інформацію про відбитки пальців, візерунок райдужної оболонки ока, формі особи.

Огляд біопараметричних параметрів

Що варто розуміти під біопараметричними параметрами. У цей час біопараметричним глосарієм у рамках проекту стандарту ISO/IEC займається одна з робочих груп спеціального біопараметричного комітету SC37, що входить у найбільший міжнародний комітет зі стандартизації в області інформаційних технологій – JTC1.

Принцип роботи пристроїв для зчитування біопараметричних даних

Прилад зчитує біопараметричні дані, проводить необхідні виміри й зводить їхні результати до унікального значення (це хеш). Приміром, пристрій зчитування відбитків пальців може розрахувати відношення між гребінцем і западиною папілярного візерунка на пальці й перетворити його в значення хеш-функції. Втім, тут варто мати на увазі одну обставину. Відносно дешеві пристрої для зчитування біопараметричних даних можуть частіше робити помилки типу I і II, чим більш дорогі пристрої. Помилка типу I – це помилка виключення (скажемо, відбиток пальця користувача або інші його біопараметричні дані помилково зізнаються недійсними). Помилка типу II, більш небезпечна, – це помилкове розпізнавання (ситуація, коли відбиток пальця або інші біопараметричні дані іншої людини помилково зізнаються даними користувача). Внаслідок недостатньої надійності біопараметричної технології виробники багатьох пристроїв зчитування біопараметричних даних не рекомендують застосовувати ці пристрої для керування доступом до корпоративних мереж або важливої фінансовій інформації. Засіб для зчитування біопараметричних даних по своїй природі однофакторний пристрій ідентифікації; він базується на тому, що біопараметричні дані кожної людини унікальні. Деякі системи біопараметричної автентифікації вимагають введення користувачем персонального ідентифікаційного номера з метою скорочення числа помилок типу II.

По великому рахунку призначення пристроїв зчитування біопараметричних даних полягає в тому, щоб звести до мінімуму обсяг облікових даних, які повинні тримати в пам'яті користувачі, а не в тому, щоб взагалі усунути необхідність у використані облікових даних.

3.2 Розробка структурної схеми

Система багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI 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 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.

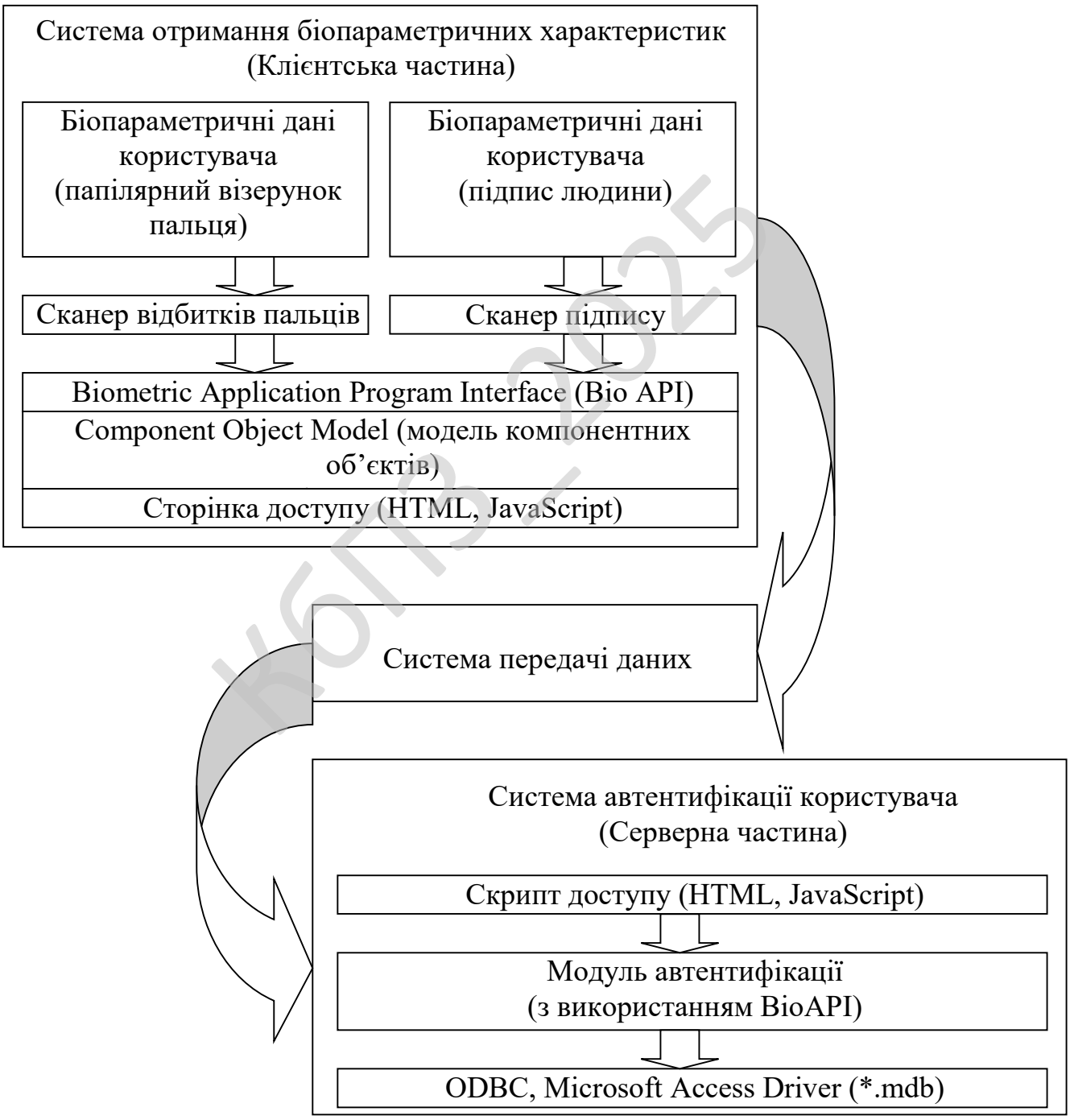


Рисунок 3.1 – Структурна схема роботи системи

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.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно. 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.

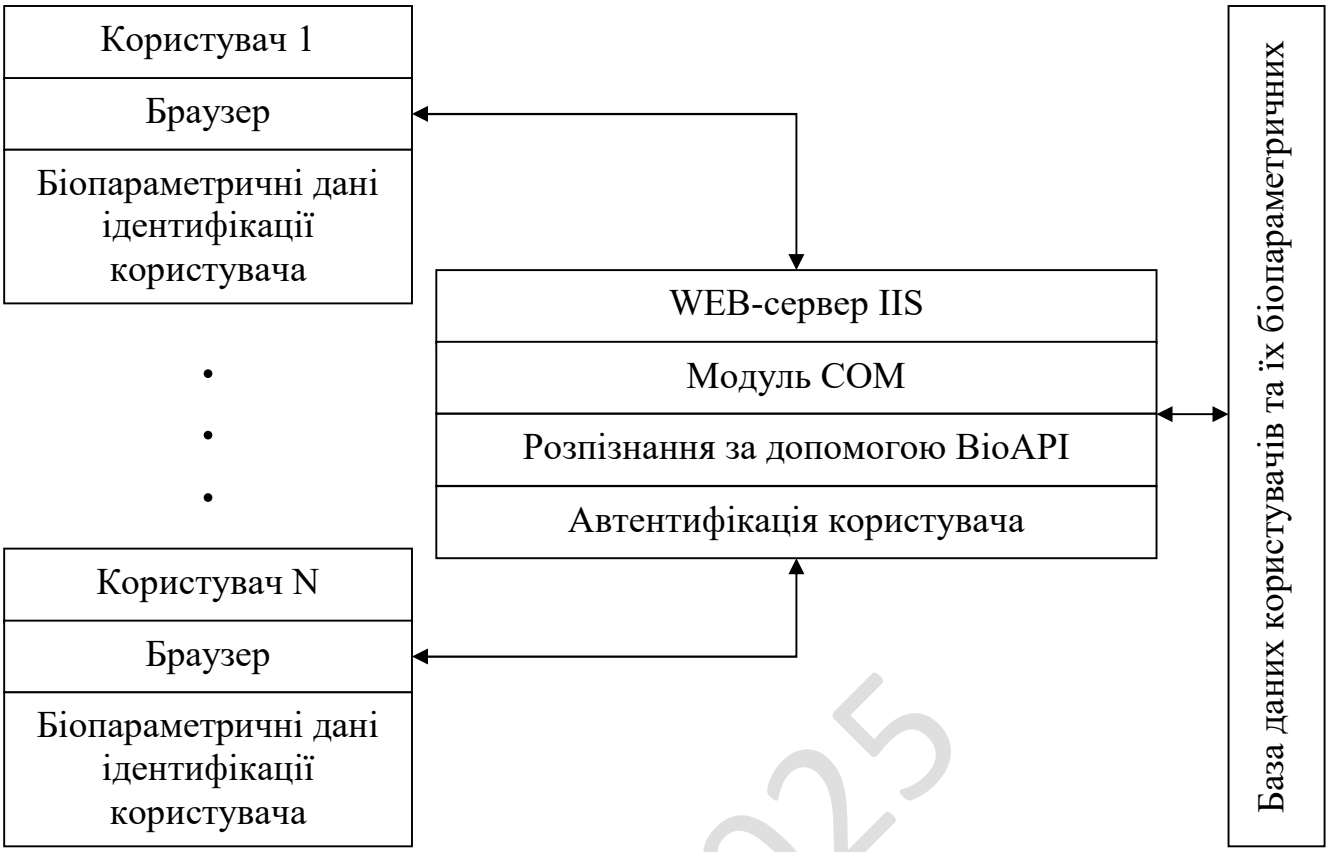


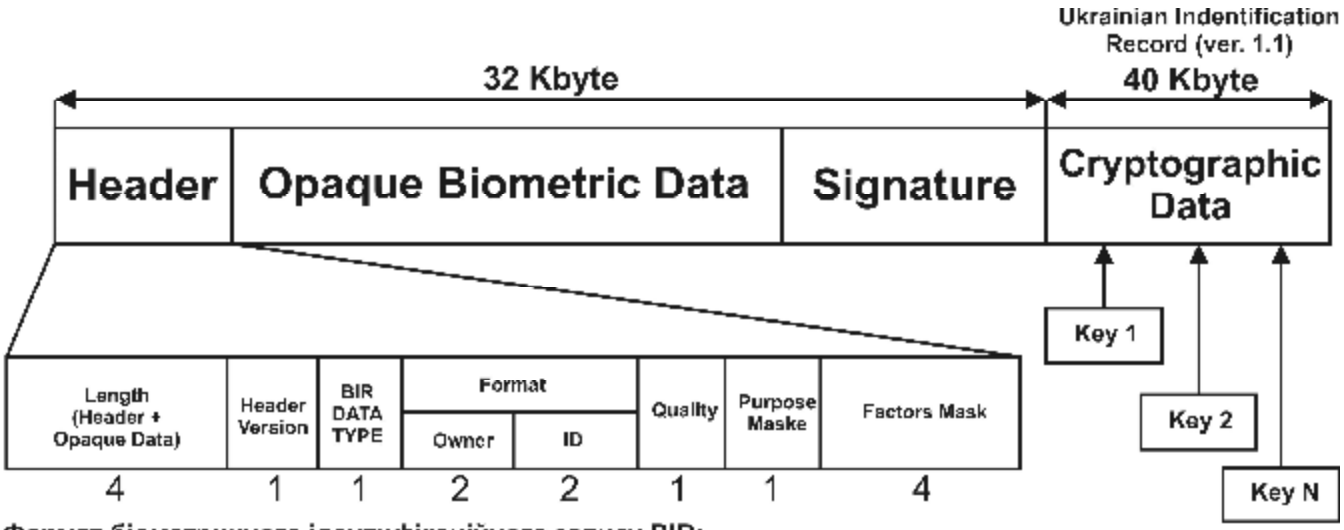
Рисунок 3.2 – Функціональна схема роботи системи

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.

Структура формату передачі даних

Розглянемо формат передачі даних, які складаються з біометричного ідентифікаційного запису й українського криптографічного додавання, зображених на рисунку 3.3.



Формат біометричного ідентифікаційного запису BIR:

Length - довжина заголовка й біометричних даних;
Header Version - версія заголовка BIR;
BIR Data Type - тип даних BIR;
Format (Owner/ID) - опис формату біометричних даних;
Quality - якість (необхідно для деяких типів біометричних даних);
Purpose Mask - ціль: верифікація, ідентифікатор по імені користувача, ідентифікація, реєстрація для ідентифікації, аудит)
Factors Mask - використовуваний метод біометрії;
Opaque Biometric Data - біометричні дані;
Signature - поле електронно-цифрового підпису;
Record Data Type - тип біометричних даних;
Creation Date - дата створення BIR, час і день одержання біометричних даних;
Creator - ідентифікатор творця BIR.

Рисунок 3.3 – 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.

Простота мови проектування та маніпулювання даними, зручність спілкування користувача з системою до мінімуму вивчення цієї програми. Користувач програми – це людина, яка повинна володіти азами програмування. При написані програми я намагалася, щоб програма відповідала наступним параметрам:

– Швидкодія. Програма працює постійно з великою кількістю кінцевих абонентів (селективний зв'язок).

– Захист. Забезпечити надійний захищений канал зв`язку.

– Відсутність проблеми дороговизни сучасних персональних комп'ютерів. – Система, що написана може встановлюватись на будь-якому персональному комп'ютері – використовувати відносно швидкі алгоритми захисту зв'язку.

– Можливість зручно і швидко формувати приклади і теорію для користувача.

– Можливість звертання до системних ресурсів. Користувача системи цікавить її інформаційний та сенсовий зміст. Подробиці організації фізичного зберігання даних його не цікавлять. Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.4. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Інтерфейс ПЗ.

– Моніторинг вхідних біометричних даних.

– Бібліотека обробки відбитків пальців.

– Бібліотека обробки підписів користувачів.

– Сканування біометричних даних.

– Обчислення матриці Гессе.

– Створення дескрипторів точок.

– Формування кіл, які показують місцезнаходження та масштаб точок.

– Формування ліній, що показують напрям градієнта яскравості.

– Порівняння зображення з наборами ключових точок БД.

– Формування біометричного пакету даних.

– Багатофакторна ідентифікація з використанням інтерфейсу BioAPI.

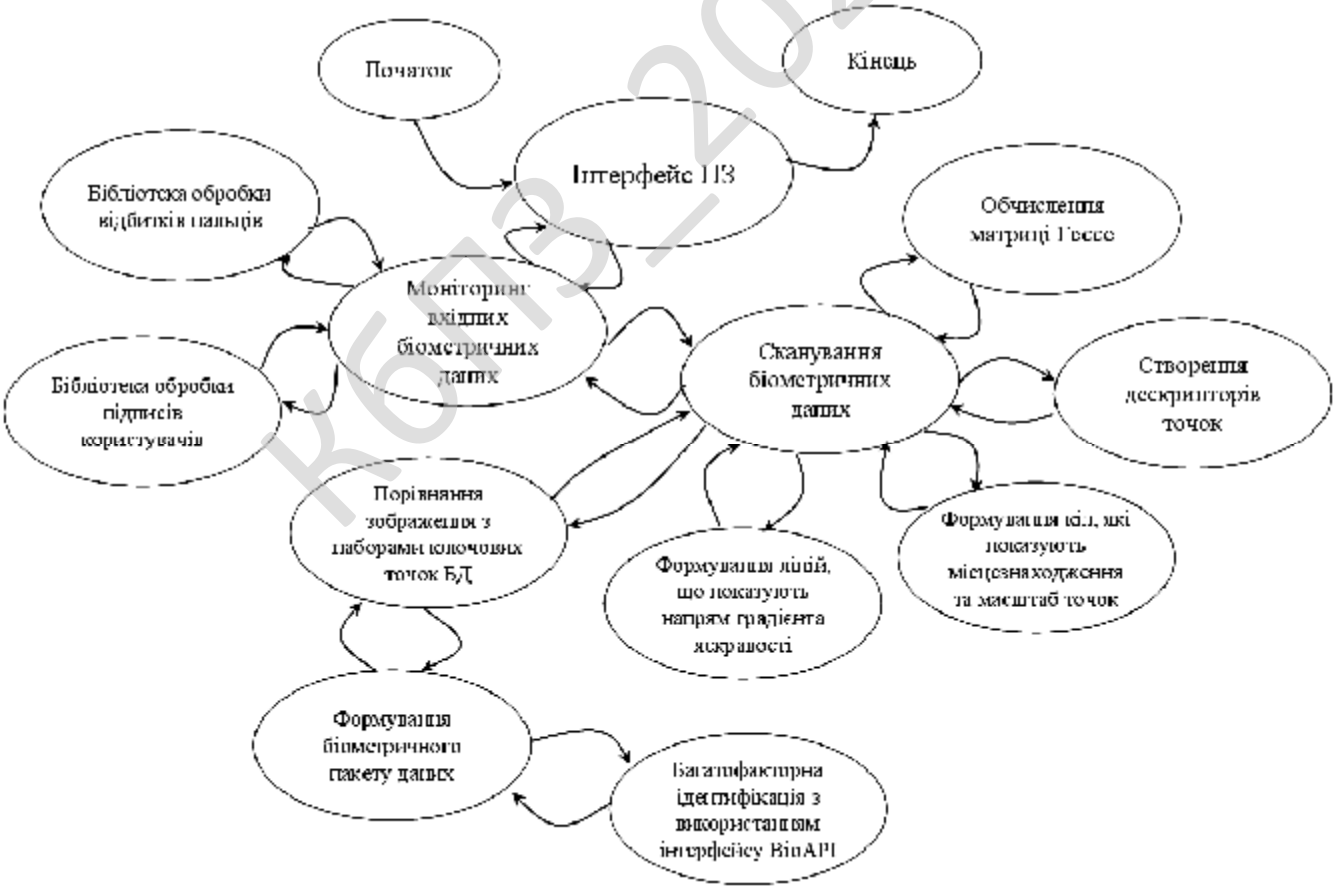


Рисунок 3.4 – Діаграма взаємодії процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. На рисунку 4.2 зображено роботу підпрограми. З якої видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

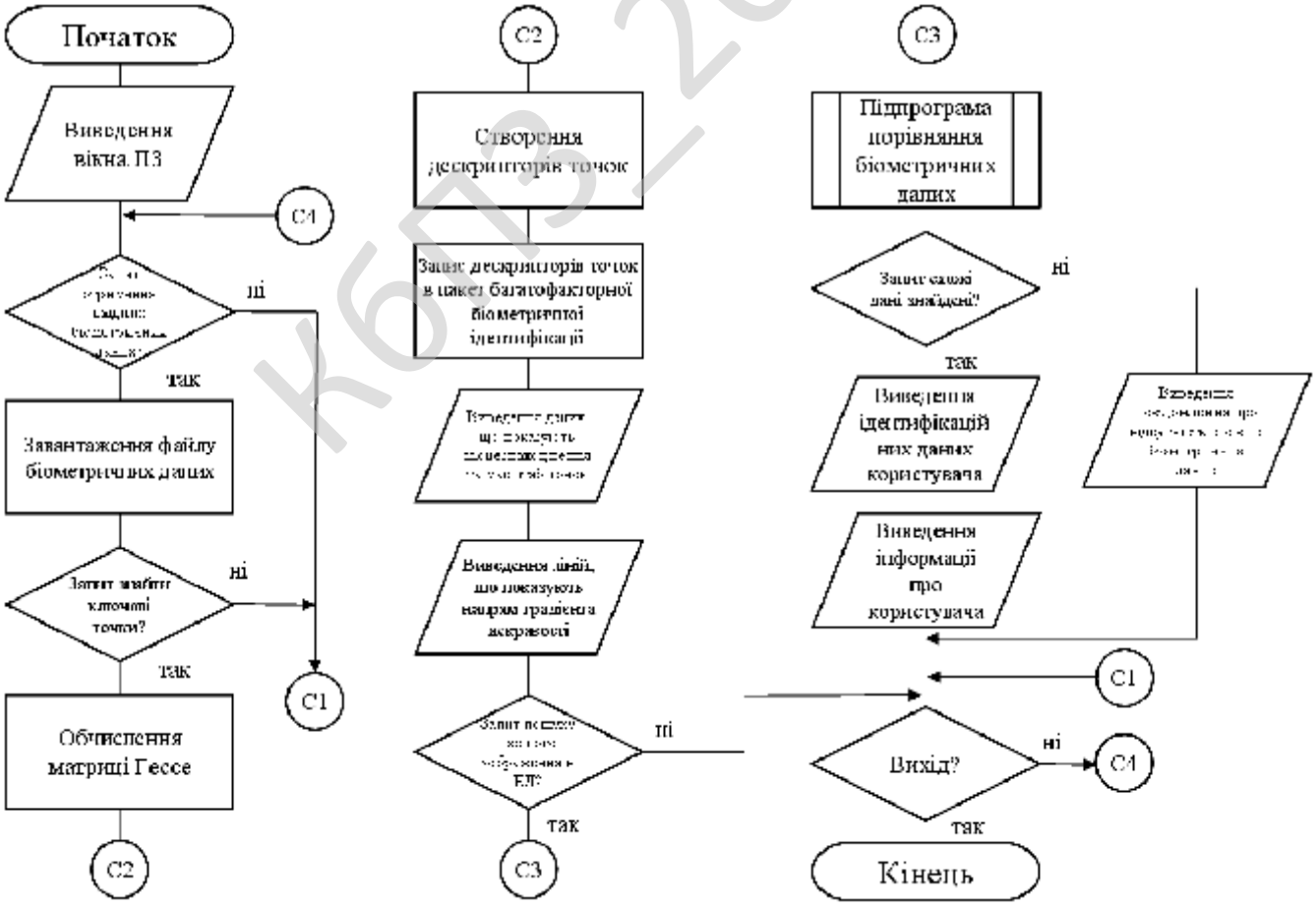


Рисунок 4.1 – Блок схема основної програми

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем я враховував, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю біометричної ідентифікації з використанням інтерфейсу BioAPI.

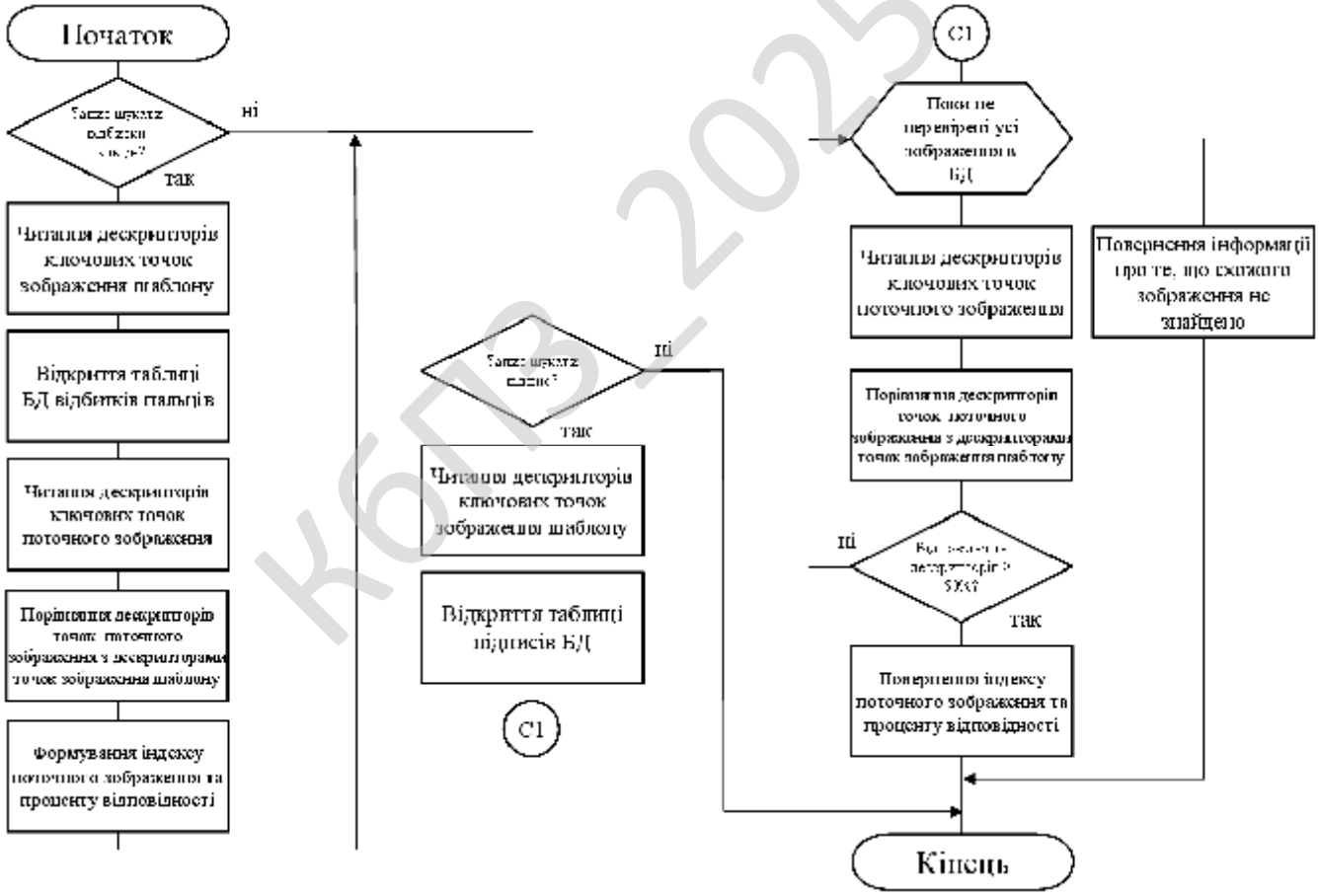


Рисунок 4.2 – Блок схема підпрограми

Опис алгоритмів функціонування системи

При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

Розглянемо заголовочний файл для шару відповідностей.

```
/**
Програма розпізнання багатофакторної біометричної ідентифікації з використанням
інтерфейсу BioAPI
**/

//#define RL_DEBUG
// не треба коментувати для тесту шару відповідностей

class ResponseLayer
{
public:
  int width, height, step, filter;
  float *responses;
  unsigned char *laplacian;
// об'ява змінних

ResponseLayer(int width, int height, int step, int filter)
  {
// обробка даних
    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;
  }

// система
  inline unsigned char getLaplacian(unsigned int row, unsigned int column)
  {
    return laplacian[row * width + column];
  }
// виклик
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];
  }

// отримання відповіді
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
};
```

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

```
//! 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;
}
// 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];
  }
}
//! 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 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 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;
}
```

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму UMAC (код автентифікації повідомлення на основі універсального гешування) – один з видів коду автентичності повідомлень (MAC).

Швидка «універсальна» функція використовується, для того, щоб гешувати вхідне повідомлення M у короткий рядок. До цього рядка потім застосовується функція XOR із псевдовипадковим значенням, у результаті чого ми одержуємо тег UMAC:

$$Tag=H_{K1}(M)\oplus F_{K2}(Nonce)$$

де K1 і K2 – секретні випадкові ключі, які мають одержувач і відправник.

Звідси видне, що безпека UMAC залежить від того, яким випадковим способом відправник і одержувач вибрали таємну геш-функцію й

псевдовипадкову послідовність. При цьому значення Nonce міняється кожний такт. Через використання Nonce, приймач і передавач повинні знати час відправлення повідомлення й принцип створення значення Nonce. Замість цього можна використовувати в якості Nonce будь-яке інше неповторюване значення, наприклад порядковий номер повідомлення. При цьому даний номер не зобов'язано бути секретним, головне щоб він не повторювався.

UMAC розрахований на використання 32-х, 64-х, 92-х, і 128-бітових тегів, залежно від необхідного рівня безпеки. UMAC звичайно використовується разом з алгоритмом шифрування AES.

Функція створення ключа й псевдовипадкової послідовності

Створення псевдовипадкових байтів необхідно для роботи UHASH і при створенні тегів

Вибір блокового шифру

Для своєї роботи UMAC використовує блоковий шифр, вибір якого визначають наступні константи:

– BLOCLEN – довжина, у байтах, блоку з яким працює блоковий шифр.

– KEYLEN – довжина, у байтах, ключа блокового шифру.

При цьому використовується функція

– ENCIPHER(K,P) – зашифрувати рядок P з BLOCLEN байтів, використовуючи ключ K.

Приклад: якщо використовується AES з 16-байтним ключем, то BLOCLEN буде рівним 16(тому що AES працює з 16-байтними блоками).

KDF – функція створення ключа

Ця функція генерує послідовність псевдовипадкових байтів, використовуваних для ключових геш-функцій.

Вхід:

– K – рядок довжиною KEYLEN байт. / / Ключ блокового шифру.

– Index – ненегативне ціле число менше, чим 2 ^ 64.

– Numbytes – ненегативне ціле число менше, чим 2 ^ 64.

Вихід:

– Y – рядок довжини numbytes байт.

PDF: функція створення псевдовипадкового числа

Ця функція ухвалює ключ і даний час і повертає псевдовипадкове число для використання його в тегу покоління. За допомогою цієї функції можуть бути отримані числа довжиною 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– Nonce – рядок довжиною від 1 до BLOCKLEN байт.

– Taglen – ціле число 4, 8, 12 або 16.

Вихід:

– Y – послідовність байтів довжини taglen.

Генерація UMAC-тегів

Генерація UMAC-тегів відбувається за допомогою UHASH функції при використанні Nonce значенні й отриманої до цього рядка. Їхня довжина може бути 4, 8, 12 або 16 байт.

Вхід:

– K – рядок довжиною KEYLEN байт.

– M – рядок довжиною менше 267 біт.

– Nonce – випадкове число від 1 до BLOCKLEN байт.

– Taglen – ціле 4, 8, 12 або 16.

Вихід:

– Тег, послідовність байтів довжиною taglen.

Алгоритм обчислення тегів:

Hashedmassage = UHASH(K, M, Taglen)

Pad = PDF(K, nonce, Taglen)

Tag = Pad xor Hashedmassage

UMAC-32 UMAC-64 UMAC-96 UMAC-128

Дані позначення містять у своїй назві певне значення довжини тегу:

– UMAC-32 (K, M, Nonce) = UMAC (K, M, Nonce, 4).

– UMAC-64 (K, M, Nonce) = UMAC (K, M, Nonce, 8).

– UMAC-96 (K, M, Nonce) = UMAC (K, M, Nonce, 12).

– UMAC-128 (K, M, Nonce) = UMAC (K, M, Nonce, 16).

Універсальна функція гешування(UHASH)

UHASH – універсальна функція гешування, серцевина алгоритму UMAC. UHASH – функція працює в три етапи. Спочатку до вхідного повідомлення застосовується L1-HASH, потім до цього результату застосовується L2-HASH і, нарешті, до результату застосовується L3-HASH . Якщо при цьому довжина вхідного повідомлення не більш 1024 біт, то L2-HASH не використовується. Тому що функція L3-hash повертає тільки слово довжини 4 байта, те якщо потрібно одержати геш довжини більше 4 байт, здійснюється кілька ітерацій даної трирівневої схеми.

Універсальна функція

Нехай функція гешування вибирається із класу геш-функцій H, які відображають повідомлення в D, набір усіляких образів повідомлення. Цей клас називається універсальним, якщо для яких-небудь окремих пар повідомлень, існує на безлічі H/D функцій, функція, яка відображає їх в елемент D. Зміст цієї функції в тому, що якщо третя сторона прагне замінити одне повідомлення іншим, але при цьому вважає, що геш-функція була обрана абсолютно випадково, те ймовірність не виявлення підміни стороною, що ухвалює, прагне до 1/D.

L1-hash – перший етап

L1-hash розбиває повідомлення на шматки з 1024 байт і до кожного шматка застосовує алгоритм гешування називаний NH. Вихідний результат алгоритму NH в 128 раз менше вхідного.

L2-hash – другий етап

L2-hash працює з виходом L1-hash, використовує поліноміальний алгоритм POLY. Другий етап гешування використовується, тільки якщо довжина вхідного повідомлення більше 16 мегабайт. Використання алгоритму POLY потрібно для того, щоб уникнути тимчасову атаку. На виході з алгоритму POLY виходить 16 байтне число.

L3-hash – третій етап

Цей етап потрібно для того щоб з вихідних 16 байтів алгоритму L2-hash одержати 4-байтне значення.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні розділи:

– Поле сканованого зображення – відбиток пальцю.

– Функціональні кнопки: Відкрити файл; Знайти у базі даних; Обчислити ключові точки; Про програму.

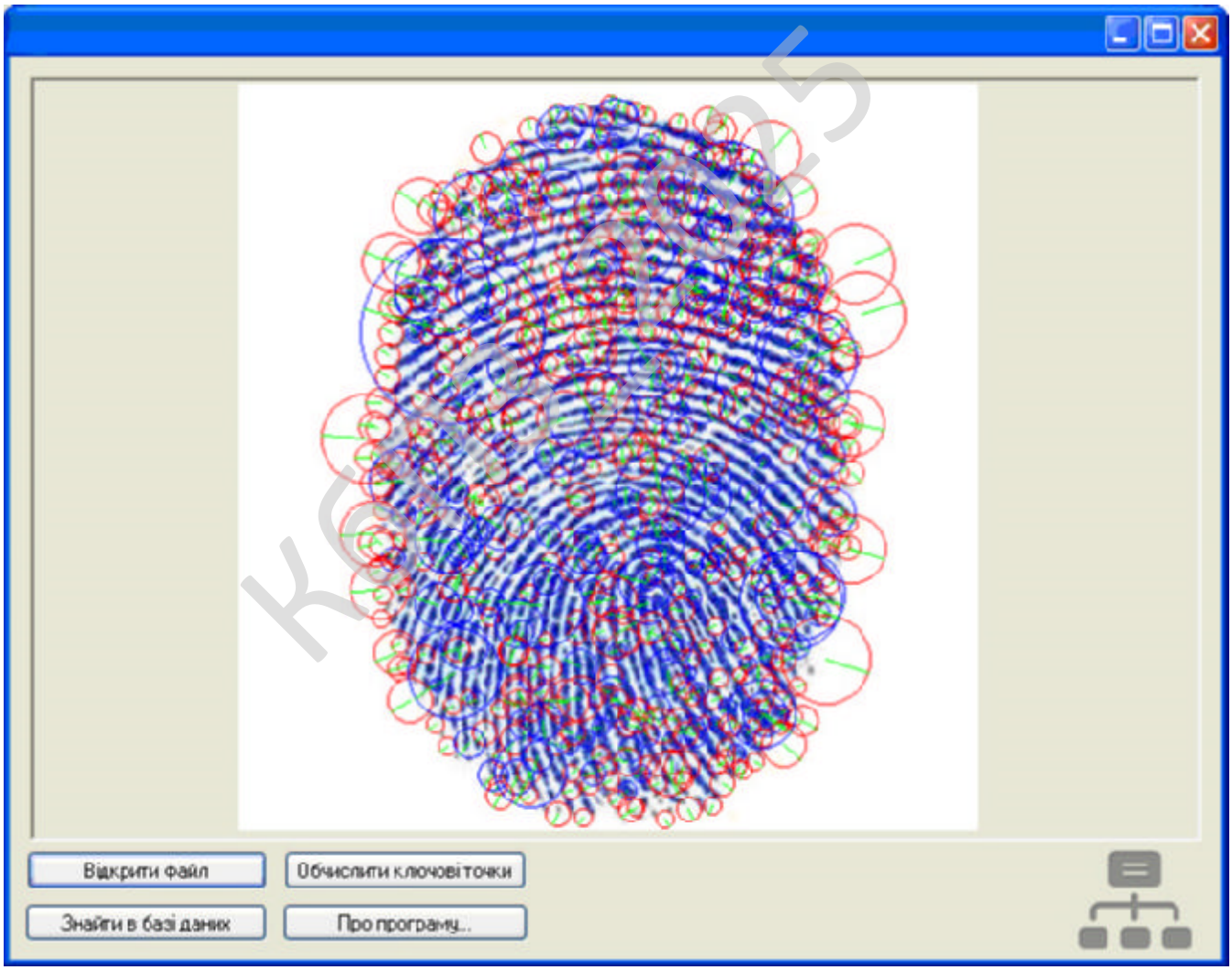


Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

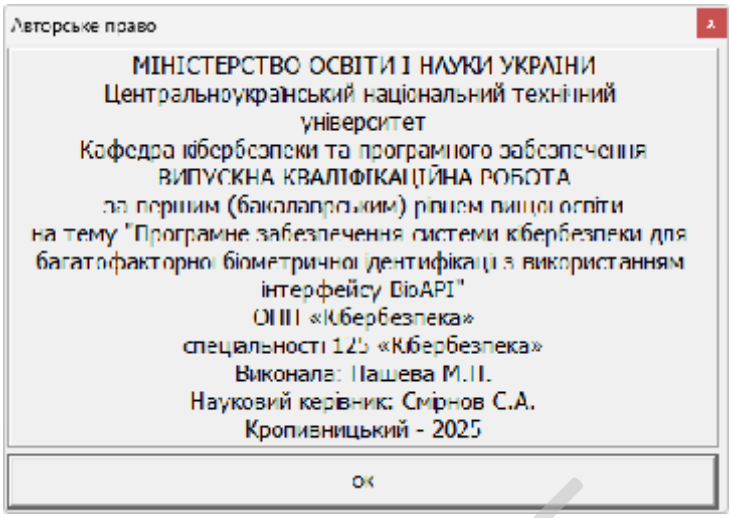


Рисунок 5.2 – Авторське право

Обрано умови розповсюдження – Shareware. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму. В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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 багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

– Досліджена система для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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 багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI. 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 Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм UMAC.

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. Josh Armitage. Cloud Native Security Cookbook. O'Reilly Media. 2022. 516 p.

2. Massimo Bertaccini. Cryptography Algorithms. Packt Publishing. 2022. 358 p.

3. Alyssa Miller. Cybersecurity Career Guide. Manning Publications. 2022. 368 p.

4. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

5. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

6. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

7. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

8. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

9. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

10. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

11. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

12. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

13. Kuznetsov O., Frontoni E., Kuznetsova Ye., Smirnov O., Chevardin V. «Achieving Enhanced Security in Biometric Authentication: A Rigorous Analysis of Code-Based Fuzzy Extractor». *CEUR Workshop Proceedings,* Volume 3624, 2023, pp. 330-339.

14. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

15. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

16. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

17. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

18. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

19. Kuznetsov, A., Oleshko, I., Chernov, K., Bagmut, M., Smirnova, T. «Biometric authentication using convolutional neural networks». Lecture Notes in Networks and Systems. Volume 152, 2021, Pages 85-98.

20. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

21. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE*

International Conference on Advanced Information and Communication Technologies (AICT) - 2021, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

22. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

23. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

24. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

25. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

26. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

27. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

28. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

29. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

30. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

31. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

32. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

33. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

34. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

35. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

36. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

37. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

38. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

39. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

40. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

41. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

42. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

43. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

44. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

45. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

46. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

47. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

48. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

49. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

50. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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. № 57-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI.

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 багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI;

– 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, 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 – 58 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 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 3.06.2025 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

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки для багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2025 року

responselayer.h – заголовочний файл для шару відповідностей

```
/*************************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*************************************************************/


//#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

```
/*********************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*********************************************************/

#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)*/);
}

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

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<< "Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI знайшла: " <<
ipts.size() << " особливі точки" << std::endl;
  std::cout<< "Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI 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("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI",
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("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI", img);

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

  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програмне забезпечення системи кібербезпеки для
багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI" );
  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("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI",
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("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI", img);

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

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програмне забезпечення системи кібербезпеки для
багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI" );
  return 0;
}


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


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

  // Створюємо вікно
  cvNamedWindow("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI",
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("Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI", img);

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

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програмне забезпечення системи кібербезпеки для
багатофакторної біометричної ідентифікації з використанням інтерфейсу BioAPI" );
  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

```
/*************************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*************************************************************/

#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 гаусціані 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 гаусціані 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

```
/*************************************************************
* --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI --- *
*************************************************************/

#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 гаусciані 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

```
/*********************************************************
* --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI --- *
*
* *
*********************************************************/

#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

```
/*********************************************************
* --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI --- *
*********************************************************/

#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

```
/*********************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*
*
*********************************************************/

#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

```
/*********************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*
*                                                        *
**********************************************************/

#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

```
/*********************************************************
* --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI --- *
* *
*********************************************************/

#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

```
/*************************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*
*
*                                                            *
*************************************************************/

#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

```
/*************************************************************
*  --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI ---
*                                                            *
*************************************************************/

#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

```
/*************************************************************
* --- Програмне забезпечення системи кібербезпеки для багатофакторної
біометричної ідентифікації з використанням інтерфейсу BioAPI --- *
* *
*************************************************************/

#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
```