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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Кравченко В.О.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2024 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 2024 року

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

Кравченку Владиславу Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача*

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

затверджені наказом вищого навчального закладу № 135-02 від 01.04.2024 року

3. Строк подання студентом роботи до захисту *23.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача*

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 2024 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	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

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

Завдання прийнято до виконання
« 17 » січня 2024 р.

Підпис здобувача
_____ Кравченко В.О.
(прізвище та ініціали)

АНОТАЦІЯ

Кравченко В.О. Програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

Метою розробки є програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

Результат роботи – програмна реалізація системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

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, ідентифікація, біометрія

ABSTRACT

Kravchenko V.O. Cyber security system software for identification based on recognition of the user's palm print. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for a cyber security system for identification based on the recognition of the user's palm print.

The goal of the development is the software of the cyber security system for identification based on the recognition of the user's palm print.

The result of the work is a software implementation of a cyber security system for identification based on recognition of the user's palm print.

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

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

The program can be used on a PC with Windows 10/11 OS.

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

Keywords: cyber security, identification, biometrics

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

MISTY1 – 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. Біометрична безпека сьогодні подолала обмеження попередніх обчислювальних днів, і люди в усьому світі вважають за краще використовувати системи біометричного розпізнавання як альтернативу звичайним методам автентифікації на основі пароля. Правоохоронні органи, прикордонний контроль, фінансові служби та різноманітні споживчі інтелектуальні пристрої вибрали біометричне розпізнавання, оскільки воно усуває необхідність запам’ятовувати паролі, є набагато точнішим у підтвердженні особи людини та діє як рівень захисту від несанкціонованого доступу. доступу, тим самим задовольняючи потреби безпеки в поточному контексті. Останні досягнення в біометричному розпізнаванні значною мірою завдячують розробкам у сфері машинного навчання, зокрема його підмножині глибокого навчання. Через потребу в ідентифікації серед мільйонів наборів даних має сенс використовувати методи машинного навчання в таких амбітних завданнях, а універсальність методів глибокого навчання для достовірної ідентифікації даних робить їх видатним методом серед інших традиційних алгоритмів класифікації. Крім того, алгоритми машинного навчання поступово використовуються в області виявлення живості та інших методологій глибокого навчання, які захищають бази даних шаблонів.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

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

– Огляд існуючих систем для ідентифікації на основі розпізнавання відбитку долоні користувача.

– Дослідження системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

– Програмна реалізація системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для ідентифікації на основі розпізнавання відбитку долоні користувача.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача , 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

У світовій біометрії домінують технології ідентифікації по відбитках пальців (на них доводиться більше 60% загального обсягу галузевого ринку). По оцінках Frost & Sullivan, до 2025 р. обсяг світового ринку біометричних засобів, експлуатованих у кредитних організаціях, перевищить два мільярди доларів, і майже 90% продажів біометричних засобів захисту інформації у фінансовому секторі доводиться на частку рішень, що розпізнають користувачів по відбитках пальців.

Для банків особливо важливо те, що відмова від дій, підтверджених пред'явленням біометричних ідентифікаторів, неможлива. На відміну від карток, відбитки пальців не мають обмежень по числу «зчитувань», у біометричних системах можлива реєстрація резервних ідентифікаторів, і тут відбитки пальців знову виявляються спереду: у звичайної людини на руках 10 пальців і всього лише одна особа, по два ока й по дві руки.

Особливо перспективною є розвиток технологій ідентифікації по відбитках пальців, оскільки ці технології були й залишаються найпоширенішими й звичними для переважної більшості користувачів. Аналітики RNCOS також думають, що в найближчі 3 роки розшириться застосування технологій біометрії з метою забезпечення безпеки електронної комерції й дистанційних платежів.

1.2 Область застосування

Приємно відзначити, що банки України рухаються в руслі загальносвітових тенденцій. Наприклад, що діє в Латинській Америці Banco Azteca також сканує у своїх клієнтів відбитки пальців при доступі до рахунків, і його інформаційна система щодня здійснює не менш 200 тисяч порівнянь цих

ідентифікаторів. Перед напором біометричних технологій не встояли навіть кредитні організації фінансової Мекки – Швейцарії. Банк Pictet & Cie.

Біометричні технології в медичних установах

Лікувально-профілактичні установи перемикаються на технології біометрії, щоб забезпечити безпеку своїх пацієнтів, прискорити їхнє обслуговування, скоротити число помилок. Біометричні технології активно інтегруються в системи електронних медичних карт і використовуються для захисту персональних даних пацієнтів.

Так, приміром, медичний центр Святого Вінсента в американському штаті Індіана домігся підвищення рівня інформаційної безпеки й швидкості обслуговування пацієнтів, впровадивши у свою діяльність сканери відбитків пальців. Медичний центр Міністерства по справах ветеранів, розташований у Флориді, ідентифікує своїх пацієнтів по їхніх голосах, а клініка Urban Health Plan у Бронксі уже два роки експлуатує сканери райдужної оболонки око, і з їхньою допомогою швидко й безпомилково встановлюється особистість пацієнтів.

Ще одна медична організація у Флориді – Simply Healthcare Plans – використовує сканери райдужної оболонки для боротьби з незаконним використанням медичних страховок. Оскільки біометричні ідентифікатори унікальні для кожної людини, спроби пред'явити чужий поліс для одержання медичних послуг виявляються марними й відразу виявляються.

Біометричні банкомати: зручно й надійно

Якщо ще недавно біометрична ідентифікація тримачів банківських карт здавалася фантастикою, то зараз вона стала зовсім звичною для десятків і сотень тисяч користувачів у Бразилії й Індії, Польщі й Саудівській Аравії, Японії й Колумбії. А в Україні пілотні проекти по інтеграції біометрії в банкомати й POS-термінали реалізує навіть Ощадбанк: у декількох регіонах тримачі емітованих їм «соціальних карт» можуть оплачувати товари й послуги, скануючи відбитки пальців. Особливу позицію в даному питанні зайняли японські банки. Вони воліють ідентифікувати тримачів карт по малюнку вен, причому нежартівлива

боротьба розгорілася між двома технологіями. Одна з них припускає сканування малюнка вен на пальці, а інша зчитує цей малюнок з усією долоні. Втім, однією тільки Країною висхідного сонця експансія біометричних банкоматів даного типу не обмежилася: бразильський Bradesco Bank оснастив більше півтори тисяч банкоматів сканерами малюнка вен на долоні, а польський кооперативний банк BPS SA першим у Європі впровадив ідентифікацію тримачів карт по малюнку вен на пальці.

Облік робочого часу за допомогою біометричних технологій

Біометричні системи обліку робочого часу являють приклад одного з найбільш успішних застосувань біометричної технології. У міру спрощення інтеграції й використання біометричної технології багато корпорацій зважуються на її впровадження з метою обліку робочого часу співробітників[1]. Біометричні системи обліку робочого часу відіграють провідну роль на сучасних висококонкурентных ринках. Так як витрати на оплату праці в усе більшому ступені визначають рентабельність бізнесу, основною метою підприємців є можливість знизити втрати робочого часу. Однією з основних переваг біометричних систем обліку робочого часу є та, що вони дозволяють уникнути «приятельського шахрайства» (коли співробітники передають картки один одному й відзначають один одного). В інших системах «Приятельське шахрайство» – основна лазівка, що дозволяє обходити традиційні системи обліку робочого часу. У цей час за допомогою впровадження біометричних систем компанії змогли покласти кінець цим вивертам. Крім того, біометричні системи обліку робочого часу виконують і інші важливі функції. Біометрична система обліку робочого часу дозволяє точно визначати, який саме співробітник (або співробітниця) відзначається при приході або відході з робочого місця. За допомогою використання відбитків пальців (або інших біометричних характеристик). Робочий час кожного співробітника відбивається у звіті по обліку робочого часу. Ці дані можуть використовуватися для визначення фонду заробітної плати. Здійснюється точний облік робочого часу співробітника і його

понаднормової роботи й, виходячи їх цього, розраховується відповідна заробітна плата.

Біометричні системи контролю доступу

Перед тим, як перейти до обговорення біометричних систем контролю доступу, давайте спробуємо прояснити, що саме розуміється під «контролем доступу». Відносно забезпечення безпеки, особливо безпеки тієї або іншої території, цей термін в основному пов'язаний із двома аспектами:

– Обмеження доступу на підприємство або територію, що є чиєї-або власністю, що досягається за допомогою надання доступу користувачам, що мають на той дозвіл, і заборони доступу тим, хто його не має;

– Відстеження входу й виходу.

У цей час біометричне встаткування або системи контролю доступу являють собою не що інше як системи, що використовують ту або іншу біометричну характеристику для забезпечення двох вищезгаданих аспектів контролю доступу.

На ринку представлені різні типи біометричних систем контролю доступу, що використовують відбитки пальців, сканування сітківки ока, геометрію руки й т.д.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача, 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

Відбитки пальців для захисту папок з даними в Windows 10/11

Операційні системи Windows 10/11 одержали убудовану підтримку сканерів відбитків пальця. Незважаючи на те, що й у більше ранніх версіях ОС була можлива робота зі сканерами, Microsoft необхідно було здобувати програмне забезпечення в третіх осіб. В Windows 10/11 з'явилася можливість розпізнавати відбитки без додаткових програм.

Користувач зможе зв'язати відбиток зі своїм обліковим записом Microsoft, що дозволить здійснити вхід у систему, підтвердити покупку програм або авторизуватися в додатках, приклавши палець до сканера. Крім того, у такий спосіб можна буде захистити особисті папки.

В Microsoft повідомили, що підтримка сканерів відбитків стала можливою завдяки тісному співробітництву з декількома виробниками подібних пристроїв. У розробці також брало участь кілька компаній-партнерів, які займалися вбудовуванням сканерів у планшети, клавіатури, мишки, а також ноутбуки.

Лінія біометричних технологій Neurotechnology для Android

Відповідно до повідомлення співробітників Neurotechnology, компанія представила нову лінію біометричних технологій, призначених для роботи на мобільних пристроях. Серед релізів фахівці відзначили VeriFinger Embedded SDK (технологія сканування відбитків пальців), VeriLook Embedded SDK (технологія розпізнання осіб) і MegaMatcher Embedded SDK (комбінація обох біометричних систем). Крім того, у найближчому майбутньому в Neurotechnology мають намір реалізувати технологію розпізнання голосу й райдужної оболонки ока.

Рисунок 2.1 – Інтерфейс користувача VeriLook Embedded SDK

Відзначимо, що дані версії, позначені виробником як «Embedded», призначені для використання в малопотужних мобільних пристроях, таких як смартфони, планшети й портативні комп'ютери. Як уточнили експерти, VeriFinger Embedded і VeriLook Embedded здатні провести ідентифікацію по відбитку пальців або особі менш чим за 1 секунду за допомогою процесора з тактовою частотою 1 ГГц. У даний момент зазначені технології підтримують ряд популярних стандартів від сторонніх виробників, у тому числі сканери AuthenTec EikonTouch, Eikon II, TouchChip і BLUEFi Bluetooth сканер від Toplink Pacific і працюють на базі ОС Android. Разом з тим, представники Neurotechnology затверджують, що в наступних версіях буде реалізована підтримка операційних систем ARM Linux і WinCE.

EKEY

ekey для будинку – інноваційний комплект сканера відбитків пальців для будинку або дверей офісу. Система сполучає у собі простоту установки й інтуїтивно зрозуміле керування користувачами. Залежно від конкретної області використання, сканери випускаються в різних виконаннях.

Основні функції ekey для будинку:

– Пам'ять приладу до 99 різних відбитків пальців.

– Керування користувачами здійснюється безпосередньо на панелі керування (ПК не потрібно).

– ekey home – панель керування з 1 релейним виходом.

– ekey home 3 – панель керування з 3 релейними виходами. Можливість керування 3-мя пристроями (наприклад, вказівний палець відкриває вхідні двері, середній палець відкриває двері гаража, підмізинний палець управляє сигналізацією).

Накладний сканер

Для безпосередньої установки на стіну. Накладний сканер сполучає у собі простоту установки за доступною ціною. Доступний у сріблисто-металевому кольорі. Може бути встановлений на стіні в кілька простих кроків.

Пристрій підключається до контрольної панелі ekey home за допомогою гвинтових клем на звороті сканера. Обмін даними між обома компонентами відбувається в зашифрованому виді.

Сканер для зовнішньої установки

Для інтеграції в панелі домофонів, поштових скриньок і т.п., поза приміщеннями. Сканер оснащений лінійним сенсором, так, що замість прикладання пальця до поверхні, палець повинен бути проведений по сенсору. Таким чином, не залишається відбитків пальців, які могли б ще бути використані зловмисниками. Крім того, сканер оснащений інтелектуальним програмним забезпеченням: при кожній операції система продовжує самонавчатися. Зміни у звичках користувача будуть записуватися автоматично. Маючи компактні

розміри 50x50 мм, зовнішній сканер є ідеальним рішенням для установки в електричні коробки, домофоні панелі й поштові скриньки. Сполучимо по розмірах із провідними виробниками й доступний у білій, антрацитовій і алюмінієвій квітках. Він прекрасно вписується в існуючий асортименти продукції.

Пристрій підключається до контрольної панелі ekey home за допомогою гвинтових клем на звороті сканера. Обмін даними між обома компонентами відбувається в зашифрованому виді.

Сканер Integra

Для інтеграції у дверні полотна й дверні рами. Завдяки витонченому й компактному дизайну, сканер Integra є ідеальним рішенням для інтеграції у фасадні двері, панелі домофонів і поштові скриньки. Стильний і елегантний прилад доступний у нержавіючій сталі, білому або золотом кольорах. За допомогою монтажних скоб на зовнішній стороні корпуса, сканер може бути встановлений у порожнину стіни, а для спрощення установки в суцільні стіни – комплектується монтажним комплектом.

Сканер оснащений лінійним сенсором, так, що замість прикладання пальця до поверхні, палець повинен бути проведений по сенсору. Таким чином, не залишається відбитків пальців, які могли б ще бути використані зловмисниками. Крім того, сканер оснащений інтелектуальним програмним забезпеченням: при кожній операції система продовжує самонавчатися. Зміни у звичках користувача будуть записуватися автоматично.

На звороті сканера розташований RJ-роз'єм, через який пристрій підключається до контрольної панелі ekey home. Використовуючи існуючі кабелі, практично неможливо підключити систему неправильно. Обмін даними між обома компонентами відбувається в зашифрованому виді.

Сканер FSB

Дверна ручка з убудованим сканером. У цьому варіанті сканера, технологія ekey майже непомітна у дверній ручці з нержавіючої сталі. Тільки ненав'язливий світлодіод на лицьовій стороні вказує на передові технології.

Сканер на звороті дверної ручки оснащений лінійним датчиком, так, що замість прикладання пальця до поверхні, палець повинен бути проведений по сенсорі. Таким чином, не залишається відбитків пальців, які могли б ще бути використані зловмисниками. Крім того, сканер оснащений інтелектуальним програмним забезпеченням: при кожній операції система продовжує самонавчатися. Зміни у звичках користувача будуть записуватися автоматично.

Сканер FSB був розроблений для установки в процесі виробництва дверей. Обмін даними між обома компонентами відбувається в зашифрованому виді.

Панелі керування ekey home

Для роботи системи ekey home потрібна підходяща панель керування. Залежно від типу установки, вона доступна в різних версіях.

Завдяки компактному розміру, панель керування DRM ідеально підходить для установки на DIN-рейку в електричну шафу (розподільний щит) або на стіну. Доступні два варіанти:

– ekey home1 CP DRM оснащений 1 сухим релейний виходом і 1 цифровим входом;

– ekey home2 CP DRM оснащений 2 сухими релейними виходами й 2 цифровими входами. Для підключення до сканера панель керування має гвинтові клеми.

Настінна панель може бути встановлена на DIN-рейку, на стіні або в електричній шафі (розподільному щиті). Доступні два варіанти:

– ekey home1 CP WM оснащений 1 сухим релейний виходом;

– ekey home2 CP WM оснащений 3 сухими релейними виходами. Для підключення до сканера панель керування має пружинні затискачі.

Панель керування Integra призначена для установки в процесі виробництва дверей. Вона встановлюється або безпосередньо у дверне полотно, або у дверну раму. Доступні два варіанти:

– ekey home1 CP IN оснащений 1 сухим релейний виходом;

– ekey home2 CP IN оснащена 2 сухими релейними виходами. Для підключення до сканера панель керування має спеціальні роз'єм, використовуючи які практично неможливо підключити систему неправильно.

Релейні виходи використовуються для керування електромеханічними замками, дверними замками, двигунами гаражних воріт, охоронною сигналізацією й т.д.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки 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 застосунків 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 застосунків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки застосунків Windows дозволяють автоматизувати процес створення застосунка. Для цього

використовуються генератори застосунків. Програміст відповідає на питання генератора застосунків і визначає властивості застосунка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор застосунків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої застосунки. Подібні засоби автоматизованого створення застосунків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики застосунка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні застосунки, а також застосунки, 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 застосунка програмістові прийдеться розробляти самостійно. Вихідний текст застосунка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон застосунка – це вже половина всієї роботи. Вихідні тексти застосунків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти застосунків 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 застосунки 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-застосунки 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-застосунків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

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

Біометричний контроль доступу – принцип роботи

Будь-яка біометрична система контролю доступу включає пристрій контролю доступу – рідер або сканер. Це пристрій, що зчитує вихідні дані у вигляді відбитка пальців або інформації, отриманої в результаті сканування сітківки ока, і т.д. Потім ця інформація аналізується й рівняється з особистою інформацією людини, записаної раніше. Якщо дані збігаються, відбувається автентифікація людини. Якщо автентифікований користувач має дозвіл на знаходження в даному приміщенні в цей період часу, пристрій подає певний сигнал і відкриває електронний замок.

Найкращим рішенням для невеликого підприємства є система контролю доступу, що використовує ідентифікацію по відбитках пальців.

Біометричні годинники – перспективне рішення для обліку робочого часу

Біометричний годинник улаштовані так само, як і біометричні системи контролю робочого часу. Тобто фактично та сама система називається по-різному. Термін «біометричні годинники» звичайно використовують люди, що прагнуть придбати системи, аналогічні табельним годинникам. Біометричні системи обліку робочого часу звичайно використовують упізнавання по відбитках пальців або іноді за формою долоні для обліку відходу й приходів співробітників. На підставі отриманої інформації розраховується заробітна плата.

Біометричні дверні замки

Поява біометричних дверних замків знаменує прихід біометричних технологій у повсякденне життя. У цей час біометричні замки доступні для всіх, а це означає, що біометричні пристрої можуть бути віднесені до категорії

побутових приладів. Біометричні дверні ручки в цей час усе більше широко використовуються в особняках, квартирних будинках, офісах і навіть у серверних кімнатах.

При використанні біометричних дверних замків ключем є відбитки пальців. Замикаючі механізми, що відкриваються звичайним ключем, заміняються зчитуючимим відбитки пальців сенсором, що може визначати, є чи дозвіл на вхід у того, хто намагається відкрити двері.

Біометричні дверні замки дають можливість забезпечити безпеку житлового будинку з використанням новітньої технології й усувають неприємні наслідки втрати або передачі ключів. Біометричні дверні замки дозволяють не турбуватися із приводу різних проблем із ключами. Як часто вам доводило втрачати ключі? Скільки разів ви міняли замок на дверях? Ми всі зіштовхувалися з такими ситуаціями й нам добре знайоме розпач, що випробовуєш, коли доводитися стояти перед замкненими дверима не в силах що-небудь зробити.

Біометричні замки вирішують масу проблем для родини, пов'язаних з необхідністю виготовлення запасних ключів і, саме головне, уже не прийдеться турбуватися із приводу втрати ключів дітьми або вторгнення зловмисників, що використовують підроблені ключі.

Дверні замки з ідентифікацією по відбитках пальців дуже просто встановлюються, як і будь-які інші замки. Кожний замок оснащений біометричним сканером, що зчитує відбитки пальців. Ті люди, чиї відбитки були записані, можуть безперешкодно одержувати доступ до приміщень у будь-який час. Відбитки пальців можуть швидко записуватися й віддалятися, і у випадку спільного з ким-небудь проживання можна легко додавати нові відбитки й видаляти непотрібні.

Біометричні сейфи

Біометричні сейфи, такі як біометричні портфелі, біометричні сейфи з ідентифікацією по відбитках пальців, біометричні сейфи для зберігання зброї й

т.д. являють наочний приклад того, як біометрична технологія може підвищити якість життя й забезпечити безпеку родини.

Біометричні сейфи дуже прості у використанні. Не потрібно запам'ятовувати кодові комбінації, зовсім виключена можливість підробки ключів. Біометричні сейфи вирішують всі ці проблеми, так як все, що потрібно – це торкнутися замка подушкою пальця. Біометричні сейфи з ідентифікацією по відбитках пальців стають усе більше популярними для забезпечення схоронності особистого майна будинку й в офісі. Існує багато різних видів біометричних сейфів з різними функціями залежно від потреб користувача. Наприклад, люди здобувають біометричні сейфи для зберігання зброї, щоб не допустити до нього дітей, а персональні сейфи з ідентифікацією по відбитках пальців використовуються для зберігання дорогих предметів, наприклад, коштовностей. Більше міцні сейфи для зберігання майна й коштовних речей використовуються в офісах. Таким чином, біометричні сейфи забезпечують просте й зручне рішення проблем безпечного зберігання особистих речей, зброї або ділової документації для широкого кола потенційних користувачів.

Рекомендації із проведення аудита біометричних систем

Міжнародна асоціація фахівців в області IT-керування, що раніше діяла за назвою ISACA (Information Systems Audit and Control Association) (Information Systems Audit and Control Association), опублікувала восени 2013 року рекомендації із проведення аудита біометричних систем. Їхня головна мета – полегшити формування незалежної оцінки функціонування згаданих систем, їхньої відповідності корпоративної IT-політиці й кращому світовому досвіду.

Рекомендації розкривають методику проведення аудита по наступних напрямках:

– оцінка ефективності архітектури біометричних систем, рівня їхньої безпеки й відповідності корпоративним стандартам IT-безпеки й функціонування інформаційних систем у цілому;

– оцінка готовності біометричних систем до відбиття спроб вторгнення й роботі в умовах «падіння» одного або декількох їхніх найважливіших компонентів;

– ідентифікація факторів, які можуть зробити негативний вплив на функціонування біометричних систем і їхня здатність контролювати фізичний доступ у будинки й приміщення й захищати корпоративні інформаційні ресурси.

Рекомендації охоплюють всі стадії життєвого циклу біометричних систем – від придбання відповідного програмного й апаратного забезпечення, проектування архітектури цих систем до уведення їх у дію, наступної експлуатації й забезпечення безпеки. У рекомендації включений опис різних політик, стандартів і процедур, які варто враховувати при проведенні аудита біометричних систем; наведені також ради по забезпеченню їхньої готовності до відбиття різних погроз (спроб вторгнення, витоків і т.д.).

3.2 Розробка структурної схеми

Структурна схема системи зображена на рисунку 3.1.

З неї ми бачимо, що розроблена у даній роботісистема автентифікації за допомогою біопараметричних характеристик складається із двох частин – клієнтської частини на стороні користувача й серверної частини що перебуває віддалено в Інтернеті.

Застосовуючи стандартизовані бібліотеки Bioparametrs Aplication Programming Interface і використовуючи стандартизований формат передачі повідомлень користувач може використовувати будь-які біопараметричні сканери.

При створені й тестуванні системи автор використовував два біопараметричні сканери, а саме – сканер відбитків пальців MorphoSmart 1300 Sagem для розпізнання папілярного рисунка пальця й планшетний сканер SignatureGem 1x5 розпізнавання підпису людини.

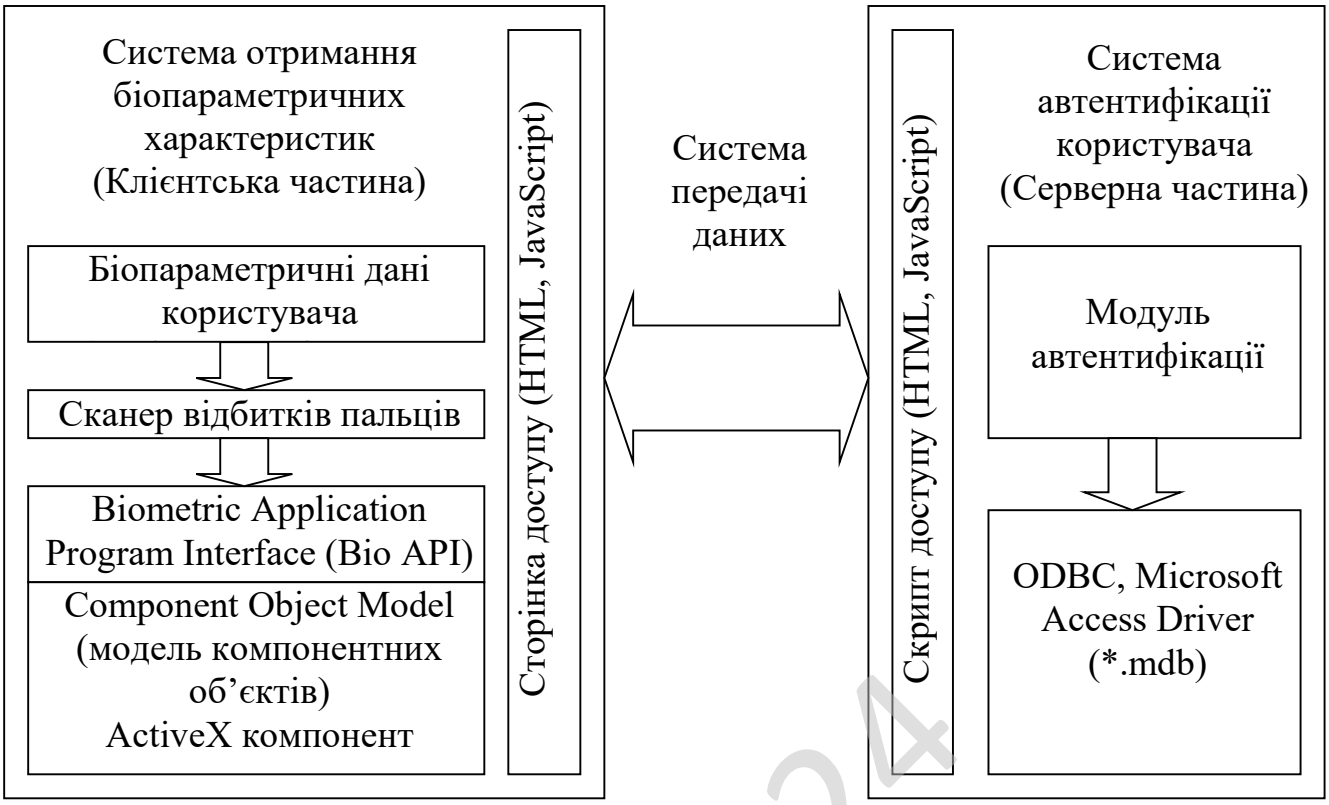

Рисунок 3.1 – Структурна схема роботи системи

Клієнтська частина виконана у вигляді Active X додатка з застосуванням стандартизованої бібліотеки Bioparametrs Aplication Programming Interface і застосовна з Microsoft Edge, а також у будь-яких інших програмах, що підтримують роботу з Active X компонентами.

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

Однією з найбільших переваг пристроїв для розпізнавання рук є простота їх використання. Майже в усіх системах, що базуються на геометрії та венах, рука користувача кладеться долонею вниз на пристрій на короткий проміжок часу, щоб дозволити перевірку один до одного або навіть ідентифікацію один до багатьох з відносно невеликою групою користувачів. У деяких пристроях використовуються напрямні кілки, щоб направляти пальці користувача в потрібне положення.

Твердотільна камера фіксує телевізійне зображення верхньої частини руки з додатковим використанням дзеркала. Потім він вимірює тривимірний розмір і форму пальців і кісточок пальців і стискає цю інформацію, щоб сформувати дев'ятибайтовий шаблон – виконується понад 90 вимірювань. Завдяки стандартному розташуванню напрямних кілків пристрою зазвичай вимірюють лише праву руку людини. Хоча з практикою можна використовувати ліву руку людини, повернуту долонею вгору. (Були певні одноразові встановлення з використанням зчитувачів, які приймають ліві руки, але це не зазвичай.)

Система розглядає лише форму та характеристики вказівного та середнього пальців обох рук і має розмір шаблону 20 байт. Першими двома пальцями будь-якої руки користувачі роблять форму «v» і кладуть їх на валик. Користувачеві допомагають звукові, візуальні та тактильні підказки, щоб дозволити правильно позиціонувати палець. Повідомляється, що час перевірки становить одну секунду.

В обох системах PIN-код необхідно ввести в режимі перевірки, щоб викликати довідковий шаблон користувача. Цей PIN-код можна ввести в PIN-код або зберігати на смарт-картці/картці з магнітною смугою.

Завдяки простоті використання більшості ручних геометричних систем вони, як правило, не страждають від високого рівня невдач для реєстрації, що може бути проблемою з іншими біометричними системами. Однак такі фактори, як погода, температура, певні захворювання та процес старіння (особливо у дуже літніх і дуже молодих людей) іноді можуть змінити розмір руки. У програмі, де система добре використовується, і з часом часто оновлюються шаблони, можна досягти хорошої статистики FAR/FRR. Розробки програмного забезпечення за останні кілька років покращили здатність систем враховувати такі предмети, як кільця та косметичні нігті.

Розмір пристроїв, який обмежує їх використання в деяких програмах, не є областю, яка активно розглядалася в останні роки. Проте нещодавно технологічним доповненням є система, яка поєднує геометрію руки та

розпізнавання обличчя. За п’ять секунд це трохи повільніше, ніж система геометрії руки під час перевірки особи людини, але, як повідомляється, це покращує продуктивність пристрою.

Розпізнавання відбитків долоні

Поверхня долоні містить виступи та западини, як і відбитки пальців. Останніми роками розпізнавання відбитків долонь з’явилося на світ завдяки зростанню бази даних шаблонів. Відбитки долонь мають певні переваги перед відбитками пальців, наприклад надання додаткових функцій, таких як зморшки та основні лінії, які можна легко виділити із зображень із дещо нижчою роздільною здатністю. Відбитки долонь надають більше інформації, ніж відбитки пальців, тому відбитки долонь можна використовувати для створення ще точнішої біометричної системи.

Для розпізнавання відбитків долонь ми досліджуватимемо два методи: перший [35] використовує архітектуру згорткової нейронної мережі (CNN-F). Він має вісім шарів: п'ять згорткових і три повнозв'язних. Експеримент щодо того ж над базою даних відбитків долонь PolyU можна знайти в [35].

Інший метод розпізнавання відбитків долонь походить від дослідження, представленого в [36]. У цьому дослідженні використовується сіамська мережева модель. Як пояснювалося раніше, сіамська мережа складається з двох мереж філій зі спільною вагою. Для розпізнавання долоні обидва методи використовують мережі CNN, обидва з моделлю VGG-16. Нарешті, замість простої повністю пов’язаної мережі ми маємо мережу прийняття рішень, яка поєднує рівні в обох галузевих мережах. Архітектура складається з подвійного п’ятизгорткового шару та трьох повністю з’єднаних шарів у кінці, за якими йде SoftMax.

Розпізнавання відбитка долоні – це нова біометрична технологія, яка пропонує ефективну та надійну особисту автентифікацію. Існує великий попит на безконтактну біометрію через різні соціальні та санітарні проблеми. Однак розробка та розробка безконтактної системи непроста. Нижче наведено

проблеми, пов’язані з розробкою безконтактної системи розпізнавання відбитків долоні:

– Відстань між рукою та датчиком введення: оскільки рука користувача не торкається жодної платформи, відстань руки від датчика введення може відрізнятися. Якщо руку розмістити надто далеко від датчика введення, деталі відбитка долоні будуть втрачені. З іншого боку, якщо рука розташована надто близько до датчика введення, датчик може не захопити все зображення руки, і деяка частина відбитка долоні може бути відсутня. Таким чином, повинна бути розроблена система, яка допускає гнучкий діапазон відстані між рукою та датчиком введення.

– Положення та орієнтація руки: Оскільки для обмеження руки користувача не використовується напрямний кілочок, користувач може розміщувати свою руку в різних напрямках і положеннях. Система повинна впоратися зі змінами положення та орієнтації руки користувача в менш обмежувальному середовищі.

– Зміни освітлення: зміна освітлення може мати значний вплив на здатність системи розпізнавати людей. Таким чином, система повинна бути здатна узагальнювати зображення відбитків долоні при зміні освітлення.

3.3 Розробка функціональної схеми

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

Програмне забезпечення функціонально складається з наступних блоків:

– Графічний інтерфейс користувача.

– Блок конфігурації й запуску.

– Блок реєстрації біометричної інформації про людину.

– Блок визначення часу доступу.

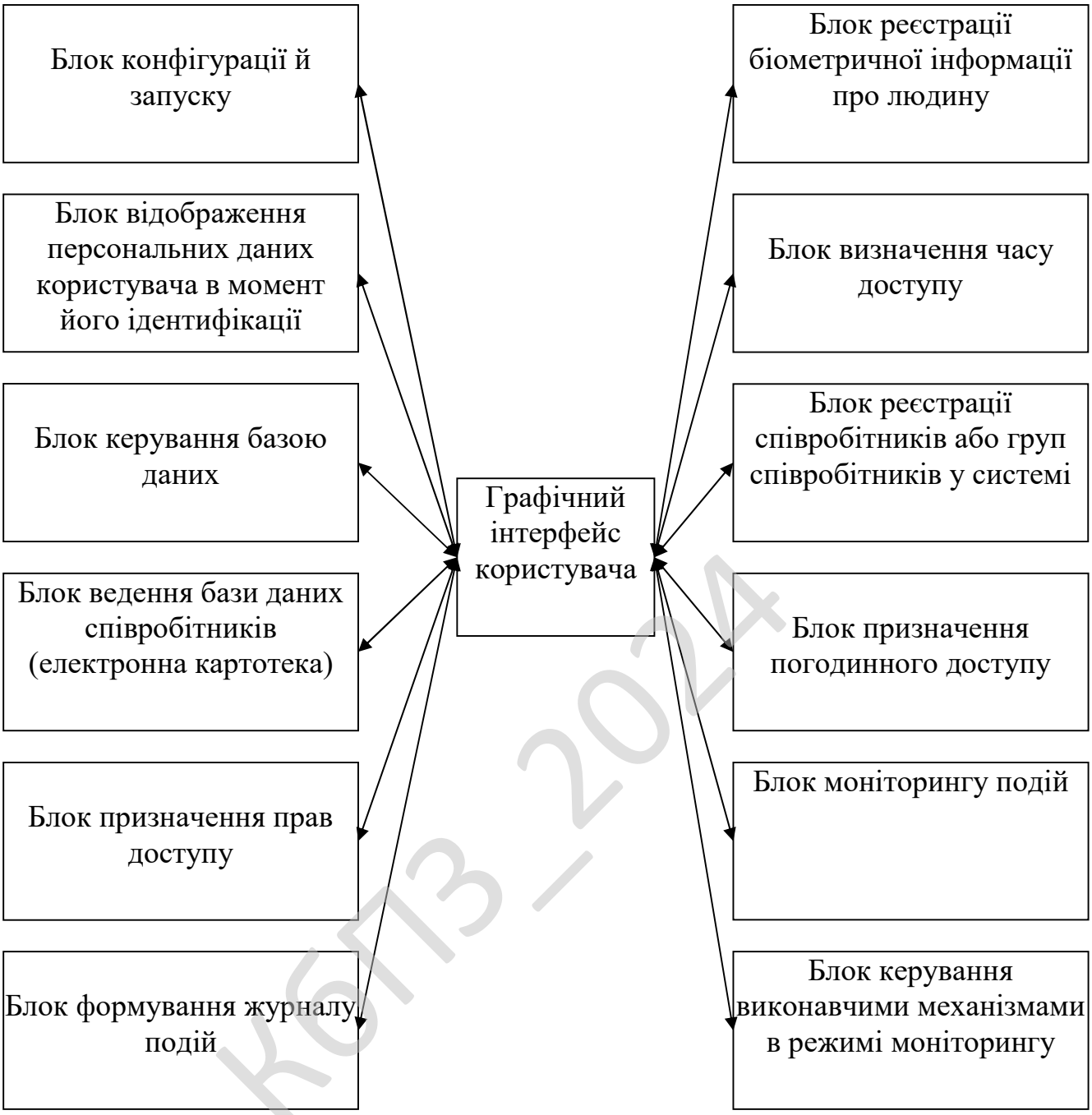

Рисунок 3.2 – Функціональна схема роботи системи

– Блок відображення персональних даних користувача в момент його ідентифікації.

– Блок керування базою даних.

– Блок реєстрації співробітників або груп співробітників у системі.

– Блок ведення бази даних співробітників (електронна картотека).

– Блок призначення погодинного доступу.

– Блок призначення прав доступу.

– Блок моніторингу подій.

– Блок формування журналу подій.

– Блок керування виконавчими механізмами в режимі моніторингу.

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

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

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

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

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

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

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

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

– Тло й перешкоди. Предмет, що ми сприймаємо як щось окреме, на зображені ніяк не виділений, і перебуває на тлі інших предметів. Крім того, зображення не ідеально й може бути піддано всякого роду перекручуванням і перешкодам.

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

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

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

– мати інший масштаб;

– бути повернуть у площині зображення;

– бути в довільному місці сцени;

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

– може мати відмінну від зразка яскравість і контраст;

– його може не бути зовсім.

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

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

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

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

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

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

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

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

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

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

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

Після початку роботи розробленого ПЗ ми потрапляємо до головного вікна ПЗ звідки через моніторинг подій можемо керувати виконавчими механізмами та призначати погодинний доступ.

Далі через блок конфігурації й запуску переглядати журнал подій, керувати БД, призначати права доступу та через порівняння зображень проводити виведення знайдених зображень з можливістю виведення значення схожості у процентах, виведення інформації про зображення, ідентифікації людини.

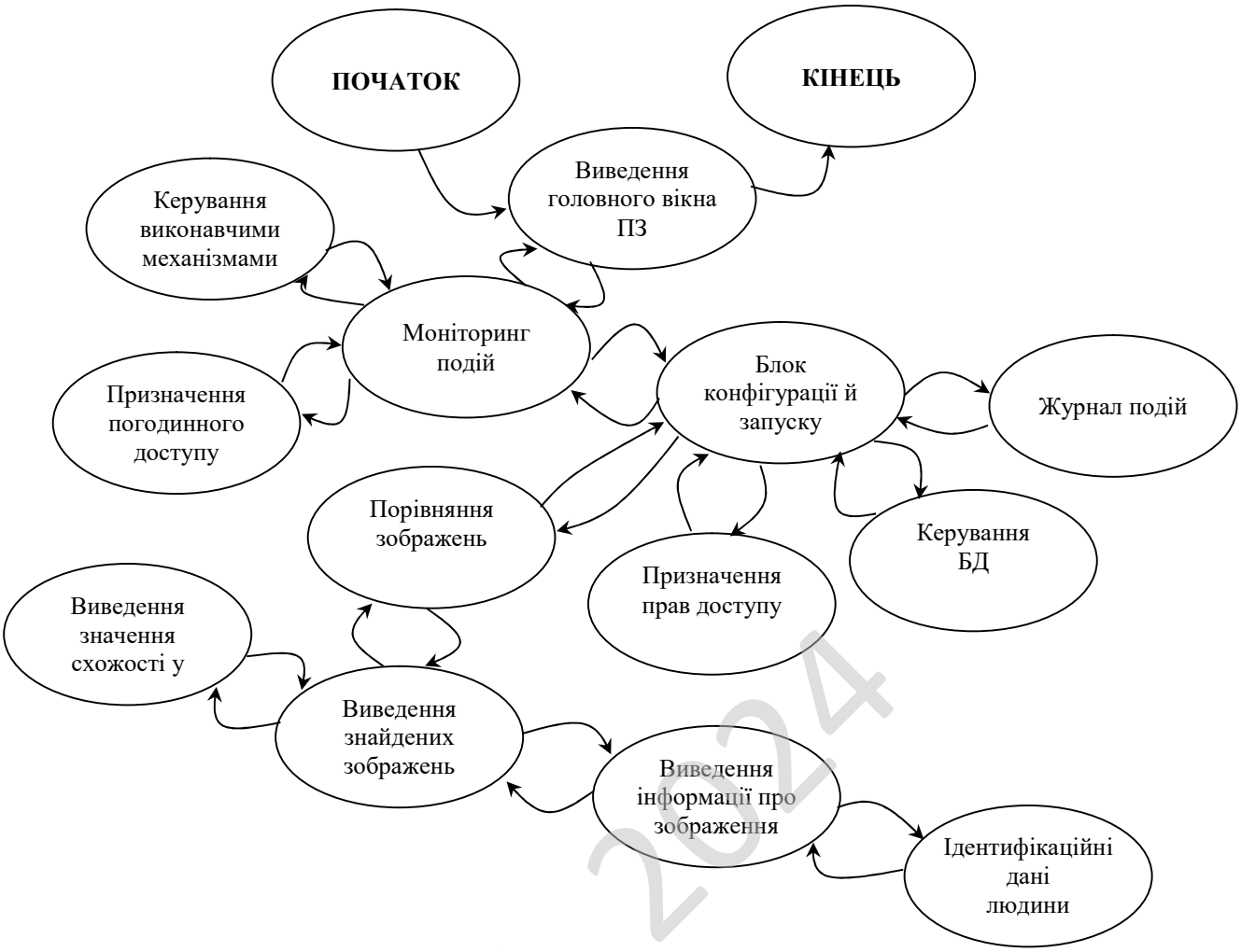

Рисунок 3.3 – Діаграма взаємодії процесів

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. Її робота складається з виконання наступних кроків:

– Ініціалізація та виведення головного вікна ПЗ.

– Завантажити ресурси ПЗ?

– Завантаження мультимедіа даних.

– Сигнал пошуку ключових точок?

– Обчислення матриці пошуку.

– Створення дескрипторів точок.

– Створення файлу даних дескрипторів зображення.

– Виведення підмножин місцезнаходження точок.

– Виведення ліній, що показують напрям градієнта яскравості.

– Пошук подібного зображення у БД?

– Підпрограма пошуку зображення у БД.

– Подібне зображення знайдено?

– Виведення знайденого зображення та значення схожості у процентах.

– Виведення інформації про зображення.

– Сигнал WM_CLOSE?

На рисунку 4.2 наведено блок-схему підпрограми пошуку зображення у БД. Її робота складається з виконання наступних кроків:

– Запит відбитків пальців?

– Читання дескрипторів ключових точок.

– Запит та відкриття БД відбитків пальців.

– Поки не перевірені усі зображення БД.

– Читання дескрипторів ключових точок поточного зображення .

– Порівняння дескрипторів зображення.

– Відповідність більше межі 65%?

– Повернення ID поточного зображення та відсотків відповідності.

– Поки не перевірені усі зображення БД.

– Читання дескрипторів ключових точок поточного зображення .

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

– Відповідність дескрипторів > 75?

– Повернення індексу поточного зображення та проценту відповідності основній програмі.

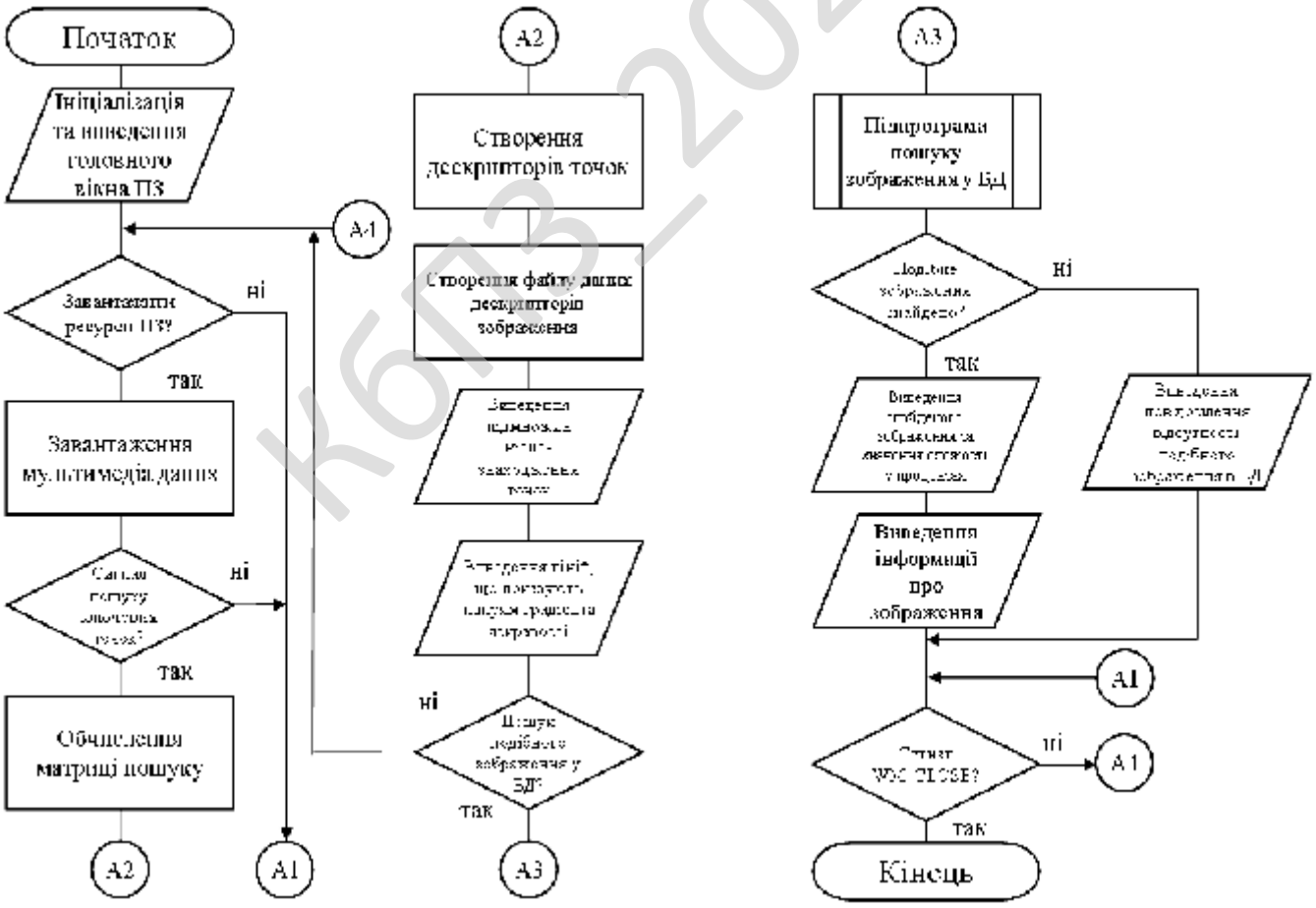

Рисунок 4.1 – Блок-схема основної програми

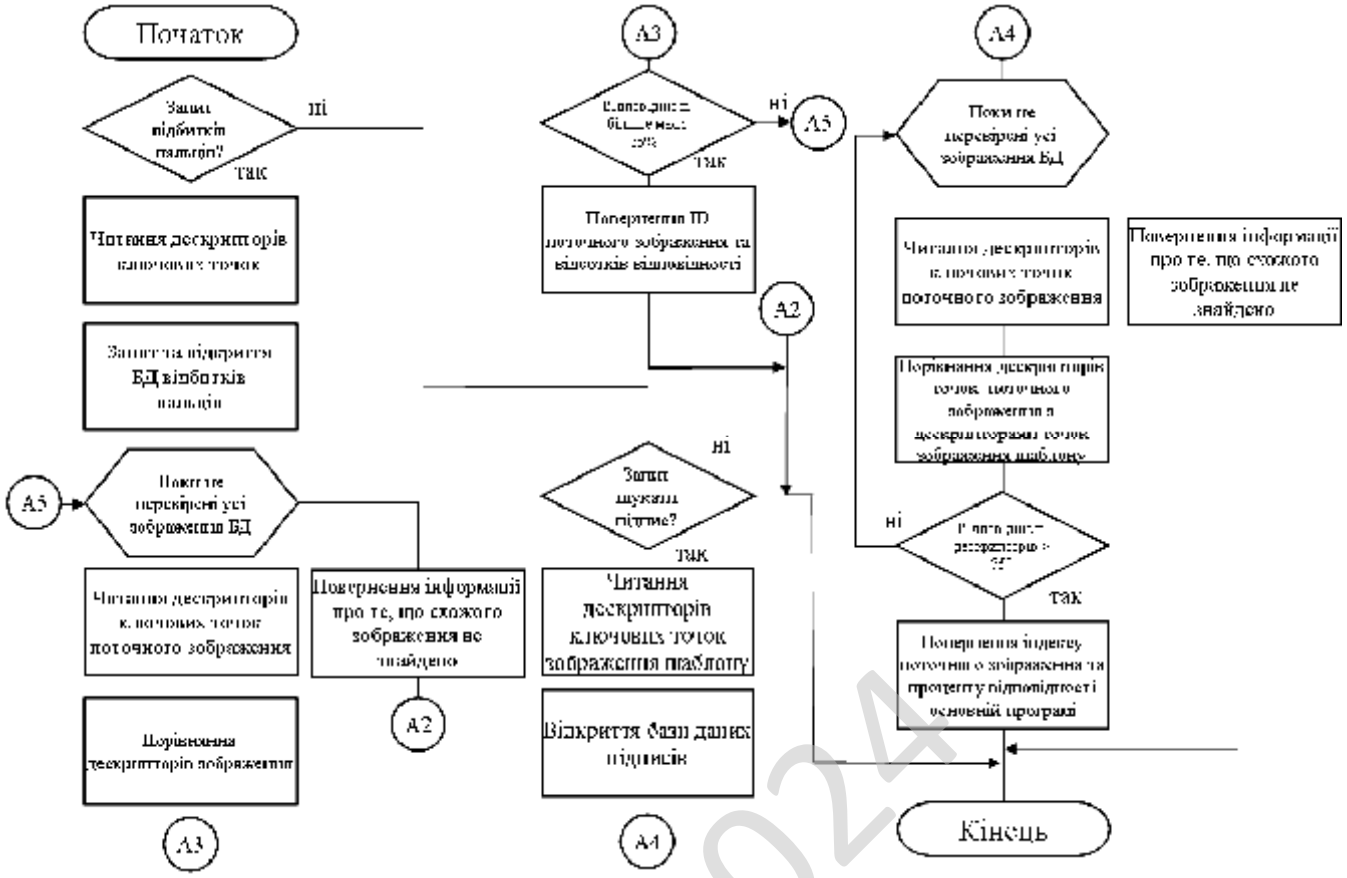

Рисунок 4.2 – Блок-схема підпрограми пошуку зображення у БД

При розробці програми використовувалася Матриця Гессе — квадратна матриця елементами якої є часткові похідні деякої функції. Поняття було введене Людвігом Отто Гессе (1844), який використовував іншу назву. Термін матриця Гессе був введений Джеймсом Джозефом Сильвестром. Використовується для порівняння дескрипторів зображення.

Матриця других похідних цільової функції f (x). Має наступний вигляд:

$$f''(x) = \begin{matrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{matrix}. \quad (4.1)$$

Матриці Гессе застосовуються в методі Ньютона, який використовується в якості одного з методів оптимізації при навчанні нейронних мереж. Визначник цієї матриці називається визначником Гессе або гессіаном.

Якщо гессіан позитивно визначений, то x – точка локального мінімуму цільової функції f, якщо гессіан негативно визначений, то x – точка локального максимуму функції f.

Характеристика матриці Гессе (її негативна або позитивна визначеність і полуопределенность) служить умовою для визначення виду стаціонарної точки: чи є вона, відповідно, максимумом, мінімумом або сідловою в задачі оптимізації функції.

Наведемо частину коду, яка реалізує роботу з матрицею Гессе (гесіаном) тобто реалізацію порівняння дескрипторів зображення.

```
//! Початкові оператори мови
//-------------------------------------------------------
//! 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);
        }
      }
    }
  }
}
//-------------------------------------------------------
//! 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;
}
//! 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;
}
//-------------------------------------------------------
//! 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;
}
//-------------------------------------------------------
//! 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]);
  }
}
//-------------------------------------------------------
//! 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];
}
```


4.2 Захист розробленого програмного забезпечення

Дані у програмному забезпеченні я захищаю за допомогою MISTY1. MISTY1 – блоковий алгоритм шифрування, створений для компанії Mitsubishi Electric криптологом Міцуру Мацуї. Назва є абревіатурою Mitsubishi Improved Security Technology. Алгоритм був розроблений в 1995-1996 рр. Відомі також дві модифікації алгоритму MISTY1: MISTY2 і KASUMI

Шифр став переможцем на Європейському конкурсі NESSIE. У результаті аналізу алгоритму експерти зробили вивід, що ніяких серйозних уразливостей

даний алгоритм не має (переважно, завдяки вкладеним мережам Фейстеля, що суттєво утрудняє криптоаналіз). У нього високий запас криптостійкості, алгоритм має високу швидкість шифрування й досить ефективний для апаратної реалізації.

Алгоритм був розроблений на основі теорії «підтвердженої безпеки» проти диференціального й лінійного криптоаналізу. Цей алгоритм був спроектований, щоб протистояти криптоатакам, відомим на момент створення.

З моменту публікації MISTY1 було проведено багато досліджень, щоб оцінити його рівень безпеки.

Диференціальний і неможливий диференціальний криптоаналіз високого порядку ефективно застосовується до блокових шифрів з малим ступенем. Найкращі результати для обох варіантів були отримані для 5-рівневого алгоритму MISTY1 без FL функцій.

Саме FL функції й широкобітні AND/OR операції в сильно утрудняють використання диференціального криптоаналізу, що не заважає проведенню в цьому напрямку всі нових досліджень і досягненню усе більш близьких до розв'язку результатів.

Параметри вихідних даних

MISTY1 – це шифр на основі вкладених мереж Фейстеля з варіюємим числом раундів. Рекомендоване використання 8-раундової версії, але може використовуватися будь-яка кількість раундів, кратне 4-м. Розмір блоку вихідного тексту – 64 біта, розмір ключа – 128 біт.

Для роботи алгоритму також попередньо виконується процедура розширення ключа, яка для 8-мі раундів обчислює 1216 бітів ключової інформації з 128-бітного ключа шифрування.

Структура алгоритму

Для задоволення вимогам конкурсу NESSIE, а також для задоволення завдання мультиплатформеності, в алгоритмі MISTY1 використовувалися наступні методи шифрування:

– Логічні операції.

– Арифметичні операції.

– Операції зрушення.

– Таблиці перестановок.

Як говорилося вище, алгоритм MISTY1 заснований на «вкладених» мережах Фейстеля. Спочатку блок вихідного тексту розбивається на два 32-бітних субблоки, після чого виконується r раундів наступних перетворень[1]:

– У кожному непарному раунді обоє субблоки обробляються операцією FL

– Над оброблюваним субблоком виконується операція FO.

– Результат цих операцій накладається логічною операцією «, що виключає або» (XOR) на неопрацьований субблок.

– Субблоки міняються місцями. Після заключного раунду обоє субблоки ще раз обробляються операцією FL.

Операція FL

Оброблюваний 32-бітний субблок розбивається на два 16-бітних фрагмента, до яких застосовуються операції, де:

– L і R – вхідні значення лівого й правого фрагментів відповідно;

– L' і R' – вихідні значення;

– і – фрагменти j-го підключа і-го раунду для функції FL (процедура розширення ключа докладно описана далі);

– і – побітіві логічні операції «і» і «або» відповідно.

Операція FO

Саме ця функція є вкладеною мережею Фейстеля. Тут, як і раніше, виконується розбивка вхідного значення на два 16-бітних фрагмента, що проходять 3 раунду наступних перетворень:

– На лівий фрагмент операцією XOR накладається фрагмент ключа, де k – номер раунду функції FO.

– Лівий фрагмент обробляється операцією FI.

– На лівий фрагмент накладається операцією XOR значення правого фрагмента.

– Фрагменти міняються місцями.

Після третього раунду операції FO на лівий фрагмент накладається операцією XOR додатковий фрагмент ключа.

Операція FI

Дана операція також представляє собою третій рівень вкладеності мережі Фейстеля. На відміну від двох верхніх рівнів, дана мережа є незбалансованою: оброблюваний 16-бітний фрагмент ділиться на дві частини: 9-бітну ліву й 7-бітну праву. Потім виконуються 3 раунду перетворень, що випливають:

– Ліва частина зазнає обробці S-box.9-бітна частина (в 1-м і 3-м раундах) обробляється таблицею S9, а 7-бітна (в 2-м раунді) – таблицею S7. Дані таблиці описані нижче.

– На ліву частину операцією XOR накладається поточне значення правої частини. При цьому, якщо праворуч 7-бітна частина, вона доповнюється нулями ліворуч, а в 9- бітної частини віддаляються ліворуч два біти.

– У другому раунді на ліву частину операцією XOR накладається фрагмент ключа раунду, а на праву – фрагмент. В інших раундах ці дії не виконуються.

– Ліва й права частини міняються місцями.

Для оптимального розв'язку завдання мультиплатформеності, таблиці S7 і S9 алгоритму MISTY1 можуть бути реалізовані як за допомогою обчислень, так і безпосередньо таблицями.

Розширення ключа

Для 8 раундів алгоритму результатом процедури розширення ключа буде наступний набір ключових значень:

– 20 фрагментів ключа (), кожний з яких має розмір по 16 бітів;

– 32 16-бітних фрагмента ();

– 24 7-бітних фрагмента (при k=4, тобто в 4-м раунді функції FO, операція FI не виконується);

– 24 9-бітних фрагмента.

Виконується дане обчислення в такий спосіб:

1. 128-бітний ключ ділиться на 8 фрагментів … по 16 бітів кожний.

2. Формуються значення: у якості використовується результат обробки значення функцією FI, яка в якості ключа (тобто сукупності необхідних 7- і 9-бітного фрагментів) використовує значення(якщо індекс n фрагмента ключа перевищує 8, то замість нього використовується індекс n-8).

Необхідні фрагменти розширеного ключа «набираються» у міру виконання перетворень із відповідних масивів і згідно з відповідними таблицями

16-бітний фрагмент ділиться на 7-бітний фрагмент і 9-бітний .

Розшифрування

Розшифрування проводиться виконанням тих же операцій, що й при зашифруванні, але з наступними змінами:

– фрагменти розширеного ключа використовуються у зворотній послідовності,

– замість операції FL використовується зворотна їй операція – FLI.

Схеми виконання функції FLI і процедури розшифрування наведено на малюнках 6 і 7 відповідно:

Методи аналізу

Як говорилося на початку розділу, диференціальний і неможливий диференціальний аналізи виявилися ефективні лише до версій шифру з меншою кількістю раундів і без операції FL [2][3]. Проте, на даний момент цей напрямок аналізу, особливе використання слабких ключів, найбільше перспективно, тому що наближене до реальних можливих допущень при використанні алгоритму.

Так само, ученим з Японії був проведений інтегральний аналіз повного алгоритму, використовуючи відкритих текстів зі складністю обчислення, рівної [4].

Лінійний аналіз дав результати тільки для 7-раундової версії шифру, і також без операції FL[5].

Так як MISTY1 створювався, у тому числі, з розрахунку на апаратну реалізацію, має сенс диференціальний аналіз, заснований на використанні атаки по помилках обчислень, що в цьому випадку наближене до реальності.

Висновок

Таким чином, була докладно описана структура алгоритму шифрування MISTY1 і розглянуті методи його аналізу, найбільш прагматичні напрямки дослідження. Далі має бути створення програмної реалізації для більш детального розгляду алгоритму й набір статистичних даних для повного дослідження й пошуку оптимального підходу до аналізу MISTY1.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків:

– Вікна результату сканування відбитку долоні.

– Вікна виведення поточної інформації.

– Функціональної кнопки очистити вікно сканування.

– Функціональної кнопки повторне сканування.

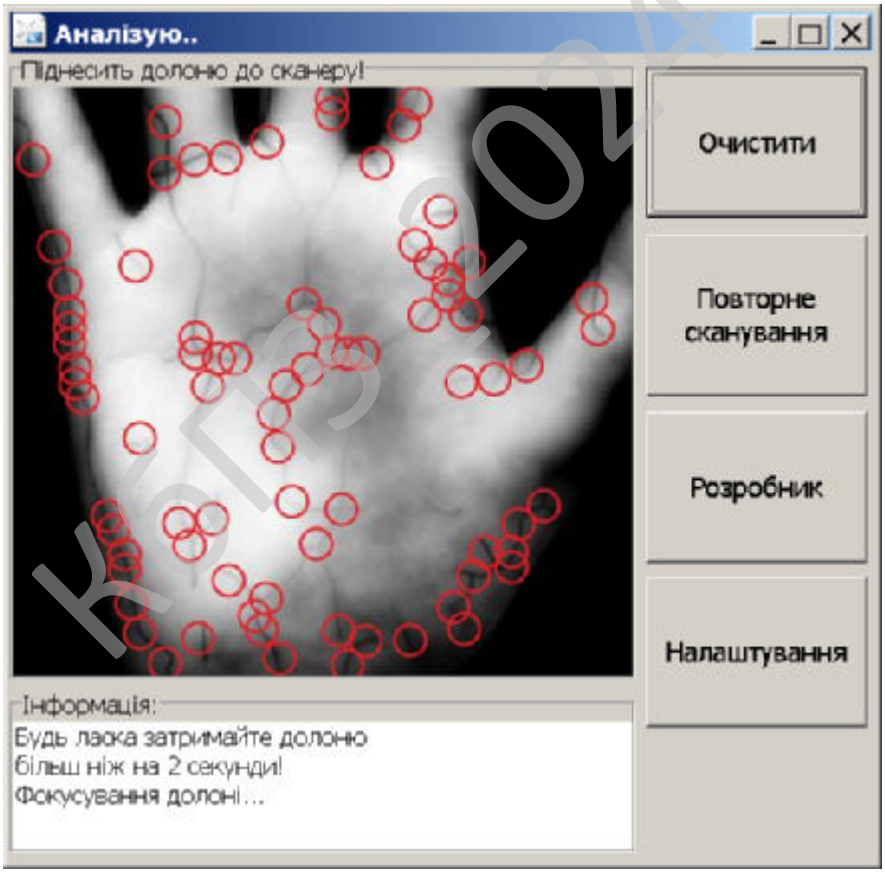

Рисунок 5.1 – Головне вікно програми

– Функціональної кнопки поле розробника;

– Функціональної кнопки налаштування.

На рисунку 5.2 зображено форму авторського права. 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 (неповнофункціональна 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). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (дискету чи CD-ROM). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості. 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.

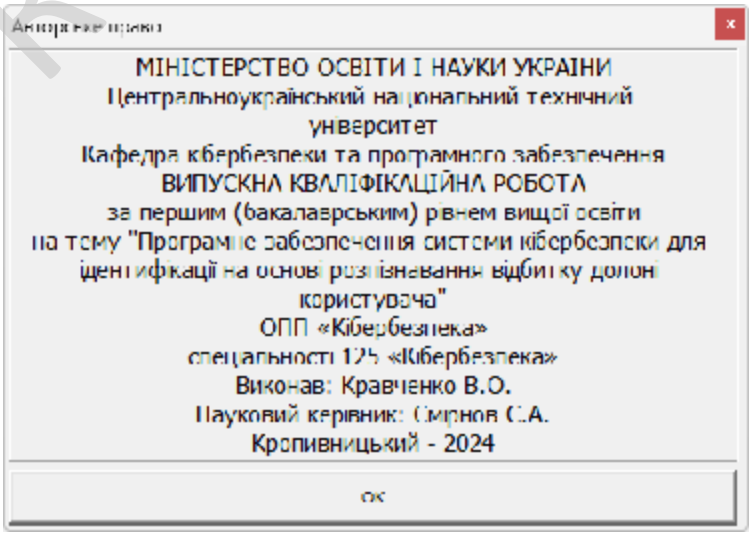

Рисунок 5.2 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для ідентифікації на основі розпізнавання відбитку долоні користувача.

– Досліджена система для ідентифікації на основі розпізнавання відбитку долоні користувача.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для ідентифікації на основі розпізнавання відбитку долоні користувача.

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

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

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм MISTY1.

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

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

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

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

5. 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.

6. 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.

7. 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.

8. 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,

9. 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.

10. 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.

11. 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

12. 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.

13. 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.

14. 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.*

15. 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.

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

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884.

43. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

44. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

45. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична*

конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

46. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

47. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

48. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

49. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

50. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

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. № 135-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для ідентифікації на основі розпізнавання відбитку долоні користувача;

– 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 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 6.06.2024 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 ідентифікації на основі розпізнавання відбитку долоні користувача

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2024 року

integral.cpp – робота з зображенням

```
/*********************************************************
* --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача--- *
* *
*********************************************************/

#include "utils.h"

#include "integral.h"

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

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

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

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

// звільняємо сіре зображення
cvReleaseImage(&img);

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

integral.h – заголовочний файл

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача---                                            *
*
*                                                         *
*********************************************************/

#ifndef INTEGRAL_H
#define INTEGRAL_H

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

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

#ifdef max
  #undef max
#endif

#include <cv.h>

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


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

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

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

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

#endif
```

fasthessian.cpp – реалізація гессіана

```
/*************************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача---
*                                                            *
*************************************************************/

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

#include <vector>

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



using namespace std;

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

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

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

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

  // Встановлюємо поточне зображення
  setIntImage(img);
}

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

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

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

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

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


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

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

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

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

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

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

  // Будуємо карту відповідностей
  buildResponseMap();

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

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

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

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

```
  // Oct4: 51, 99, 147,195
  // Oct5: 99, 195,291,387

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  return 1;
}

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

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

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

  // Якщо точка досить близька до фактичного екстремуму
  if( fabs( xi ) < 0.5f  &&  fabs( xr ) < 0.5f  &&  fabs( xc ) < 0.5f )
  {
    Ipoint ipt;
    ipt.x = static_cast<float>((c + xc) * t->step);
    ipt.y = static_cast<float>((r + xr) * t->step);
    ipt.scale = static_cast<float>((0.1333f) * (m->filter + xi * filterStep));
    ipt.laplacian = static_cast<int>(m->getLaplacian(r,c,t));
    ipts.push_back(ipt);
  }
}

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

//! Виконуємо один крок інтерполяції екстремуму.
void FastHessian::interpolateStep(int r, int c, ResponseLayer *t, ResponseLayer
*m, ResponseLayer *b,
                                  double* xi, double* xr, double* xc )
{
  CvMat* dD, * H, * H_inv, X;
  double x[3] = { 0 };

  dD = deriv3D( r, c, t, m, b );
  H = hessian3D( r, c, t, m, b );
  H_inv = cvCreateMat( 3, 3, CV_64FC1 );
  cvInvert( H, H_inv, CV_SVD );
  cvInitMatHeader( &X, 3, 1, CV_64FC1, x, CV_AUTOSTEP );
  cvGEMM( H_inv, dD, -1, NULL, 0, &X, 0 );

  cvReleaseMat( &dD );
  cvReleaseMat( &H );
  cvReleaseMat( &H_inv );

  *xi = x[2];
  *xr = x[1];
  *xc = x[0];
}

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

//! Обчислюємо часткові похідні слова в x, y, і масштаб пікселя.
CvMat* FastHessian::deriv3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* dI;
  double dx, dy, ds;

  dx = (m->getResponse(r, c + 1, t) - m->getResponse(r, c - 1, t)) / 2.0;
  dy = (m->getResponse(r + 1, c, t) - m->getResponse(r - 1, c, t)) / 2.0;
  ds = (t->getResponse(r, c) - b->getResponse(r, c, t)) / 2.0;

  dI = cvCreateMat( 3, 1, CV_64FC1 );
  cvmSet( dI, 0, 0, dx );
  cvmSet( dI, 1, 0, dy );
```

```
  cvmSet( dI, 2, 0, ds );

  return dI;
}

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

//! Обчислюємо 3D матрицю гессіана для пікселя.
CvMat* FastHessian::hessian3D(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b)
{
  CvMat* H;
  double v, dxx, dyy, dss, dxy, dxs, dys;

  v = m->getResponse(r, c, t);
  dxx = m->getResponse(r, c + 1, t) + m->getResponse(r, c - 1, t) - 2 * v;
  dyy = m->getResponse(r + 1, c, t) + m->getResponse(r - 1, c, t) - 2 * v;
  dss = t->getResponse(r, c) + b->getResponse(r, c, t) - 2 * v;
  dxy = ( m->getResponse(r + 1, c + 1, t) - m->getResponse(r + 1, c - 1, t) -
          m->getResponse(r - 1, c + 1, t) + m->getResponse(r - 1, c - 1, t) ) /
4.0;
  dxs = ( t->getResponse(r, c + 1) - t->getResponse(r, c - 1) -
          b->getResponse(r, c + 1, t) + b->getResponse(r, c - 1, t) ) / 4.0;
  dys = ( t->getResponse(r + 1, c) - t->getResponse(r - 1, c) -
          b->getResponse(r + 1, c, t) + b->getResponse(r - 1, c, t) ) / 4.0;

  H = cvCreateMat( 3, 3, CV_64FC1 );
  cvmSet( H, 0, 0, dxx );
  cvmSet( H, 0, 1, dxy );
  cvmSet( H, 0, 2, dxs );
  cvmSet( H, 1, 0, dxy );
  cvmSet( H, 1, 1, dyy );
  cvmSet( H, 1, 2, dys );
  cvmSet( H, 2, 0, dxs );
  cvmSet( H, 2, 1, dys );
  cvmSet( H, 2, 2, dss );

  return H;
}

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

fasthessian.h – заголовочний файл

```
/*************************************************************
* --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача--- *
* *
*************************************************************/

#ifndef FASTHESSIAN_H
#define FASTHESSIAN_H

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

#include <vector>

class ResponseLayer;
static const int OCTAVES = 5;
static const int INTERVALS = 4;
static const float THRES = 0.0004f;
static const int INIT_SAMPLE = 2;


class FastHessian {

public:

//! Конструктор без зображення
FastHessian(std::vector<Ipoint> &ipts,
const int octaves = OCTAVES,
const int intervals = INTERVALS,
const int init_sample = INIT_SAMPLE,
const float thres = THRES);

//! Конструктор з зображенням
FastHessian(IplImage *img,
std::vector<Ipoint> &ipts,
const int octaves = OCTAVES,
const int intervals = INTERVALS,
const int init_sample = INIT_SAMPLE,
const float thres = THRES);

//! Деструктор
~FastHessian();

//! Зберігаємо параметри
void saveParameters(const int octaves,
const int intervals,
const int init_sample,
const float thres);

//! Встановлюємо або перевстановлюємо джерело цілочиселього зображення
void setIntImage(IplImage *img);

//! Знайходимо, що зображення змальовує і вписуємо у вектор особливостей
void getIpoints();

private:

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

//! Будуємо карту відповідностей DoH
void buildResponseMap();

//! Обчислюємо відповідний DoH для забезпечуваного шару
void buildResponseLayer(ResponseLayer *r);

//! 3x3x3 тест на екстремум
```

```
int isExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);

//! Функція інтерполяції – адаптовується для SIFT додатку
void interpolateExtremum(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b);
void interpolateStep(int r, int c, ResponseLayer *t, ResponseLayer *m,
ResponseLayer *b,
double* xi, double* xr, double* xc );
CvMat* deriv3D(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);
CvMat* hessian3D(int r, int c, ResponseLayer *t, ResponseLayer *m, ResponseLayer
*b);

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

//! Точка цілочисельного зображення , та її атрибути
IplImage *img;
int i_width, i_height;

//! Посилаємося до вектора особливостей проходів за межами
std::vector<Ipoint> &ipts;

//! Стек відповідностей детермінанту значення гессіана
std::vector<ResponseLayer *> responseMap;

//! Число октетів
int octaves;

//! Число інтервалів між октетами
int intervals;

//! Початковий пробний крок для виявлення Ipoint
int init_sample;

//! Порогове значення для  відповідностей кіл
float thresh;
};


#endif
```

responselayer.h – заголовочний файл ля шару відповідностей

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача---                                           *
*********************************************************/


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

class ResponseLayer
{
public:

  int width, height, step, filter;
  float *responses;
  unsigned char *laplacian;

  ResponseLayer(int width, int height, int step, int filter)
  {
    assert(width > 0 && height > 0);

    this->width = width;
    this->height = height;
    this->step = step;
    this->filter = filter;

    responses = new float[width*height];
    laplacian = new unsigned char[width*height];

    memset(responses,0,sizeof(float)*width*height);
    memset(laplacian,0,sizeof(unsigned char)*width*height);
  }

  ~ResponseLayer()
  {
    if (responses) delete [] responses;
    if (laplacian) delete [] laplacian;
  }

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

  inline unsigned char getLaplacian(unsigned int row, unsigned int column,
ResponseLayer *src)
  {
    int scale = this->width / src->width;

    #ifdef RL_DEBUG
    assert(src->getCoords(row, column) == this->getCoords(scale * row, scale *
column));
    #endif

    return laplacian[(scale * row) * width + (scale * column)];
  }

  inline float getResponse(unsigned int row, unsigned int column)
  {
    return responses[row * width + column];
  }

  inline float getResponse(unsigned int row, unsigned int column, ResponseLayer
*src)
  {
    int scale = this->width / src->width;
```

```
    #ifdef RL_DEBUG
    assert(src->getCoords(row, column) == this->getCoords(scale * row, scale *
column));
    #endif

    return responses[(scale * row) * width + (scale * column)];
  }

#ifdef RL_DEBUG
  std::vector<std::pair<int, int>> coords;

  inline std::pair<int,int> getCoords(unsigned int row, unsigned int column)
  {
    return coords[row * width + column];
  }

  inline std::pair<int,int> getCoords(unsigned int row, unsigned int column,
ResponseLayer *src)
  {
    int scale = this->width / src->width;
    return coords[(scale * row) * width + (scale * column)];
  }
#endif
};
```

kmeans.h – заголовочний файл

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача---                                                    *
*********************************************************/

#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<< "Програма системи ідентифікації на основі розпізнавання відбитку
долоні користувачазнайшла: " << ipts.size() << " особливі точки" << std::endl;
```

```
  std::cout<< "Програма системи ідентифікації на основі розпізнавання відбитку
долоні користувачавиконується: " << float(end - start) / CLOCKS_PER_SEC  << "
секунд" << std::endl;

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

  // Виведення результату на екран
  showImage(img);

  return 0;
}

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

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

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

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

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

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

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

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

    // Виводимо фігуру FPS
    drawFPS(img);

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

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

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


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


int mainMatch(void)
{
```

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

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

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

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

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

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

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

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

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

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

    // Виводимо фігуру FPS
    drawFPS(img);

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

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

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
```

```
  cvDestroyWindow( "Програма розпізнання біопараметричних характеристик методом
SURF" );
  return 0;
}


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


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

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

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

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

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

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

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

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

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


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

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

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

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

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

    const int & w = img1->width;

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

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

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

  return 0;
}

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

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

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

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

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

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

    showImage(img);
  }

  return 0;
}

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

int main(void)
{
  if (PROCEDURE == 1) return mainImage();
  if (PROCEDURE == 2) return mainVideo();
```

```
  if (PROCEDURE == 3) return mainMatch();
  if (PROCEDURE == 4) return mainMotionPoints();
  if (PROCEDURE == 5) return mainStaticMatch();
  if (PROCEDURE == 6) return mainKmeans();
}
```

surf.cpp – реалізація алгоритму SURF

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача---                                          *
*********************************************************/

#include "utils.h"

#include "surf.h"

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

const double gauss25 [7][7] = {

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

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

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

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

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

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

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

const double gauss33 [11][11] = {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

//! Беремо модифікований дескріптор.

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

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

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

i = -8;

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

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

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

    j = j - 4;

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

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

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

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

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

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

      }
    }

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

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

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

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

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

}


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

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

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

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

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

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

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

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

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

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

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

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

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

  return 0;
}
```

surf.h – файл заголовків

```
/*************************************************************
* --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача--- *
*************************************************************/

#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

```
/*********************************************************
* --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача--- *
*
* *
*********************************************************/

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

#include "utils.h"

using namespace std;

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

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

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

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

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

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

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

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

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

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

IplImage* gray8, * gray32;

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

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

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

cvReleaseImage( &gray8 );
return gray32;
}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}

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

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

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

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

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

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

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

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

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


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

}
}

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

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

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

```
t=clock();
counter = 0;
}

// Інкрементуємо лічильник
++counter;

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

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

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

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

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

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

outfile << "\n";
}

outfile.close();
}

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

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

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

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

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

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

```
infile >> ipt.scale;

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

ipts.push_back(ipt);

}
}

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

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

utils.h – заголовочний файл

```
/*********************************************************
* --- Програма системи ідентифікації на основі розпізнавання відбитку долоні користувача--- *
*********************************************************/

#ifndef UTILS_H
#define UTILS_H

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

#include <vector>


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

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

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

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

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

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

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

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

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

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

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

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

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

#endif
```

ipoint.cpp – підпрограма визначення базових точок

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача---                                            *
*
*
*********************************************************/

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

#include "ipoint.h"

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

  matches.clear();

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

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

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

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

//
// Ця функція користується CV_RANSAC (
//

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

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

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

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

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

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

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

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

ipoint.h – заголовочний файл

```
/*********************************************************
*  --- Програма системи ідентифікації на основі розпізнавання відбитку долоні
користувача---                                          *
*
*                                                        *
**********************************************************/

#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
```