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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів”

Виконав здобувач вищої освіти
ІІ курсу, групи КН-22М-1
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Мошуренко Д.А.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Доренський О.П.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

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

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Мошуренко Дмитру Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів*

2. Керівник роботи *Доренський Олександр Павлович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 32-13 від 04.08.2023 року

3. Строк подання студентом роботи до захисту *10.12.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів*

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

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

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

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2023	14.11.2023
Охорона праці	Оришака О.В.	06.10.2023	16.11.2023

7. Дата видачі завдання « 6 » вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2023 р.	
3.	Розробка моделі компонента	20.10.2023 р.	
4.	Розробка структур даних	25.10.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2023 р.	
6.	Програмування алгоритмів	10.11.2023 р.	
7.	Розрахунок економічної ефективності	13.11.2023 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2023 р.	
9.	Оформлення ПЗ	17.11.2023 р.	
10.	Попередній захист роботи	10.12.2023 р.	

Дата видачі завдання
« 6 » вересня 2023 р.

Підпис керівника

_____ _____
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2023 р.

Підпис здобувача

_____ _____
(прізвище та ініціали)

АНОТАЦІЯ

Мошуренко Д.А. Дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Метою розробки є дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Об'єктом дослідження є процес комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Предметом дослідження є методи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Методи дослідження базуються на методах розпізнавання образів, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

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

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

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

Програму розроблено в середовищі Visual C++.

Ключові слова: комп'ютерні науки, комп'ютерний зір

ABSTRACT

Moshurenko D.A. Research and software implementation of a computer vision system for creating interactive user interfaces. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for the computer vision system for creating interactive user interfaces.

The purpose of the development is research and software implementation of a computer vision system for creating interactive user interfaces.

The object of research is the computer vision process for creating interactive user interfaces.

The subject of research is computer vision methods for creating interactive user interfaces.

Research methods are based on pattern recognition methods, mathematical statistics methods, and software development methods.

The result of the work is a software implementation of a computer vision system for creating interactive user interfaces.

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

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

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

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

Keywords: computer science, computer vision

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

CBIR	–	content-based image retrieval
QBIC	–	query by image content
ПЗ	–	програмне забезпечення
ПК	–	персональний комп’ютер

ВСТУП

Актуальність теми. Якби вам сказали назвати певні речі, які ви знайдете в парку, ви випадково згадали б такі речі, як трава, лавка, дерева тощо. Це дуже просте завдання, яке будь-яка людина може виконати миттєво. Однак існує дуже складний процес, який відбувається в глибині нашої свідомості.

Людський зір включає наші очі, але також включає все наше абстрактне розуміння понять і особистий досвід через мільйони взаємодій, які ми мали із зовнішнім світом. Донедавна комп'ютери мали дуже обмежені можливості самостійного мислення. Комп'ютерний зір – це нещодавня галузь технології, яка зосереджується на відтворенні людського зору, щоб допомогти комп'ютерам ідентифікувати й обробляти речі так само, як це роблять люди.

Завдяки нещодавнім розробкам у таких сферах, як штучний інтелект і обчислювальні можливості, сфера комп'ютерного зору досягла значного прогресу, щоб стати більш проникливою в повсякденне життя. Очікується, що до 2030 року ринок комп'ютерного зору наблизиться до 41,11 мільярда доларів, а середній річний темп зростання (CAGR) становитиме 16,0% між 2020 і 2030 роками.

У цьому зв'язку представляється доцільним проведення досліджень можливості використання моделей висхідної уваги для знаходження області зображення, приблизно відповідному об'єкту, розробка методів пошуку зображень, механізм яких подібний до механізму сприйняття зображень людиною. При цьому можна чекати, що використання інформації, що витягається з даних областей, дозволить підвищити якість пошуку CBIR-систем у порівнянні із широко використовуваним пошуком по глобальних ознаках зображень. Відзначимо, що останнім часом моделі візуальної уваги привернули увагу ряду дослідників, у тому числі: O. Marques, L. M. Mayron, G. B. Borba, H. R. Gamba.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

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

– Огляд існуючих систем комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– Дослідження системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– Програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Об’єктом дослідження є процес комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Предметом дослідження є методи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Методи дослідження базуються на методах розпізнавання образів, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– Розроблено вітчизняний продукт комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2023, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №14.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для реалізації комп'ютерного зору для створення інтерактивних користувацьких інтерфейсів. Комп'ютерний зір – це одна з областей штучного інтелекту, яка навчає комп'ютери та дозволяє їм розуміти візуальний світ. Комп'ютери можуть використовувати цифрові зображення та моделі глибокого навчання, щоб точно ідентифікувати та класифікувати об'єкти та реагувати на них.

Комп'ютерне бачення в штучному інтелекті спрямоване на розробку автоматизованих систем, які можуть інтерпретувати візуальні дані (наприклад, фотографії чи кінофільми) так само, як це роблять люди. Ідея, що лежить в основі комп'ютерного зору, полягає в тому, щоб навчити комп'ютерам інтерпретувати та розуміти зображення на піксельній основі. Це основа поля комп'ютерного зору. Щодо технічної сторони речей, комп'ютери намагатимуться отримувати візуальні дані, керувати ними та аналізувати результати за допомогою складних програм.

Обсяг даних, який ми створюємо сьогодні, величезний – 2,5 квінтильйона байтів даних щодня. Таке зростання даних виявилося одним із рушійних факторів розвитку комп'ютерного зору.

1.2 Область застосування

Для комп'ютерного зору потрібні величезні обсяги інформації. Повторні аналізи даних виконуються, доки система не зможе розрізняти об'єкти та ідентифікувати візуальні елементи. Глибоке навчання, специфічний вид машинного навчання, і згорткові нейронні мережі, важлива форма нейронної мережі, є двома ключовими техніками, які використовуються для досягнення цієї

мети. За допомогою попередньо запрограмованих алгоритмічних структур система машинного навчання може автоматично дізнаватися про інтерпретацію візуальних даних. Модель може навчитися розрізняти схожі зображення, якщо їй надати достатньо великий набір даних. Алгоритми дозволяють системі навчатися самостійно, щоб вона могла замінити людську працю в таких завданнях, як розпізнавання зображень.

Згорткові нейронні мережі допомагають зрозуміти моделі машинного та глибокого навчання, розділяючи візуальні елементи на менші частини, які можна позначати тегами. За допомогою тегів він виконує згортки, а потім використовує третинну функцію для надання рекомендацій щодо сцени, яку спостерігає. З кожним циклом нейронна мережа виконує згортки та оцінює правдивість своїх рекомендацій. І саме тоді він починає сприймати та ідентифікувати картинки, як людина. Комп'ютерний зір схожий на вирішення головоломки в реальному світі. Уявіть, що у вас є всі ці фрагменти мозаїки, і вам потрібно зібрати їх, щоб сформувати справжнє зображення. Саме так працюють нейронні мережі всередині комп'ютерного зору. Завдяки серії фільтрів і дій комп'ютери можуть об'єднати всі частини зображення, а потім думати самостійно. Однак комп'ютеру не просто дається головоломка із зображенням – скоріше, його часто годують тисячами зображень, які навчають його розпізнавати певні об'єкти.

Наприклад, замість того, щоб навчати комп'ютер шукати гострі вуха, довгі хвости, лапи та вуса, які складають кота, програмісти завантажують і передають у комп'ютер мільйони зображень котів. Це дозволяє комп'ютеру розуміти різні риси кота та миттєво розпізнавати його.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

За останні роки технологія комп'ютерного бачення швидко розвинулась і стала важливою технологією в різних галузях, таких як безпека, охорона здоров'я, сільське господарство, розумне місто, промислове виробництво, автомобільна промисловість тощо. Завдяки численним інструментам, платформам, фреймворкам і бібліотекам програмного забезпечення знайти найкращий інструмент для конкретного завдання комп'ютерного зору може бути важко.

У цьому розділі досліджуємо найпопулярніші інструменти комп'ютерного зору та їх використання, щоб допомогти вам прийняти обґрунтовані рішення під час вибору правильного інструменту для вашого проекту.

Viso Suite

Viso Suite – це наскрізне рішення для платформи комп'ютерного зору. Працюючи в галузі майже 10 років, ми натрапили на багато таких інструментів для створення комерційних систем комп'ютерного зору. У viso.ai використовується провідна безкодова платформа комп'ютерного зору Viso Suite, яку також включено до списку нижче.

Viso Suite: корпоративне безкодове рішення комп'ютерного зору.

Далі ми перерахуємо деякі з найпотужніших і популярних програмних інструментів комп'ютерного бачення для науковців з обробки даних, машинного навчання та команд розробників.

Список найпопулярніших засобів комп'ютерного зору у 2023 році:

– Інструмент №1: OpenCV.

– Інструмент №2: Viso Suite.

– Інструмент №3: TensorFlow.

– Інструмент №4: CUDA.

– Інструмент №5: MATLAB.

– Інструмент №6: Керас.

– Інструмент №7: SimpleCV.

– Інструмент №8: BoofCV.

– Інструмент №9: CAFFE.

– Інструмент №10: OpenVINO.

– Інструмент №11: DeepFace.

– Інструмент №12: YOLO.

1. OpenCV – бібліотека комп'ютерного зору в реальному часі

OpenCV – це бібліотека машинного навчання та комп'ютерного бачення з відкритим кодом. Створений з метою забезпечення загальної інфраструктури для програм комп'ютерного зору, OpenCV забезпечує доступ до понад 2500 класичних і найсучасніших алгоритмів. Ці алгоритми корисні для кількох завдань, зокрема для виявлення та розпізнавання обличчя, усунення ефекту червоних очей, ідентифікації об'єктів, вилучення 3D-моделей об'єктів, відстеження рухомих об'єктів і з'єднання кількох кадрів у зображення з високою роздільною здатністю.

Надзвичайно популярний інструмент обробки зображень OpenCV OpenCV має кілька інтерфейсів, таких як C++, Python, Java і MATLAB, і підтримує більшість операційних систем, включаючи Windows, Android, Linux і Mac. Бібліотека комп'ютерного зору широко використовується міжнародними компаніями, зокрема Google, Facebook, IBM, Toyota, Sony, Honda та Microsoft.

Переваги:

– Де-факто стандартний інструмент для обробки зображень.

– Користування є безкоштовним і є відкритим кодом.

– Велика підтримка громади.

– Пропонує доступ до понад 2500 алгоритмів.

– Дозволяє налаштувати код для певних цілей.

Недоліки:

– Він не такий простий у використанні, як інші інструменти, такі як MATLAB.

– Досить крута крива навчання.

2. Viso Suite – безкодова платформа комп'ютерного зору

Viso Suite – це наскрізна платформа комп'ютерного бачення для компаній, що дозволяє створювати, розгортати та контролювати реальні програми комп'ютерного бачення. Платформа без коду базується на найкращому у своєму класі програмному стеку для комп'ютерного зору, включаючи CVAT, OpenCV, OpenVINO, TensorFlow або PyTorch. Viso Suite включає в себе понад 15 продуктів в одному рішенні, включаючи анотації зображень, навчання моделям, керування моделями, розробку додатків без коду, керування пристроями, зв'язок IoT і спеціальні інформаційні панелі. Підприємства та урядові організації в усьому світі використовують Viso Suite для створення та керування портфоліо програм комп'ютерного зору (для промислової автоматизації, візуального контролю, віддаленого моніторингу тощо). Архітектура, керована моделлю, забезпечує надійну та безпечну інфраструктуру для створення конвеєрів комп'ютерного бачення за допомогою будівельних блоків. Модульна архітектура дозволяє використовувати будь-яку камеру (CCTV, IP, USB тощо), будь-яке обчислювальне обладнання (CPU, GPU, VPU, TPU тощо) або структуру ML. Висока гнучкість дозволяє легко додавати спеціальний код або інтегруватися з Tableau, PowerBI, SAP або зовнішніми базами даних (AWS S3, MongoDB тощо).

Переваги:

– Наскрізна платформа для створення та доставки всіх програм комп'ютерного зору за допомогою одного рішення.

– Немає коду комп'ютерного бачення, щоб будувати конвеєри комп'ютерного бачення набагато швидше.

– Кросплатформенність: створіть один раз, розгорніть де завгодно – використовуючи будь-яку камеру чи модель ШІ.

– Повне керування пристроями, безпечне розгортання на великій кількості пристроїв.

– Підходить для професійних команд, експертів із бачення та новоспечених розробників.

– Безпека корпоративного рівня, безпека довіри, керування доступом і повна конфіденційність даних..

Недоліки:

– Корпоративна платформа не має безкоштовного плану. Viso Suite, безкодова платформа комп'ютерного зору.

3. TensorFlow – бібліотека програмного забезпечення для машинного навчання

TensorFlow є однією з найпопулярніших наскрізних платформ машинного навчання з відкритим кодом із повним набором інструментів, ресурсів і бібліотек. TensorFlow особливо корисний для створення та розгортання програм, пов'язаних із комп'ютерним зором, які працюють на основі машинного навчання. TensorFlow – один із найпростіших інструментів комп'ютерного зору, який дозволяє користувачам розробляти пов'язані з комп'ютерним зором моделі машинного навчання для таких завдань, як розпізнавання облич, класифікація зображень, виявлення об'єктів тощо. Tensorflow, як і OpenCV, також підтримує різні мови, такі як Python, C, C++, Java або JavaScript.

Глибоке навчання з TensorFlow Для реальних проектів комп'ютерного бачення TensorFlow Lite є легкою реалізацією для машинного навчання на пристрої з периферійними пристроями. Як частина TensorFlow, TF Lite значно прискорює впровадження периферійного машинного навчання зі зменшеним розміром моделі та високою точністю при значно вищій ефективності, що дає змогу запускати машинне навчання всюди.

Переваги:

– Це платформа з відкритим кодом.

– Платформа сумісна з кількома мовами.

– Він забезпечує постійні оновлення для додаткових функцій і вдосконалень Потужні функції та хороша продуктивність.

Недоліки:

– Це надзвичайно ресурсомісткий інструментарій

4. CUDA – Паралельні обчислення та програмування

CUDA (скорочення від Compute Unified Device Architecture) – це паралельна обчислювальна платформа та модель інтерфейсу прикладного програмування (API), розроблена NVIDIA. Це дозволяє розробникам використовувати потужність графічних процесорів (Graphics Processing Units), щоб пришвидшити інтенсивну обробку програм. Набір інструментів включає в себе бібліотеку NVIDIA Performance Primitives (NPP), яка забезпечує функції обробки зображень, відео та обробки сигналів із прискоренням GPU для багатьох доменів, включаючи комп'ютерне бачення. Крім того, архітектура CUDA корисна для широкого кола завдань, таких як розпізнавання облич, маніпулювання зображеннями, відтворення 3D-графіки та інших. Обробка зображень у режимі реального часу за допомогою Nvidia CUDA підтримується для реалізацій Edge AI, щоб виконувати висновки AI на пристрої на периферійних пристроях, таких як Jetson TX2. Він підтримує різні мови програмування, включаючи C, C++, Python, Fortran або MATLAB, а також сумісний з більшістю операційних систем.

Переваги:

– Бібліотека NPP містить понад 5000 примітивів для обробки зображень і сигналів.

– Він включає підтримку кількох мов.

– Це швидко та ефективно.

– Потужний, високопродуктивний аналіз відео.

Недоліки:

– Його енергоспоживання досить високе Обмежена міжплатформна гнучкість

5. MATLAB – Платформа програмування для інженерів та вчених

MATLAB – це платформа програмування, яка корисна для низки різних програм, таких як машинне навчання, глибоке навчання та обробка зображень, відео та сигналів. Він постачається з набором інструментів комп'ютерного зору, який містить численні функції, програми та алгоритми, які допоможуть вам розробити рішення для завдань, пов'язаних із комп'ютерним зором.

Переваги:

– Його легко використовувати та вивчати; на MATLAB є багато безкоштовних ресурсів.

– Оскільки це мова програмування, писати код легше MATLAB дозволяє дуже швидко кодувати, чудово підходить для швидкого прототипування.

– Він має зручний автоматичний процес налагодження.

– Вважається найкращим інструментом для дослідників.

– Усі функції дуже добре задокументовані.

Недоліки:

– Інструмент не є безкоштовним для використання.

– Це досить повільно для багатьох завдань.

– Непросто інтегрувати зі сторонніми інструментами

6. Keras – API глибокого навчання Python

Keras – це бібліотека програмного забезпечення з відкритим кодом на основі Python, яка діє як інтерфейс для платформи машинного навчання TensorFlow. Він особливо підходить для початківців, оскільки дозволяє швидко побудувати модель нейронної мережі, забезпечуючи підтримку серверної частини.

Переваги:

– Проста у використанні бібліотека Python, зручна та швидка.

– Забезпечує підтримку кількох серверів.

– Він має велику підтримку спільноти.

Недоліки:

– Функції можна покращити.

– Налагодження може бути дещо складним

7. SimpleCV – фреймворк з відкритим вихідним кодом для машинного зору

SimpleCV – це колекція бібліотек і програмного забезпечення з відкритим вихідним кодом, яка дозволяє легко розробляти програми машинного зору. Завдяки його інфраструктурі ви отримуєте доступ до кількох потужних бібліотек комп'ютерного бачення, таких як OpenCV, без необхідності володіти глибокими знаннями про такі складні поняття, як бітова глибина, колірний простір, керування буфером або формати файлів. SimpleCV написано мовою Python і сумісно з кількома операційними системами, такими як Mac, Windows і Linux.

Переваги:

– Його можна використовувати безкоштовно.

– Більшість алгоритмів значною мірою оптимізовано.

– Включає хорошу документацію.

Недоліки:

– Він не підтримує жодних мов програмування, крім Python

8. BoofCV – Бібліотека комп'ютерного бачення для програм реального часу

BoofCV – це програмне забезпечення комп'ютерного бачення на основі Java, яке спеціально розроблено для рішень комп'ютерного бачення в режимі реального часу. Він є відкритим вихідним кодом і випущений під ліцензією Apache 2.0, що робить його вільним для використання в академічних і комерційних цілях. Це повна бібліотека з усіма основними та розширеними функціями, які можуть знадобитися для розробки програми комп'ютерного зору.

Переваги:

– Він має зручний інтерфейс Забезпечує підтримку кількох мов.

Недоліки:

– Повільніше в операціях низького рівня

9. CAFFE – швидка відкрита платформа для глибокого навчання

CAFFE або Convolutional Architecture for Fast Feature Embedding – це структура глибокого навчання та комп'ютерного зору, розроблена в Каліфорнійському університеті в Берклі. Ця структура написана мовою програмування C++ і підтримує кілька архітектур глибокого навчання, пов'язаних із класифікацією та сегментацією зображень. Це особливо корисно для дослідницьких цілей і промислового впровадження завдяки його чудовій швидкості та можливостям обробки зображень.

Переваги:

– Це відкритий код.

– Швидкий і простий у використанні.

– Підтримує кілька мов.

Недоліки:

– Документація може бути вдосконалена.

– Забезпечує лише часткову підтримку навчання з кількома GPU

10. OpenVINO – безкоштовний інструментарій для моделей глибокого навчання на обладнанні Intel

OpenVINO (Open Visual Inference and Neural Network Optimization) – це набір комплексних інструментів комп'ютерного зору, корисних для розробки додатків, що емулюють людський зір. Розроблений Intel, це безкоштовний кросплатформний інструментарій. Набір інструментів OpenVINO містить моделі для кількох завдань, таких як виявлення об'єктів, розпізнавання облич, розфарбовування, розпізнавання рухів тощо. Щоб дізнатися більше про цей інструмент, рекомендую прочитати статтю Що таке OpenVINO? Остаточний огляд.

Переваги:

Це безкоштовний і ефективний інструментарій.

– Підтримує кілька фреймворків глибокого навчання.

– Він сумісний з операційними системами Windows, Mac і Linux.

– Екосистема, що швидко розвивається, хороша апаратна підтримка.

Недоліки:

– Лише кілька прикладів використання Python

11. DeepFace – безкоштовна бібліотека глибокого навчання для розпізнавання облич

DeepFace наразі є найпопулярнішою відкритою бібліотекою комп'ютерного зору для розпізнавання обличчя з глибоким навчанням. Бібліотека пропонує простий спосіб виконати комп'ютерне бачення на основі розпізнавання обличчя за допомогою Python.

DeepFace – інструмент для розпізнавання облич і аналізу емоцій Якщо ви шукаєте засоби обробки зображень для розпізнавання обличчя, верифікації обличчя або аналізу атрибутів обличчя в режимі реального часу, DeepFace – чудовий спосіб використовувати найефективніші моделі розпізнавання глибокого навчання (Google FaceNet, VGG -Face, OpenFace, Facebook DeepFace, і більше).

Переваги:

– Це безкоштовно та з відкритим кодом, навіть для комерційного використання.

– Легкий і простий в установці.

– Підтримує популярні моделі та детектори.

– Оптимізовано для виконання висновків на пристрої в реальному часі (Edge AI).

Недоліки:

– Хмарний API недоступний.

12. YOLO – Виявлення об’єктів у реальному часі

You Only Look Once, або YOLOv7, є одним із найшвидших інструментів комп’ютерного зору, який ви можете вибрати в 2023 році. Розроблений Джозефом Редмоном і Алі Фархаді в 2016 році, він був спеціально створений для виявлення об’єктів у реальному часі. Швидший за всі інші інструменти виявлення об’єктів, YOLO зобов’язаний своєю швидкістю застосування нейронної мережі до повного зображення, яка потім розбиває зображення на сітки. Потім програмне забезпечення одночасно прогнозує ймовірності кожної сітки. Після надзвичайно популярних YOLOv3 і YOLOv4, YOLOR досяг найкращої продуктивності, поки його не перевершив YOLOv7, випущений у 2022 році.

Переваги:

– Це виключно швидко.

– Інструмент дуже точний, з мінімальними фоновими помилками.

– Алгоритм має першокласні можливості навчання.

Недоліки:

– Він не настільки ефективний у виявленні дрібних об'єктів.

– Існує обмежена підтримка громади.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка додатків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки

класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні додатки, а також додатки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої додатки в найкоротший термін. Як уже згадувався, MFC – це базовий набір

(бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-додатки на базі об'єктно-орієнтованого підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

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

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) 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 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 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

Вивчення алгоритмів, на яких базується сучасна технологія комп'ютерного зору, має важливе значення для розуміння її розвитку. Глибоке навчання – це різновид машинного навчання, яке сучасне комп'ютерне бачення використовує для отримання інформації на основі даних.

Коли справа доходить до комп'ютерного зору, глибоке навчання – це шлях. Використовується алгоритм, відомий як нейронна мережа. Шаблони в даних витягуються за допомогою нейронних мереж. Алгоритми базуються на наших поточних знаннях про структуру та функціонування мозку, зокрема про зв'язки між нейронами в корі головного мозку.

Перцептрон, математична модель біологічного нейрона, є фундаментальною одиницею нейронної мережі. Можна мати багато шарів пов'язаних персептронів, схожих на шари нейронів у біологічній корі головного мозку. Коли необроблені дані надходять у згенеровану перцептроном мережу, вони поступово перетворюються на прогнози.

Скільки часу потрібно, щоб розшифрувати зображення

Надзвичайно швидкі процесори та відповідна технологія разом із швидким, надійним Інтернетом і хмарними інфраструктурами роблять весь процес надзвичайно швидким у наш час. Важливо, що кілька найбільших компаній, які інвестують у дослідження штучного інтелекту, як-от Google, Facebook, Microsoft і IIBM, відкрито повідомляють про свої дослідження та розробки в цій галузі. Таким чином люди можуть спиратися на фундамент, який вони заклали.

Це призвело до нагрівання сектору штучного інтелекту, і дослідження, які раніше тривали тижнями, тепер можуть бути завершені за кілька хвилин. Крім

того, для багатьох завдань комп’ютерного зору в реальному світі весь цей процес відбувається постійно за лічені мікросекунди. У результаті комп’ютер наразі може досягти статусу, який дослідники називають «свідомим обставинами».

Програми комп'ютерного зору

Однією з сфер машинного навчання, де фундаментальні ідеї вже включені в основні продукти, є комп’ютерне бачення. Додатки включають:

– Безпілотні автомобілі. За допомогою комп’ютерного зору автономні транспортні засоби можуть розуміти навколишнє середовище. Кілька камер записують навколишнє середовище, яке потім надсилається в алгоритми комп’ютерного зору, які аналізують фотографії в ідеальній синхронізації, щоб визначити місцезнаходження країв дороги, розшифрувати покажчики та побачити інші транспортні засоби, перешкоди та людей. Тоді автономний транспортний засіб може самостійно пересуватися вулицями та шосе, об’їжджати перешкоди та безпечно доставляти своїх пасажирів туди, куди їм потрібно.

– Розпізнавання обличчя. Програми розпізнавання облич, які використовують комп’ютерний зір для розпізнавання людей на фотографіях, значною мірою покладаються на цю область дослідження. Риси обличчя на фотографіях ідентифікуються за допомогою алгоритмів комп’ютерного зору, які потім зіставляють ці аспекти зі збереженими профілями обличчя. Щоб підтвердити особу людей, які використовують споживчу електроніку, все частіше використовується розпізнавання обличчя. Розпізнавання облич використовується в програмах соціальних мереж як для виявлення користувачів, так і для позначення користувачів. З тієї ж причини правоохоронні органи використовують програмне забезпечення для розпізнавання облич, щоб вистежувати злочинців за допомогою записів із камер спостереження.

– Доповнена та змішана реальність. Доповнена реальність, яка дозволяє таким комп’ютерам, як смартфони та носимим технологіям, накладати або вбудовувати цифровий вміст у реальне середовище, також значною мірою залежить від комп’ютерного зору. Віртуальні елементи можуть бути розміщені в

реальному середовищі за допомогою комп’ютерного зору в обладнанні доповненої реальності. Щоб правильно генерувати глибину та пропорції та розташовувати віртуальні елементи в реальному середовищі, програми доповненої реальності покладаються на методи комп’ютерного зору для розпізнавання поверхонь, таких як стільниці, стелі та підлоги.

– Охорона здоров'я. Комп’ютерний зір зробив значний внесок у розвиток медичних технологій. Автоматизація процесу пошуку злоякісних родимок на шкірі людини або визначення місцезнаходження індикаторів на рентгенівському знімку або МРТ-скануванні є лише одним із багатьох застосувань алгоритмів комп’ютерного зору.

Нижче наведено кілька прикладів добре налагоджених дій із використанням комп’ютерного зору:

– Категоризація зображень. Комп’ютерна програма, яка використовує зображення за категоріями, може визначити, що це зображення (собака, банан, обличчя людини тощо). Зокрема, можна з упевненістю стверджувати, що вхідне зображення відповідає певній категорії. Його може використовувати платформа соціальних мереж, наприклад, для фільтрації образливих фотографій, які публікують люди.

– Виявлення об'єктів. Спочатку класифікуючи зображення за категоріями, виявлення об’єктів може потім використовувати цю інформацію для пошуку та каталогізації екземплярів потрібного класу зображень. У обробній промисловості це може включати пошук дефектів на виробничій лінії або визначення місцезнаходження зламаного обладнання.

– Спостереження за рухомими об'єктами. Якщо предмет буде виявлено, відстеження об’єкта продовжуватиме рухатися в тому самому місці. Загальний спосіб зробити це – використання прямого відеопотоку або серії фотографій, зроблених послідовно. Наприклад, безпілотні автомобілі мають не лише ідентифікувати та класифікувати рухомі речі, як-от люди, інші автомобілісти та

дорожні системи, щоб запобігти аваріям і дотримуватися правил дорожнього руху.

– Отримання зображень на основі їхнього вмісту. На відміну від традиційних методів візуального пошуку, які покладаються на мітки метаданих, система розпізнавання на основі вмісту використовує комп'ютерне зір для пошуку, дослідження та отримання зображень із величезних сховищ даних на основі фактичного вмісту зображення. Для цієї роботи можна використовувати автоматичні анотації до зображень, які можуть замінити традиційне візуальне тегування.

Алгоритми комп'ютерного зору

Алгоритми комп'ютерного зору включають різні методи, які використовуються для розуміння об'єктів у цифрових зображеннях і вилучення високорозмірних даних із реального світу для створення числової чи символьної інформації. Існує багато інших алгоритмів комп'ютерного зору, задіяних у розпізнаванні речей на фотографіях. Деякі поширені:

– Класифікація об'єктів. Яка основна категорія об'єктів, присутніх на цій фотографії?

– Ідентифікація об'єкта – який тип об'єкта присутній на цій фотографії?

– Виявлення об'єкта – де знаходиться об'єкт на фотографії?

– Сегментація об'єкта – які пікселі належать об'єкту на зображенні?

– Перевірка об'єкта – чи є об'єкт на фотографії?

– Розпізнавання об'єктів – які об'єкти зображені на цій фотографії та де вони розташовані?

– Виявлення орієнтирів об'єкта – які ключові моменти об'єкта на цій фотографії?

Поряд з алгоритмами глибокого навчання можна вивчати багато інших передових алгоритмів комп'ютерного зору, таких як передача стилів, розфарбовування, оцінка пози людини, розпізнавання дій тощо.

Проблеми комп'ютерного зору

Створити машину з людським баченням напрочуд складно, і не лише через технічні труднощі, пов’язані з комп’ютерами. Нам ще потрібно багато чого дізнатися про природу людського зору.

Щоб повністю зрозуміти біологічний зір, потрібно дізнатися не лише про те, як працюють різні рецептори, як-от око, а й про те, як мозок обробляє те, що він бачить. Процес було намічено, його хитрощі та короткі шляхи виявлено, але, як і в будь-якому дослідженні мозку, ще потрібно подолати значну відстань.

Переваги комп’ютерного зору

Комп’ютерний зір може автоматизувати кілька завдань без втручання людини. У результаті це надає організаціям ряд переваг:

– Швидший і простіший процес – системи комп’ютерного зору можуть швидше виконувати повторювані та монотонні завдання, що спрощує роботу для людей.

– Кращі продукти та послуги – добре навчені системи комп’ютерного зору не допускатимуть помилок. Це призведе до швидшої доставки високоякісних продуктів і послуг.

– Зменшення витрат – компаніям не потрібно витрачати гроші на виправлення недоліків процесів, тому що комп’ютерне бачення не залишить місця для несправних продуктів і послуг.

Недоліки комп'ютерного зору

Не існує жодної технології, яка б не мала недоліків, як це стосується систем комп’ютерного зору. Ось кілька обмежень комп’ютерного зору:

– Брак спеціалістів. Компанії потребують команди висококваліфікованих професіоналів із глибокими знаннями про відмінності між технологіями ШІ та машинного навчання та технологіями глибокого навчання для навчання систем комп’ютерного зору. Існує потреба в більшій кількості спеціалістів, які могли б допомогти сформувати майбутнє технологій.

– Необхідність регулярного моніторингу. Якщо система комп'ютерного зору стикається з технічним збоєм або виходить з ладу, це може завдати величезних збитків компаніям. Отже, компанії повинні мати спеціальну команду для моніторингу та оцінки цих систем.

3.2 Розробка структурної схеми

У цій роботі представлено систему пошуку зображень на основі вмісту (CBIR). Пошук зображень на основі вмісту (CBIR), також відомий як Query By Image Content (QBIC), представляє технології, що дозволяють упорядковувати цифрові зображення за їхніми візуальними характеристиками. Вони засновані на застосуванні методів комп'ютерного зору до проблеми пошуку зображень у великих базах даних. Отримання зображень на основі вмісту (CBIR) складається з отримання найбільш візуально подібних зображень до даного зображення запиту з бази даних зображень.

В результаті проведеного аналізу сучасного стану проблеми пошуку зображень на основі змісту, у ході якого були розглянуті наступні CBIR-системи, що перебувають у вільному доступі, а також описана архітектура CBIR-систем і інформаційні ознаки зображень, використовувані в CBIR-системах і існуючі класифікації ознак, було виконано оцінку якості пошуку в CBIR-системах. Як критерій якості пошуку була обрана точність на рівні 20 перших знайдених зображень (P_{20}) – кількість зображень, релевантних запиту, серед перших 20-ти, виданих системою пошуку. Даний критерій дозволяє оцінити інформативність першої сторінки результатів практично для всіх розглянутих систем. Виявилося, що для більшої частини CBIR-систем $P_{20} < 16\%$. Для системи img(Anaktisi) P_{20} – 45%, однак настільки високий результат пошуку складно вважати об'єктивним – у базі зображень, по якій ведеться пошук цією системою, утримується велика кількість дублікатів зображень. Результати пошуку у випадку, якщо зображення-запит не має дубліката, істотно гірше. Якість пошуку зображень на основі

текстової інформації в системах Altavista і Yandex виявилося вище, ніж в CBIR-системах (P_{20} – 39% і P_{20} – 51%, відповідно). Таким чином, виявлено, що сучасні CBIR-системи мають недостатньо високу якість пошуку.

Показано, що одна із причин недостатньо високої якості пошуку полягає в тому, що в більшості CBIR-систем запит задається у вигляді зображення-зразка, що вибирається з наявних у базі зображень. Недолік подібної форми запиту полягає в тому, що користувач задає ціле зображення без вказівки його області, що цікавить. Тому в ряді випадків система знаходить зображення, які в цілому схожі на шукане, але найчастіше не містять шуканої інформації. Таким чином, один з можливих підходів підвищення якості пошуку зображень складається в наданні користувачеві можливості робити запит не у вигляді повного зображення, а у вигляді обраної конкретної області на зображенні-запиті.

Далі проведемо дослідження алгоритмів сегментації зображень і критеріїв кількісної оцінки якості результатів сегментації, а також виконане порівняння декількох алгоритмів сегментації зображень.

Сегментація зображення – це процес поділу зображення на безліч непересічних областей, однорідних з урахуванням обраних характеристик зображення. Об'єднання даних областей дає вихідне зображення. При цьому виділені області можуть грубо відповідати об'єктам, частинам об'єктів або групам об'єктів, наявних на зображенні. Незважаючи на те, що розглянута процедура не приводить до ідентифікації візуально спостережуваних об'єктів (оскільки необхідно відділення областей тла від областей, що містять об'єкти), сегментація зображення є невід'ємним етапом пошуку зображень за змістом.

У роботі проведене дослідження супервізорних критеріїв, використовуваних для кількісної оцінки якості сегментації, засновані на обчисленні міри відмінності результатів сегментації від дійсної форми областей зображень. При цьому дійсна форма областей задається експертами (як у базі зображень університету Берклі (рис. 3.1)) або вважається відомою на штучно згенерованих зображеннях із заздалегідь заданими геометричними формами.

Рисунок 3.1 – Приклади сегментаций, виконаних людиною, для зображень із бази університету Берклі

Проведені дослідження супервізорних критеріїв якості сегментації дозволили одержати:

1. Оцінки чутливості критеріїв до результатів сегментації, виконуваної різними експертами.

2. Оцінки відповідності значення супервізорних критеріїв візуальній оцінці результатів сегментації, що виставляється експертом (при візуальній класифікації результатів сегментації).

3. Оцінка відповідності значення супервізорних критеріїв візуальній оцінці, що виставляється експертами (при формальному оцінюванні результатів сегментації по кількості виділених областей).

На основі результатів кількісних досліджень супервізорних критеріїв оцінки якості сегментації зображень виділені чотири критерії (Dku, GCE, RI, RMS), що забезпечують найбільш об'єктивну оцінку якості сегментації. Дані критерії були використані далі для порівняльного аналізу алгоритмів сегментації зображень.

Проведено порівняння наступних алгоритмів сегментації: алгоритм еволюції кривої на основі моделі геодезичних активних контурів (Geodesic Active Contours), алгоритм еволюції кривої на основі потоку вектора градієнта (Gradient vector flow), алгоритм еволюції кривої під управлінням потоку границь (Edgeflow-driven Curve Evaluation), алгоритм анізотропної дифузії під управлінням потоку границь (Edgeflow-driven Anisotropic Diffusion), алгоритм

анізотропної дифузії, запропонований Перону й Маліком (Perona Malik Flow), алгоритм анізотропної дифузії, запропонований Шапіро (Self– Snakes), алгоритм JSEG сегментації зображень із обліком колірних і текстурних ознак зображення, оператори країв Кенні, Робертса, Превітт, Собела, Zerocross, Log з додатковою обробкою по зв'язуванню границь. Порівняння алгоритмів виконувалося на базі зображень університету Берклі (Berkeley Segmentation Dataset).

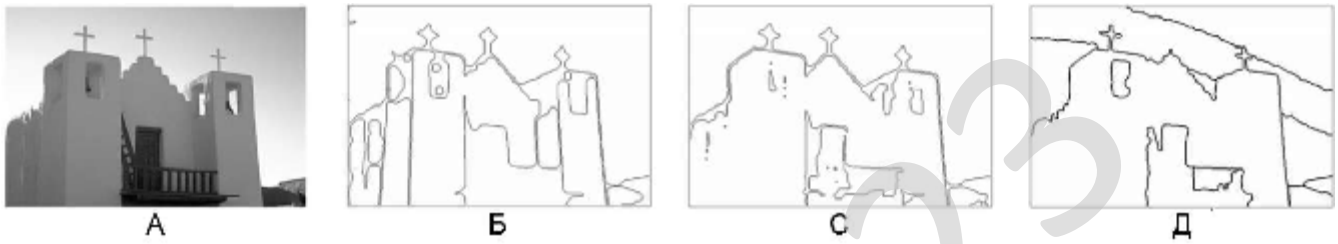

Рисунок 3.2 – Приклад зображення (А), результати сегментації: Canny (Б), Roberts (З), JSEG (Д)

Відповідно до обраних критеріїв Dku, GCE, RI, RMS кращі результати були отримані в алгоритмів JSEG, Canny, Roberts. Візуальний аналіз результатів сегментації (рис. 3.2), отриманих за допомогою даних алгоритмів сегментації, показав, що, на відміну від JSEG, області, виділені за допомогою алгоритмів Canny і Roberts, містять велика кількість дрібних отворів, для видалення яких необхідна додаткова обробка результатів сегментації. Тому далі в роботі використовувався алгоритм JSEG.

Далі проведемо дослідження ступеня суб'єктивності уваги людини, розглянуті відомі моделі висхідної уваги й описаний розроблений авторами алгоритм знаходження прото-об’єкту.

Увага – це вибірний процес, що дозволяє зоровій системі відокремлювати релевантні зовнішні подразники від нерелевантних. З його допомогою із зорового поля відбираються сегменти зображення для більше детальної подальшої переробки. Відомі два базових підходи до аналізу уваги: на основі висхідних

процесів (bottom-up image-based) і на основі спадних процесів (top-down task-dependent). Підхід, заснований на висхідних процесах, базується на тому, що розподіл уваги повністю визначається властивостями образа (наприклад, несподіваний рух на периферії зорового поля, відмінність кольору образа від тла). При цьому рішення приймається без обліку свідомості людини. Зорова система людини, навпроти, функціонує за принципом висхідного процесу – створення образа стає результатом об'єднання базових елементів, виявлених зоровою системою. Підхід, заснований на спадних процесах, переважно базується на знаннях, раніше отриманих спостерігачем, його попередньому досвіді, осмисленні й інтерпретації, а також на його очікуваннях. Процеси, що лежать в основі уваги, можуть бути складовою частиною як висхідних, так і спадних процесів. У літературі вказується, що визначальну роль в увазі грають висхідні процеси. Тому далі в нашій роботі виконане дослідження можливості застосування моделей висхідної уваги для знаходження області зображення, приблизно відповідному об'єкту.

Для оцінки можливості використання методів теорії візуальної уваги проведені експериментальні дослідження, ціль яких складалася в пошуку відповіді на питання: «Наскільки стабільним виявляється суб'єктивна увага людини, що розглядає те або інше зображення?». При цьому була використана методика, що реалізується наступною послідовністю дій:

1. Вибір набору зображень I (не менш 50 зображень):

$$I = \{I_1, I_2, ..., I_i, ..., I_n\},$$

де n – кількість зображень у наборі, i ∈ (1, n).

2. Формування безлічі незалежних друг від друга експертів E (не менш 10):

$$E = \{E^1, E^2, ..., E^j, ...E^m\},$$

де m – кількість експертів, j ∈ (1, m), яким пропонувалося на кожному зображенні безлічі I знайти області, що залучають їхню увагу.

3. Формування експертами незалежно друг від друга для кожного зображення двовимірних бінарних масок, привласнюючи значенням яскравостей пікселів одиницю, коли піксель належить області, що привертає їхню увагу, і нуль – у протилежному випадку.

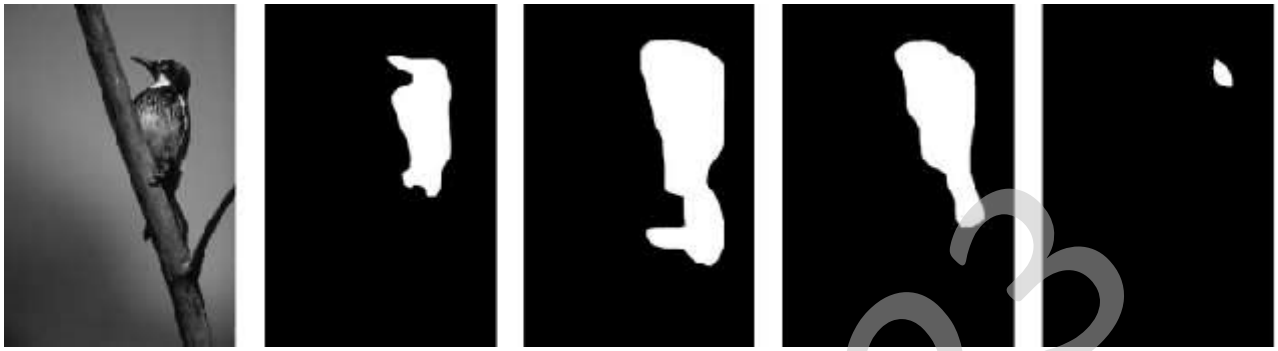

Рисунок 3.3 – Приклади бінарних масок, сформованих обраними експертами

Результатом виконання даного кроку є безліч бінарних масок:

$$\left\{E_i^j\right\},\ i=\overline{1,n},\ j=\overline{1,m}:$$

$$E^1=\left\{E_1^1,E_2^1,\ldots,E_i^1,\ldots,E_n^1\right\}$$
$$E^2=\left\{E_1^2,E_2^2,\ldots,E_i^2,\ldots,E_n^2\right\}$$
$$\ldots$$
$$E^j=\left\{E_1^j,E_2^j,\ldots,E_i^j,\ldots,E_n^j\right\}$$
$$\ldots$$
$$E^m=\left\{E_1^m,E_2^m,\ldots,E_i^m,\ldots,E_n^m\right\}$$

Приклади одного із зображень, запропонованого експертам, і відповідних бінарних масок представлені на рис. 3.3.

4. Накладення один на одного для кожного зображення I. бінарних масок E^1. і одержання напівтонового зображення, у якому яскравість кожного пікселя рівняється сумі яскравостей відповідних пікселів у бінарних масках.

$$Gray_i=\sum_j E_j^i.$$

Приклад напівтонового зображення Gray, представлений на рис. 3.4. Тут шкала, розташована ліворуч від рисунка, установлює відповідність між інтенсивністю сірого кольору й відсотком збігів між бінарними масками:

$$\left\{E_i^j\right\},\ \ i=\overline{1,n},\ \ j=\overline{1,m}\,.$$

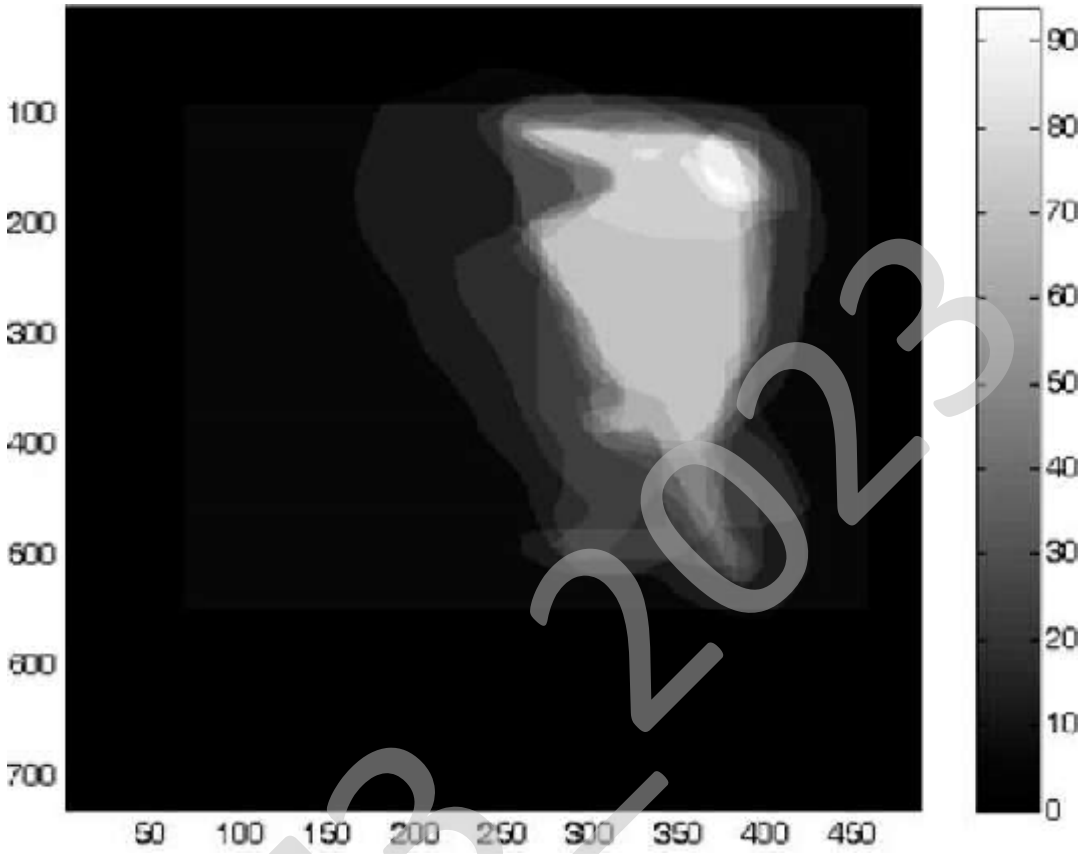

Рисунок 3.4 – Приклад напівтонового зображення $Gray_i$

5. Обчислення для кожного напівтонового зображення Gray, максимальної кількості погоджень (у відсотках від загального числа експертів):

$$Congruence_i = \frac{\max(Gray_i)\cdot 100}{m}.$$

6. Побудова гістограми й обчислення середнього відсотка погодженості:

$$Mean\_Congruence = \frac{\sum_i Congruence_i}{n}.$$

Для проведення дослідження суб'єктивності уваги людини відповідно до вищеописаної методики були сформовані 3 набори зображень (табл. 3.1):

1. Набір «Animals» – тварини. Практично на всіх зображеннях тільки один явно, що виділяється об'єкт;

2. Набір «Landscape» – пейзажі. Зібрано зображення природи. На перший погляд на таких зображеннях немає явно, що виділяється об'єкта;

3. Набір «People» – люди. Зображення із цього набору містять трохи, що виділяються на перший погляд об'єктів.

Таблиця 3.1 – Характеристики наборів зображень

№	Назва набору	Кількість зображень (n)	Кількість експертів (m)
1	«ANIMALS»	50	17
2	«LANDSCAPE»	55	15
3	«PEOPLE»	55	21

Середні значення рівня погодженості думок експертів по наборах «Animals», «Landscape», «People» представлені в табл. 3.2, з якої видно, що для кожного набору зображень рівень погодженості при визначенні області, що привертає увагу людини, досить великий (не нижче 80%).

Таблиця 3.2 – Середні значення рівня погодженості

№	Назва набору	Середній відсоток погодженості (Mean_Congruence)
1	«ANIMALS»	91.88 ± 8.14
2	«LANDSCAPE»	83.03 ± 12.69
3	«PEOPLE»	82.6 ± 10.1

Таким чином, результати проведеного експериментального дослідження виявляють високий рівень погодженості думок незалежних експертів (природно, суб'єктивного) при виділенні областей візуальної уваги. Отже, дану характеристику можна розглядати як якусь інформативну ознаку, по якому

можна встановлювати взаємно однозначна відповідність між зображенням і його змістом, і використовувати його для побудови алгоритмів пошуку зображень за змістом.

Відзначимо, що в літературі для позначення області зображення, що містить елементи, на яких концентрується увага людини, використовується поняття «фокус уваги». У більшості випадків фокус уваги визначається у вигляді окружності різного радіуса, тобто так само як і сегментація зображень не приводить до ідентифікації візуально спостережуваних об'єктів. У роботі Уалтера (Walther) використовується найбільш удале на наш погляд поняття «прото-об'єкт» – область зображення, що привертає увагу людини, що має довільну форму, що є грубим наближенням до спостережуваного об’єкту (або об'єктам).

Далі в роботі досліджені наступні моделі висхідної візуальної уваги: KU-модель, IKN-модель, HKP-модель, WK-модель, KB-модель, з яких, як показав Уалтер (Walther), найкращою моделлю серед всіх відомих є WK-модель. Однак, як показали отримані нами результати, WK-модель також не вільна від ряду недоліків:

1. Прото-об'єкт, який шукається, є лише грубим наближенням до об’єкту, що привертає увагу людини.

2. Оцінка алгоритму по базі природних зображень показала, що з його допомогою правильно знайдені прото-об'єкти тільки для 52% зображень.

3. На основі візуального аналізу знайдених прото-об'єктів зроблений вивід, що області, що відповідають прото-об'єктам невеликі в порівнянні з розміром присутніх на зображеннях об'єктів, а їхні границі виявляються дуже грубими (східчастими).

Приклади прото-об'єктів, знайдених за допомогою WK-моделі, показані на рис. 3.5.

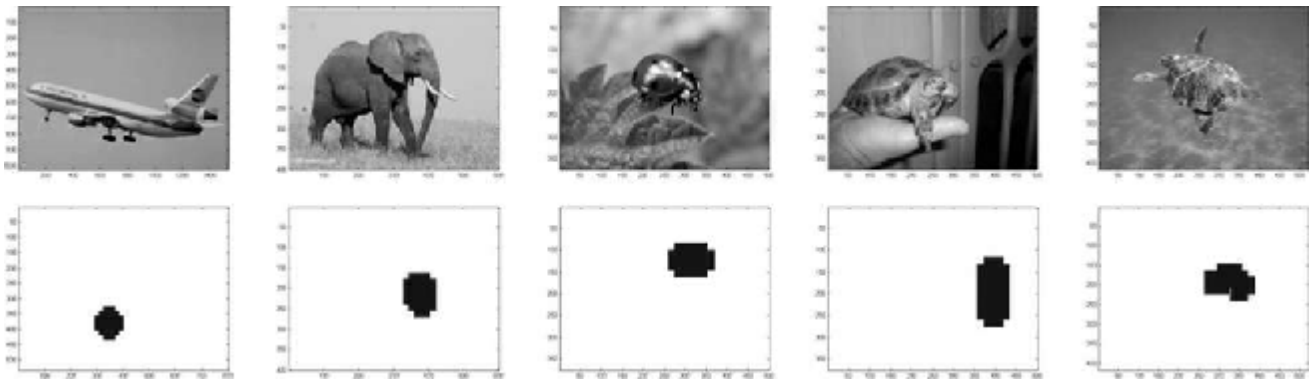

Рисунок 3.5 – Прото-об'єкти, знайдені за допомогою WK-моделі

Для усунення перерахованих вище недоліків WK-моделі, був розроблений алгоритм знаходження прото-об’єкту на кольорових природних зображеннях. Нехай $I(x)^{RGB}$ – вихідне зображення в просторі RGB, задане на поле зору X (x – піксель зображення, $x \in X$). НПО-алгоритм реалізується наступною послідовністю дій:

1. Створення ескізу зображення $\Psi(\chi)$. Під ескізом зображення розуміється зменшена до розміру 200 пікселів по більшій стороні зображення кольорова копія вихідного зображення. Зображення Ψ задано на поле зору ξ (χ – піксель зображення, $\chi \in \xi$).

2. Перетворення ескізу в простір LAB:

$$\Psi(\chi) \rightarrow \Psi(\chi)^{LAB}.$$

3. Знаходження фокуса уваги на кожному компоненті LAB:

– згладжування усереднюючим фільтром, область усереднення задавалася за замовчуванням рівної 3 x 3;

– застосування різницевого текстурного фільтра Local range of image. Значення кожного пікселя χ визначається різницею максимального й мінімального значення пікселів в околиці 3 x 3 цього пікселя. Одержуємо зображення $\Psi(\chi)$;

– гранична обробка:

$$\Psi(\chi)^{thr} = \begin{cases} 0, & \text{если } \Psi(\chi)^t = R \cdot \max_{i,j}(\chi)^t \\ \Psi(\chi)^t, & \text{иначе.} \end{cases},$$

де R = 0.75 – значення, обумовлене експериментально;

– перетворення Ψ^{thr} у бінарне зображення Ψ^{bin}

– видалення статистично незначущих пікселів χ, які ідентифікуються як викиди. Тут аналіз викидів виконується окремо по осях X і Y. Статистична значимість викидів оцінювалася за допомогою критерію Романовського. Пікселі визнані викидами хоча б по одній осі видалялися;

– обчислення вершин прямокутника, що обмежує отриману безліч білих пікселів. Отримана прямокутна область характеризує фокус уваги. Зображення F є бінарною маскою фокуса уваги. на якій значення пікселя дорівнює 1, якщо піксель належить знайденій прямокутній області (інакше 0).

Результатом виконання п. 3 є три знайдених фокуси уваги $F(\chi)^L$, $F(\chi)^A$, $F(\chi)^B$.

4. Обчислення узагальненого фокуса уваги F:

$$F(\chi) = F(\chi)^L + F(\chi)^A + F(\chi)^B.$$

5. Видалення пікселів домінантного кольору. Об'єкт на зображенні відрізняється від тла в тому числі й кольором (причому тло займає значну частину зображення), тому доцільно не відносити пікселі домінантного кольору до об'єкту:

$$\Phi(\chi) = \begin{cases} 0, & \text{если } C_{min} \le \text{Ind}(\chi) \le C_{max}; \\ F(\chi), & \text{иначе.} \end{cases},$$

де Ind(χ) – індексоване зображення, C_{min}, C_{max} – границі діапазону домінантного кольору, що перебуває в такий спосіб:

– перетворення Ψ (χ) в індексоване зображення Ind(χ);

– побудова гістограми для Ind(χ);

– визначення інтервалу, якому відповідає максимум гістограми:

$$[C_{min}, C_{max}];$$

6. Сегментація зображення ?(?) за допомогою алгоритму JSEG. Навмисно був обраний режим пересегментації (параметр -l установлювався рівним 10).

7. Вибір серед результатів сегментації областей, що належать прото-об’єкту. Область належить об'єкту, якщо в ній є присутнім не більше 30% пікселів $\chi = 0$ на зображенні $\Phi(\chi)$. Обрані області поєднуються й утворять прото-об'єкт.

8. Збільшення бінарної маски знайденого прото-об’єкту до розмірів вихідного зображення I(?).

Ілюстрація алгоритму знаходження прото-об’єкту представлена на рис. 3.6.

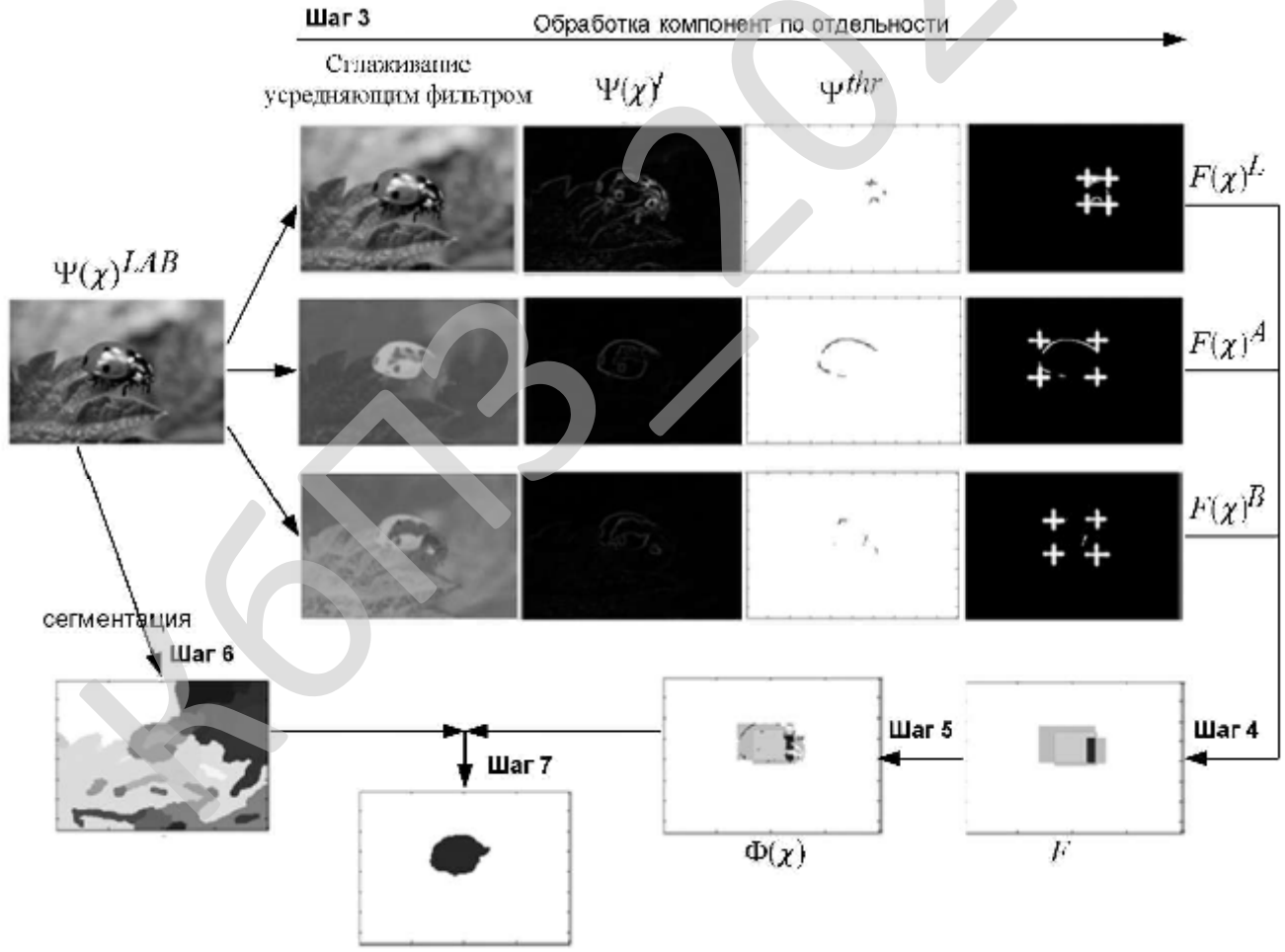

Рисунок 3.6 – Ілюстрація роботи алгоритму знаходження прото-об’єкту на різних етапах

Приклади прото-об'єктів, знайдених за допомогою алгоритму знаходження прото-об’єкту на інших зображеннях, представлені на рис. 3.7.

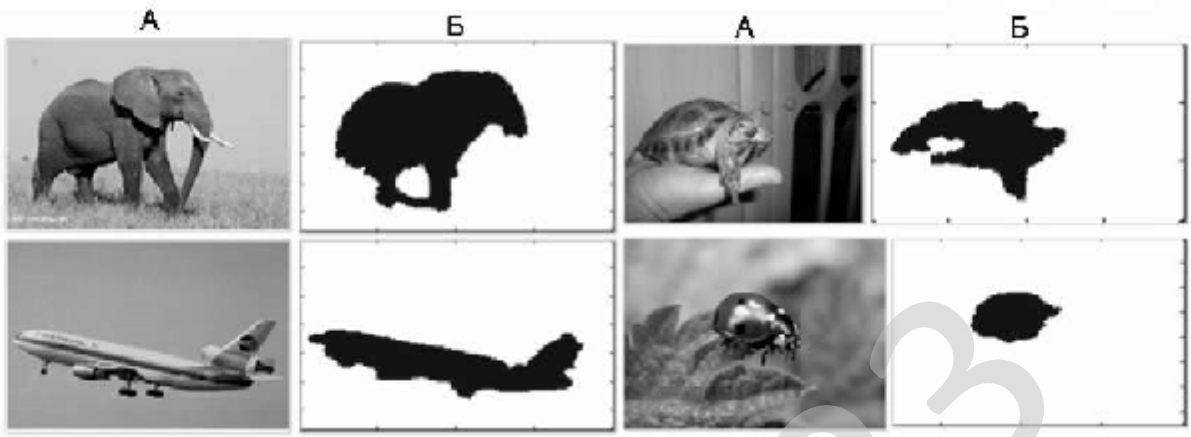

Рисунок 3.7 – Приклади прото-об'єктів, знайдених за допомогою алгоритму знаходження прото-об’єкту (А – вихідні кольорові зображення, Б – маски знайдених прото-об'єктів)

НПО-алгоритм застосуємо до наступних зображень:

1. Кольорові зображення. В алгоритмі JSEG використовується інформація про розподіл кольорів на зображенні.

2. Природні зображення (фотографії), для яких характерна присутність різноманітних текстур. У НПО-алгоритмі при знаходженні фокуса уваги використовується текстурний фільтр.

3. Зображення, що не містять складних сцен з більшою кількістю об'єктів. У НПО-алгоритмі закладене знаходження одного прото-об’єкту. Якщо на зображенні присутнє більше одного об’єкту, то алгоритм знаходить один прото-об'єкт, що відповідає одному з наявних об'єктів.

Виконано порівняння НПО– і WK-алгоритмів. Для об'єктивної оцінки результатів автоматичної локалізації об’єкту використовувалися наступні критерії:

1. Повнота (recall) – частка пікселів прото-об’єкту, що входять до складу об’єкту, від загального числа пікселів, що належать об'єкту.

2. Точність (precision) – частка пікселів прото-об’єкту, що входять до складу об’єкту, від загального числа пікселів, що належать прото-об'єкту:

Нехай є зображення I (x), де x – піксель зображення (x ∈ X).

Позначимо M бінарну маску об’єкту, знайдену експертом. Об'єкту на зображенні відповідає безліч:

$$O = \{x \in X | M > 0\}.$$

Позначимо M' бінарну маску прото-об’єкту, знайдену автоматично. Прото-об'єкту на зображенні відповідає безліч:

$$O' = \{x \in X | M' > 0\}.$$

Перетинанням безлічей O і O' є підмножина Cross, що втримується одночасно в O і O' для кожного x ∈ X:

$$Cross = O \cap O' = \{x | x \in O \wedge x \in O'\},$$

тоді повнота й точність обчислюються по формулах:

$$recall = \frac{|cross|}{|O|}, \quad precision = \frac{|cross|}{|O|}.$$

Проведено серію експериментів по дослідженню впливу наступних факторів на результативність локалізації об'єкта:

1. Положення об’єкту щодо центра природного зображення. Тестування виконувалося на колекції з 50 кольорових зображень. На кожному зображенні присутнє один добре виражений об'єкт, розташований не по центрі зображення.

2. Положення об’єкту щодо центра штучно згенерованого кольорового зображення. Тестування виконувалося на колекції 49 зображень розміром 200 x 200 пікселів. У відсотках від площі всього зображення середня площа об’єкту на зображенні становить S = 3.7 ± 0.1.

3. Невеликий розмір об’єкту на природному зображенні. Тестування виконувалося на колекції з 72 зображень, на кожному з яких є присутнім один

виражений об'єкт, розташований не по центрі. У відсотках від площі всього зображення середня площа об’єкту становить S = 6.2 ± 0.3.

4. Невеликий розмір об’єкту на штучно згенерованому кольоровому зображенні. Тестування виконувалося на колекції 50 зображень. У відсотках від площі всього зображення середня площа об’єкту становить S = 0.802 ± 0.008.

5. Наявність шумів на штучно згенерованому кольоровому зображенні. Тестова колекція складається з 49 зображень, на які доданий шум і об'єкти розташовані не по центрі. У відсотках від площі всього зображення середня площа об’єкту на зображенні становить:

$$\overline{S} = 3.7 \pm 0.1.$$

Результати експериментів представлені на рис. 3.8.

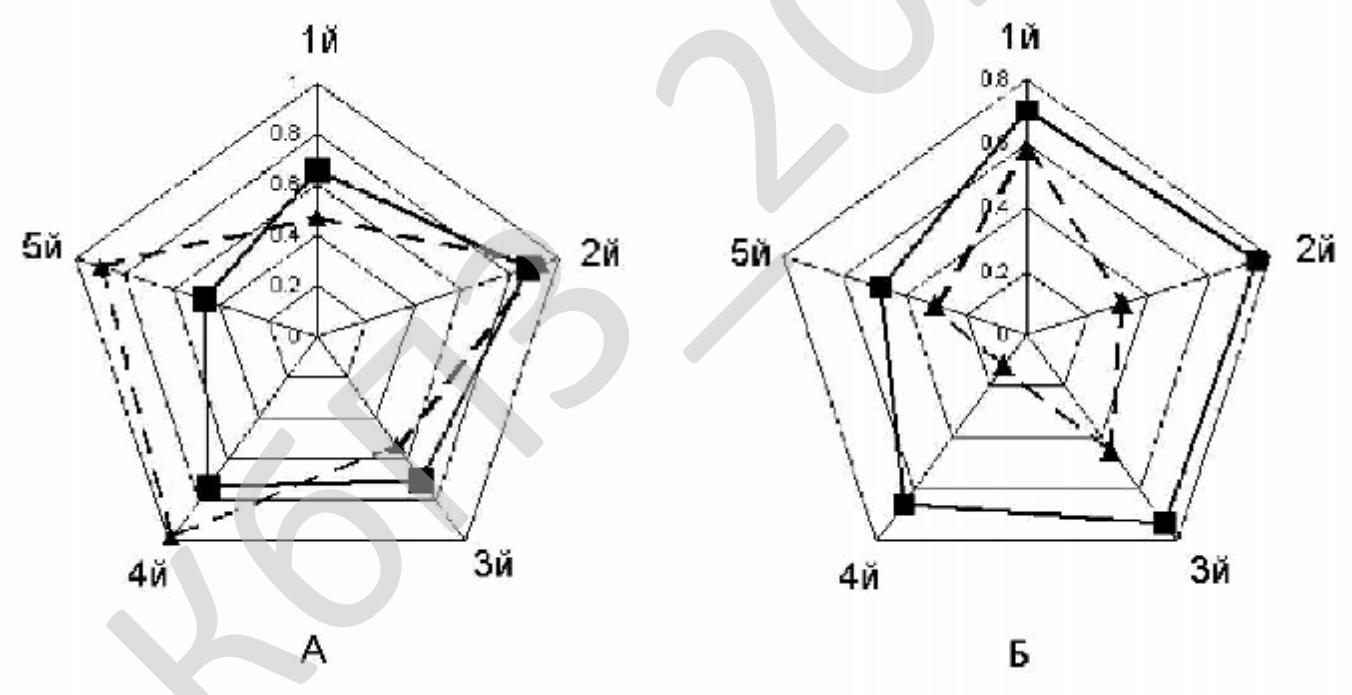

Рисунок 3.8 – Результати 5-ти експериментів у порівнянні НПО– і WK-алгоритмів. (А) – оцінки повноти **(recall),** (Б) – оцінки точності **(precision).** Суцільна лінія відповідає НПО-алгоритму, пунктирна – WK -алгоритму

З рис. 3.8(Б) видно, що у всіх експериментах значення критерію точності для НПО-алгоритму вище, ніж для WK-алгоритму. З рис. 3.8(А) видно, що при пошуку прото-об’єкту на природних зображеннях (експерименти №1 і №3) значення повноти вище для НПО-алгоритму, чим WK-алгоритму. Середній час обробки одного зображення алгоритмом знаходження прото-об’єкту склало 3.9 з, що на 1.1 с. більше в порівнянні з WK-алгоритмом.

Таким чином, НПО-алгоритм дозволяє точніше знайти прото-об'єкт на природних зображеннях, і може бути використаний при пошуку зображень за змістом.

Далі опишемо розроблений автором прототип CBIR-системи, орієнтованої на пошук схожих об'єктів на зображеннях, методи пошуку зображень із використанням ознак прото-об’єкту й результати експериментів по перевірки їхньої працездатності.

Відзначимо, що при пошуку зображень по візуальній подобі варто враховувати наступні обставини:

1. На зображеннях присутнє тло, що вносить перекручування в глобальні характеристики зображень;

2. При пошуку зображення людини в більшості випадків цікавить конкретний об'єкт на зображенні-запиті;

3. Ряд відомих сайтів (наприклад, Яндекс.Фоткі, Flickr, Вконтакті) надають користувачеві можливість викладати власні фотографії, постачена сервісом, що дозволяє облямовувати область, що цікавить, обмежуючим прямокутником, а також підписувати об'єкти на зображенні.

Додаткова інформація, одержувана від користувача про положення об’єкту на зображенні, може бути використана для уточнення алгоритмів автоматичного знаходження зображень. Отже, доцільно попередньо знаходити на зображеннях об'єкти, що залучають увагу людини, після чого оцінювати подібність об'єктів на зображеннях у базі з об'єктом на зображенні запиту. Запропонований підхід реалізується наступною послідовністю дій:

1. Формування запиту: запит задається у вигляді зображення, на якому користувач визначає область, що відповідає об'єкту. При цьому об'єкт може бути заданий двома способами – вручну (користувач задає його об'єкт, що цікавить, виділяючи його на зображенні спеціальним інструментом, окреслюючи область ламаною лінією) або автоматично (об'єкт на зображенні перебуває за допомогою алгоритму знаходження прото-об’єкту).

2. Добування ознак: для перевірки підходу пошуку зображень із використанням інформації про прото-об'єкт обчислювалася тільки ознака кольору – гістограма кольору в просторі RGB (8 інтервалів у гістограмі).

3. Вимір подібності зображень: як міра подібності зображень використовувалася відстань Бхаттачарія (Bhattacharyya distance)).

Проведено експериментальну перевірку запропонованого підходу, у ході якої пошук здійснювався по колекції зображень, що складає з 2233 кольорових зображень (обраних з фотохостінга Яндекс.Фотки і Яндекс.Картинки). Кожне зображення було віднесено до однієї з 49-ти заздалегідь заданих категорій. Причому в кожній категорії була різна кількість зображень, що дозволило наблизитися до реальної бази зображень, зміст якої в цілому носить випадковий характер.

Знайдені зображення вважалися релевантними, якщо вони попадали в ту ж категорію, що й зображення-запит (Строга релевантність). Відзначимо, що в ряді випадків людина відносить зображення до однієї категорії не тільки на основі візуальної оцінки їхньої подібності, але й з використанням додаткової наявної в нього інформації. Отже, Строга релевантність заснована на семантичній подібності. Наприклад, слон, носоріг, бегемот по зовнішньому вигляді часом бувають дуже схожі. Однак людина звичайно шукає саме те, що зображено на картинці, тому в нашій тестовій колекції зображень вони віднесені в різні категорії, і при строгій оцінці вони вважаються нерелевантними.

Для пошуку зображення були використані наступні методи:

1. Пошук за ознакою кольору прото-об’єкту (об'єкт на зображенні-запиті задає користувач).

2. Пошук за ознакою кольору прото-об’єкту (об'єкт на зображенні-запиті перебуває автоматично за допомогою розробленого алгоритму знаходження прото-об’єкту).

3. Комбінований пошук. Об'єднання пошуку по глобальній ознаці кольору й пошуку за ознакою кольору прото-об’єкту (об'єкт на зображенні-запиті задає користувач).

Пошук по глобальній ознаці кольору є широко використовуваним методом пошуку зображень за змістом, тому в роботі виконане порівняння запропонованих методів з пошуком по глобальній ознаці кольору.

Для кількісної оцінки якості пошуку використовувалися наступні критерії:

1. Точність на рівні n документів (Точність(n)). Тут точність – частка релевантних зображень у загальному числі знайдених. Точність на рівні n документів визначається як кількість релевантних документів серед перших n виданих документів, ділена на n (наприклад, точність(5) – частка релевантних зображень у перших 5-ти знайдених). Даний критерій є незамінним критерієм при оцінці якості сучасних систем пошуку, тому що, зокрема, дозволяє оцінити корисність першої сторінки відповіді системи для користувача.

2. Повнота(50) – частка релевантних зображень, виявлених серед перших 50-ти знайдених, у загальній кількості релевантних по даному запиту.

3. R-точність. R-точність дорівнює точності на рівні n документів для n рівного кількості релевантних документів для даного запиту. Дана метрика особливо корисна в тих випадках, коли для різних запитів спостерігається більша різниця в кількості відомих релевантних документів.

4. Середня точність. Середня точність для даного запиту визначається в такий спосіб: нехай для даного запиту є k релевантних документів. Точність на рівні i-го релевантного документа prec_rel (i) дорівнює precision (pos (i)), якщо

і-й релевантний документ перебуває в результатах запиту на позиції pos (i). Якщо і-й релевантний документ не знайдений, то prec_rel (i) = 0. Середня точність для даного запиту дорівнює середньому значенню величини prec_rel(i) по всім k релевантним документам:

$$Avg\ Prec = \frac{1}{k} \cdot \sum_{i=1}^{k} prec\_rel(i).$$

Проведені 3 експерименти, у ході яких отримані оцінки ефективності:

Експеримент I. Пошук по глобальній ознаці кольору.

Експеримент II. Пошук за ознакою кольору рівня прото-об’єкту (об'єкт на зображенні-запиті перебуває автоматично).

Експеримент III. Пошук за ознакою кольору рівня прото-об’єкту (об'єкт на зображенні-запиті задає користувач).

Експеримент IV. Комбінований пошук (облік глобальних ознак кольору й ознак прото-об’єкту).

Результати експериментів при оцінці строгої релевантності представлені на рис. 3.9.

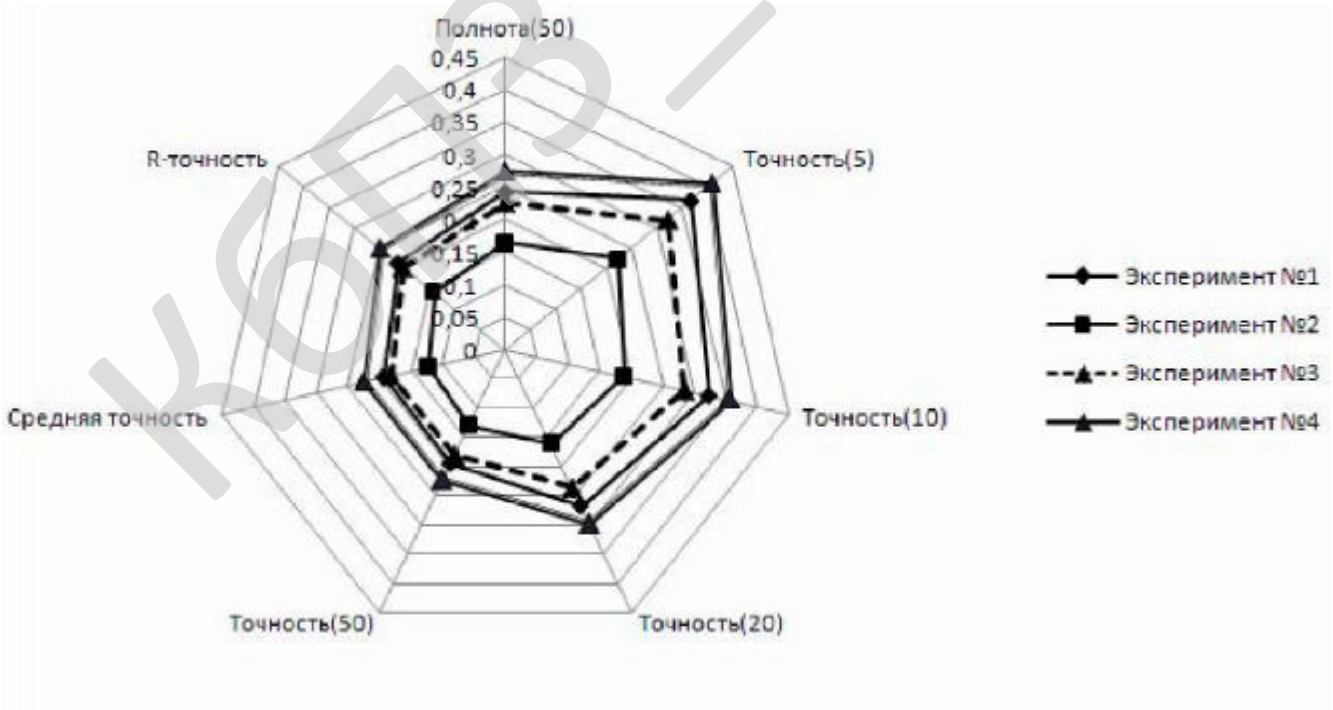

Рисунок 3.9 – Результати 3-х експериментів. Строга релевантність

Результати порівняння запропонованих методів з пошуком по глобальній ознаці кольору представлені в табл. 3.3.

Таблиця 3.3 – Порівняння результатів 3-х експериментів. Строга релевантність

		Порівняння з пошуком по глобальній ознаці кольору (експеримент 1)		
№	критерій	Пошук за ознакою кольору прото-об’єкту (об’єкт на запиті перебуває автоматично)	Пошук за ознакою кольору прото-об'єкту (об'єкт на запиті задає користувач)	Комбінований метод пошуку (облік глобального кольору й кольори прото-об’єкту)
1	повнота(50)	нижче на 32 %	нижче на 6%	вище на 12%
2	точність(5)	нижче на 39%	нижче на 13%	вище на 10%
3	точність(10)	нижче на 42%	нижче на 12%	вище на 9%
4	точність(20)	нижче на 40%	нижче на 12%	вище на 10%
5	точність(50)	нижче на 33%	нижче на 8%	вище на 13%
6	середня точність	нижче на 37%	нижче на 6%	вище на 14%
7	R-точність	нижче на 33%	нижче на 4%	вище на 14%
Номер експерименту		2	3	4

Виявляється, що якщо об'єкт на зображенні задається автоматично (експеримент №2), то пошук менш ефективний у порівнянні з пошуком по об'єкту, що задається вручну (експеримент №3). Отже, об'єкт на зображенні-запиті доцільно задавати вручну, оскільки користувач точніше формулює запит для пошуку.

Однак, на рис. 3.9 видно, що значення критеріїв досить близький друг до друга, тому далі проведене дослідження з оцінки збігу в зображеннях, вірно знайдених за допомогою пошуку, по наступних ознаках:

1. Глобальній ознаці кольору (експеримент №1) і ознаці кольору прото-об’єкту, що перебуває автоматично (експеримент №2).

2. Глобальній ознаці кольору (експеримент №1) і ознаці кольору прото-об’єкту, що задається користувачем (експеримент №3).

3. Ознаці кольору прото-об’єкту, що перебуває автоматично (експеримент №2) і ознаці кольору прото-об’єкту, що задається користувачем (експеримент №3).

Результати оцінки збігу в зображеннях представлені в табл. 3.4.

Таблиця 3.4 – Відсоток збігів від загального числа вірно знайдених зображень

№	Характеристика	Експерименти №1 і №2	Експерименти №1 і №3	Експерименти №2 і №3
1	Загальна кількість вірно знайдених зображень по всіх запитах (сума знайдених 2-ма методами)	1872	2036	1600
2	Відсоток знайдених зображень, що збіглися, (від загальної кількості вірно знайдених)	25.5 %	33.4 %	40.0 %

З табл. 3.4 видно, що:

1. При пошуку по глобальній ознаці кольору (експеримент №1) і ознаці кольору прото-об’єкту, що перебуває автоматично (експеримент №2) збігається лише 25.5 % серед вірно знайдених зображень обома методами.

2. При пошуку по глобальній ознаці кольору (експеримент №1) і ознаці кольору прото-об’єкту, що задається користувачем (експеримент №3) збігається лише 33.4 % серед вірно знайдених зображень обома методами.

3. При пошуку за ознакою кольору прото-об’єкту, що перебуває автоматично (експеримент №2) і ознаці кольору прото-об’єкту, що задається користувачем (експеримент №3) збігається лише 40.0 % серед вірно знайдених зображень обома методами.

Аналіз результатів проведених експериментів по глобальному пошуку зображень і за ознакою прото-об'єктів дозволяють зробити наступні виводи:

1. Пошук по глобальній ознаці кольору й за ознакою прото-об’єкту дають приблизно однакова якість пошуку (за значеннями критеріїв повноти й точності на рівні перших 50-ти знайдених зображень);

2. Існують класи зображень, які доцільно шукати тільки за ознаками прото-об'єктів (1 клас), і зображень, які доцільно шукати тільки по глобальних ознаках (2 клас);

3. Виявилося, що відсоток збігу в зображеннях, вірно знайденими по глобальній ознаці кольору й за ознакою кольору прото-об’єкту, становить приблизно 30%. Отже, 70% зображень можливо знайти, використовуючи по черзі пошук по глобальній ознаці кольору й за ознакою кольору прото-об’єкту.

Таким чином, однієї з можливостей поліпшення якості пошуку є комбінування пошуку по глобальних ознаках і за ознаками прото-об'єктів. З рис. 3.9 і табл. 3.3 видно, що комбінований метод пошуку за всіма критеріями показав найкращі результати.

Ілюстрація результатів пошуку за глобальним кольором й за ознаками прото-об'єктів представлена на рис. 3.10-11.

Рисунок 3.10 – Ілюстрація результатів пошуку за глобальним кольором (перше зображення – запит)

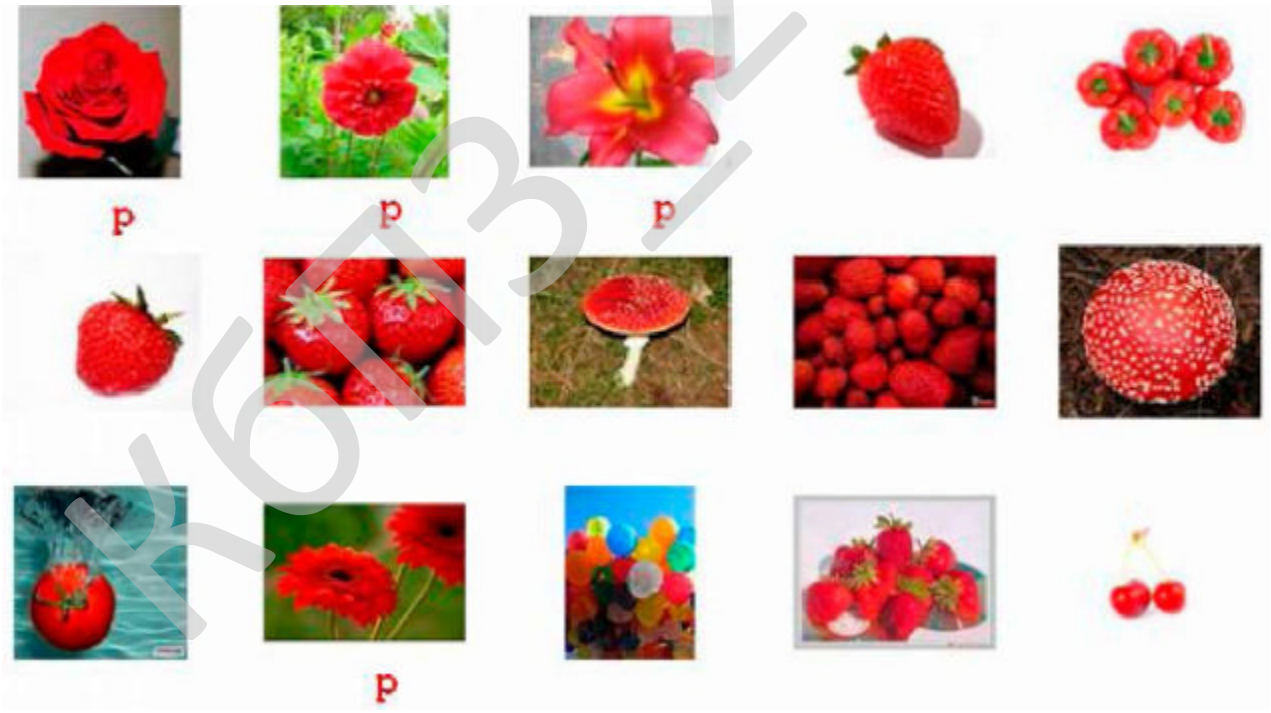

Рисунок 3.11 – Ілюстрація пошуку за ознаками прото-об'єктів (зображення-запит теж саме, що й у попередньому випадку, об'єкт на запиті задає користувач)

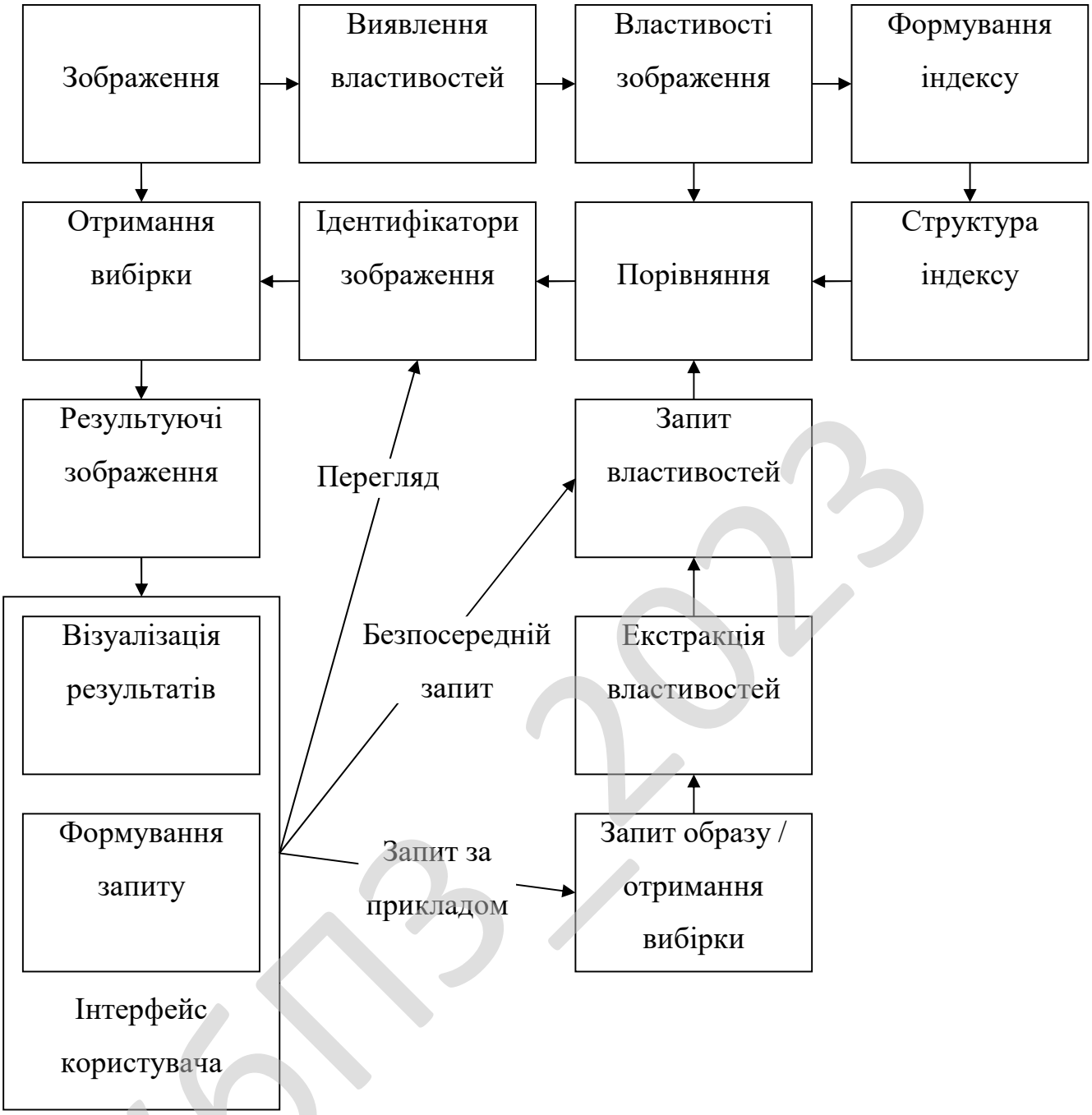

Рисунок 3.12 – Структурна схема системи

З рис. 3.10 видно, що серед перших 14-ти знайдених зображень релевантним запиту немає жодного. При цьому очевидно, що в цьому випадку якість пошуку перекручена впливом зеленого тла.

З рис. 3.11 видно, що серед знайдених зображень релевантними запиту є 3 зображення (не з огляду на знайдене зображення-запит). Причому всі об'єкти на

знайдених зображеннях того ж кольору, що й об'єкт на запиті. Отже, використання інформації тільки про об'єкт на зображенні в цьому випадку дозволило уникнути впливу тла на результати пошуку. У той же час необхідно використовувати додаткові ознаки прото-об'єктів, щоб мати можливість розрізняти друг від друга об'єкти одного кольору. Однак дані приклади ілюструю можливість використовувати пошук по прото-об'єкті для пошуку зображень зі схожими об'єктами.

3.3 Розробка функціональної схеми

Функціональна схема розробленої системи зображена на рисунку 3.13.

З рисунку видно, що розроблена система складається з наступних частин:

1. Інтерфейс користувача системи комп’ютерного зору для створення інтерактивних користувацьких інтерфейсів.

2. Вибір області застосування, й відповідних, цим областям, прото-об’єктів:

− Пошук зображень в мережі Інтернет.

− Каталогізація зображень творів мистецтва.

− Організація роботи з архівами фотографічних знімків.

− Організація каталогів роздрібного продажу товарів.

− Медична діагностика захворювань.

− Запобігання злочинів і заворушень.

− Застосування у військових цілях.

− Питання контролю за поширенням інтелектуальної власності.

− Отримання інформації про місце знаходження віддалених зондів і географічне позиціювання.

− Контроль за вмістом масивів зображень.

3. Блок обрання бази даних, звідки буде відбуватися аналіз зображень.

4. Введення ключового слова-тега для пошуку зображення.

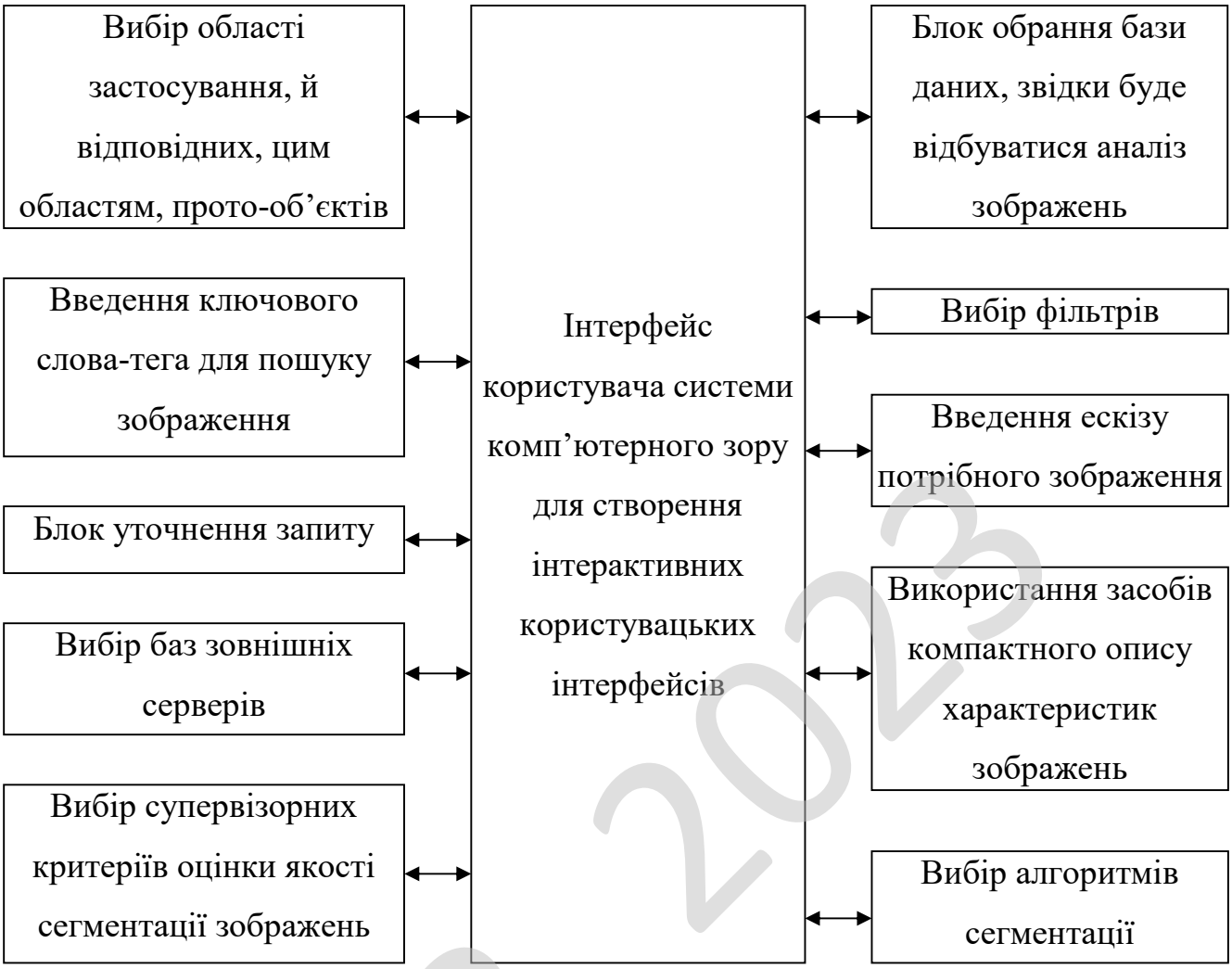

Рисунок 3.13 – Функціональна схема системи.

5. Блок уточнення запиту:

– Знайти схожі за тематикою.

– Знайти схожі за кольором й текстурою.

– Режим пошуку знімків конкретного автора

6. Блок введення тегів, для опису конкретного зображення.

7. Вибір баз зовнішніх серверів:

– Yahoo!

– Flickr.

– Picasa.

– Fotolia.

– iStockphoto.

– Facebook.

8. Вибір фільтрів:

– Сюжети зображень (портрети, загальні плани або пейзажі).

– Основна колірна гама зображення.

– Формат (звичайний або для широких екранів).

– Орієнтація зображення (книжкова або альбомна)

– Палітра найпростіших геометричних фігур.

9. Введення ескізу потрібного зображення.

10. Використання засобів компактного опису характеристик зображень:

– CEDD (фільтрт MPEG-7).

– FTCH (вейвлет-перетворення Хаара).

– JCD.

11. Вибір супервізорних критеріїв оцінки якості сегментації зображень:

– Dku.

– GCE.

– RI.

– RMS.

12. Вибір алгоритмів сегментації:

Алгоритм еволюції кривої на основі моделі геодезичних активних контурів (Geodesic Active Contours).

– Алгоритм еволюції кривої на основі потоку вектора градієнта (Gradient vector flow).

– Алгоритм еволюції кривої під управлінням потоку границь (Edgeflow-driven Curve Evaluation).

– Алгоритм анізотропної дифузії під управлінням потоку границь (Edgeflow-driven Anisotropic Diffusion).

– Алгоритм анізотропної дифузії, запропонований Перону й Маліком (Perona Malik Flow).

– Алгоритм анізотропної дифузії, запропонований Шапіро (Self– Snakes)

– Алгоритм JSEG сегментації зображень із обліком колірних і текстурних ознак зображення.

– Оператори країв Кенні, Робертса, Превітт, Собела, Zerocross, Log з додатковою обробкою по зв'язуванню границь.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.14. Першим процесом, який завантажується у системі, є процес виведення головного вікна програми.

Він взаємодіє з наступними процесами:

– Процес пошуку зображення у базі даних, який взаємодіє з процесом визначення особливих точок, за допомогою алгоритму SURF.

– Процес редагування бази даних.

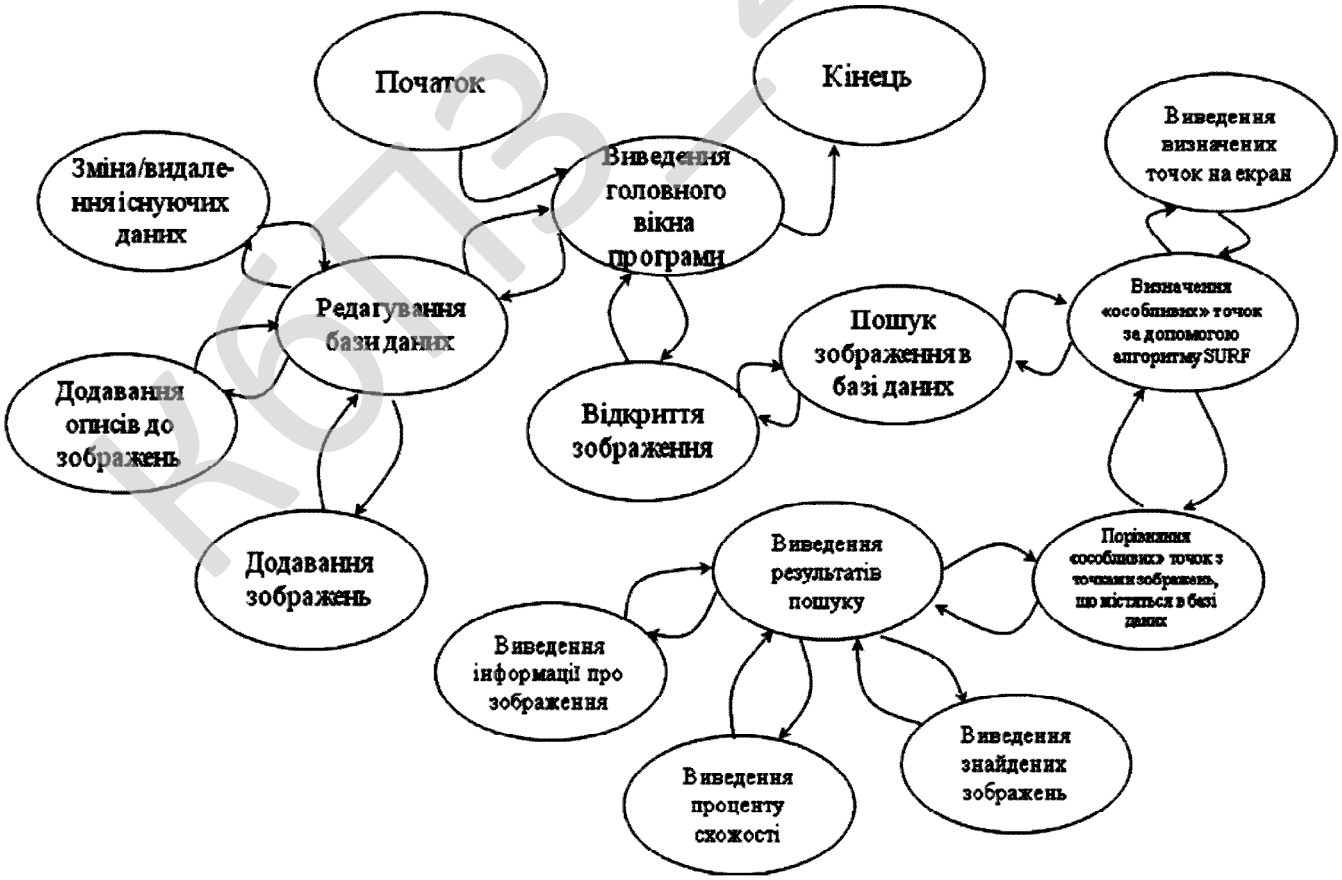

Рисунок 3.14 – Діаграма взаємодії процесів

Процес визначення особливих точок, за допомогою алгоритму SURF взаємодіє з наступними процесами:

– Процес виведення визначених точок на екран.

– Процес порівняння особливих точок з точками зображень, які містяться у базі даних.

Останній процес, у свою чергу, взаємодіє з процесом виведення результату пошуку, який взаємодіє з наступними процесами:

– Процес виведення знайдених зображень.

– Процес виведення процесу схожості.

– Процес виведення інформації про зображення.

Процес редагування бази даних взаємодіє з наступними процесами:

– Процес додавання зображень.

– Процес додавання описів до зображень.

– Процес зміни/видалення існуючих даних.

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо алгоритм роботи основної програми. Його блок-схема зображена на рисунку 4.1.

Спочатку виводиться основне вікно програми, після чого пропонується відкрити зображення. Якщо зображення відкривати не потрібно, програма надає можливість відредагувати базу даних, далі відбувається програма завершує свою роботу.

Основна частина роботи програми починається із завантаження файлу із зображенням та виведення його на екран. Далі виконуються наступні операції:

– Обчислення матриці Гессе.

– Створення дескрипторів характерних точок.

– Запис дескрипторів в окремий файл, прикріплений до файлу із зображенням.

– Виведення обчислених «особливих» точок на екран.

Після проведених операцій програма за допомогою методу SURF виконує пошук схожих зображень в базі даних.

Якщо схожі зображення знайдені, то на екран виводяться:

– Знайдені зображення.

– Відсоток схожості.

– Текстова інформація про зображення.

В протилежному випадку виводиться повідомлення про відсутність схожих зображень в базі даних.

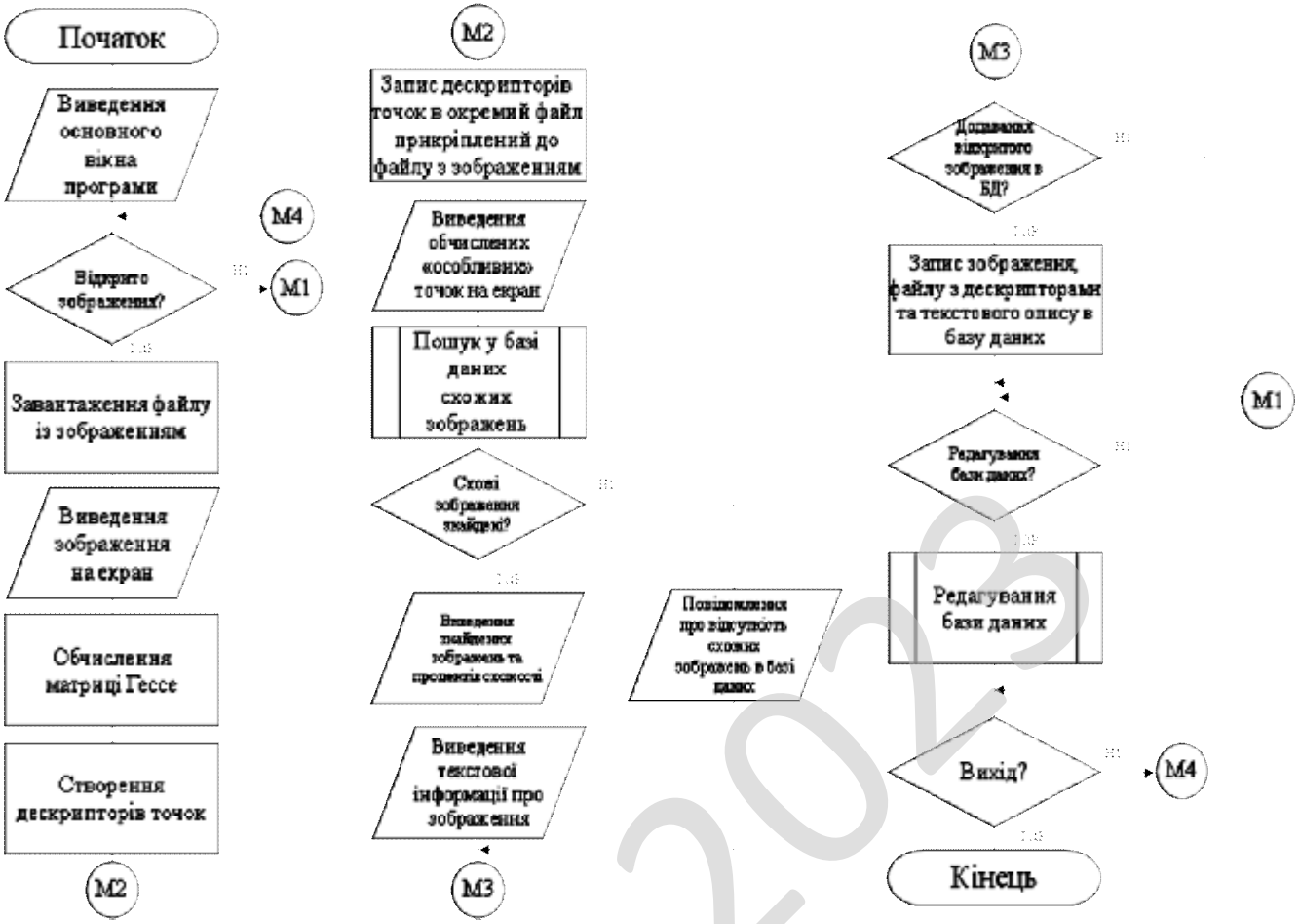

Рисунок 4.1 – Блок-схема основної програми

На наступному кроці пропонується додати відкритий файл до бази даних. В цьому випадку до бази даних записуються:

– Зображення.

– Файл із дескрипторами.

– Текстовий опис до зображення.

Після цього знову надається змога редагувати базу даних, після чого або відкривається нове зображення, або програма завершує роботу.

На рисунку 4.2 наведено блок-схему підпрограми пошуку зображень за методом SURF. Розглянемо її роботу.

Підпрограма зчитує дескриптори ключових точок зображення шаблону та підключається до бази даних.

Далі, поки не перевірені всі зображення в базі даних виконуються в циклі наступні операції:

– Зчитуються дескриптори ключових точок поточного зображення.

– Порівнюються дескриптори точок поточного зображення із дескрипторами точок шаблону.

– Обчислюється процент схожості.

– Якщо процент схожості (відповідність дескрипторів) більше 60%, то індекс поточного зображення та процент відповідності зберігаються в буфер.

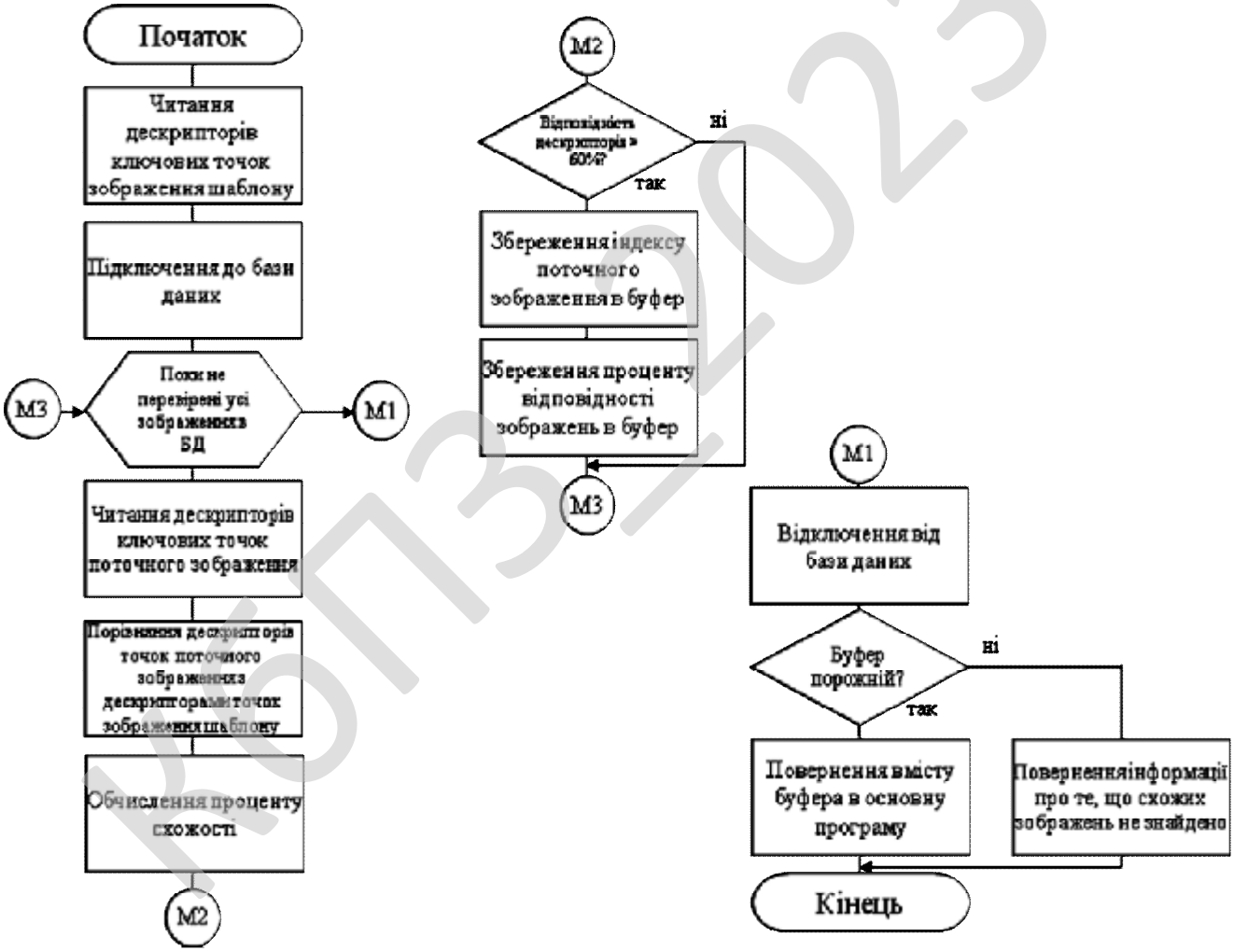

Рисунок 4.2 – Блок-схема підпрограми пошуку зображень за методом SURF

Для порівняння зображень із врахуванням їх можливих поворотів визначається орієнтація зображення. Наведемо частину коду, призначену для цієї мети:

```
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;
  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;
      }
    }
  }
  float sumX=0.f, sumY=0.f;
  float max=0.f, orientation = 0.f;
  float ang1=0.f, ang2=0.f;
  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];
            }
          }
          if (sumX*sumX + sumY*sumY > max)
          {
            max = sumX*sumX + sumY*sumY;
            orientation = getAngle(sumX, sumY);
          }
        }

        ipt->orientation = orientation;
      }
```

Після цього пошук дескрипторів звужується до блоків Гауса 4x4, в яких координати визначених точок повертаються на вісь обертання. Для цього призначена наступна послідовність операцій в коді програми:

```
      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;
```

Дескриптор для цієї особливої точки розраховується наступним чином:

```
  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)
        {
          sample_x = fRound(x + (-l*scale*si + k*scale*co));
          sample_y = fRound(y + ( l*scale*co + k*scale*si));
          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));
          rrx = gauss_s1*(-rx*si + ry*co);
          rry = gauss_s1*(rx*co + ry*si);
          dx += rrx;
          dy += rry;
          mdx += fabs(rrx);
          mdy += fabs(rry);
        }
      }
```

Далі значення додаються 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;
        }
        len = sqrt(len);
        for(int i = 0; i < 64; ++i)
          desc[i] /= len;
      }
```

Порівняння відбувається шляхом обчислення суму пікселів в межах прямокутника, вказаного верхньою лівою координатою і розміром, за допомогою операції BoxIntegral:

```
      inline float BoxIntegral(IplImage *img, int row, int col, int rows, int
cols)
      {
        float *data = (float *) img->imageData;
        int step = img->widthStep/sizeof(float);
        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);
      }
```

Для визначення вмісту зображення, воно розбивається на шари 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]);
          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);
        }
      }
    }
```

Потім будується карта відповідностей. Для цього спершу беремо атрибути зображення:

```
int w = (i_width / init_sample);
int h = (i_height / init_sample);
int s = (init_sample);
```

Далі розраховуваємо апроксимаційний 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));
}
подібним чином для значень octaves >= 2, 3, 4, 5  }
for (unsigned int i = 0; i < responseMap.size(); ++i)
{
  buildResponseLayer(responseMap[i]);
}
```

Після цього визначаємо відповідний DoH для шару за допомогою BuildResponseLayer:

```
void FastHessian::buildResponseLayer(ResponseLayer *rl)
{
  float *responses = rl->responses;
  unsigned char *laplacian = rl->laplacian;
  int step = rl->step;
  int b = (rl->filter - 1) / 2 + 1;
  int l = rl->filter / 3;
  int w = rl->filter;
  float inverse_area = 1.f/(w*w);
  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++)
      {
        r = ar * step;
        c = ac * step;
        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);
        Dxx *= inverse_area;
        Dyy *= inverse_area;
        Dxy *= inverse_area;

        responses[index] = (Dxx * Dyy - 0.81f * Dxy * Dxy);
        laplacian[index] = (Dxx + Dyy >= 0 ? 1 : 0);
#ifdef RL_DEBUG
        rl->coords.push_back(std::make_pair<int,int>(r,c));
#endif
      }
    }
  }
```

Тут спершу зберігаються відповіді та знак лапласіана, після чого визначається розмір кроку, границя та чинник нормалізації для цього фільтру. За координатами зображення r, c розраховуються компоненти Dxx, Dyy, Dxy та нормалізуються відносно розміру.

Після обчислення детермінанту відповідності гессіана створюється список координат зображень для кожної пари.

Після завершення пошуку відбувається відключення від бази даних.

Якщо буфер не порожній, його вміст повертається в основну програму. В протилежному випадку видається інформація про відсутність схожих зображень в базі даних.

На рисунку 4.3 наведена блок-схема підпрограми редагування бази даних.

Для початку роботи на екран виводиться вікно бази даних. Після цього користувач може додати нове зображення в базу даних або змінити існуючі дані.

Додавання нового зображення здійснюється шляхом виконання наступних кроків:

– Відкриття файлу із зображенням.

– Збереження зображення в базі даних.

– Обчислення ключових точок зображення.

– Збереження ключових точок в базі даних.

– Введення текстового опису зображення в базі даних.

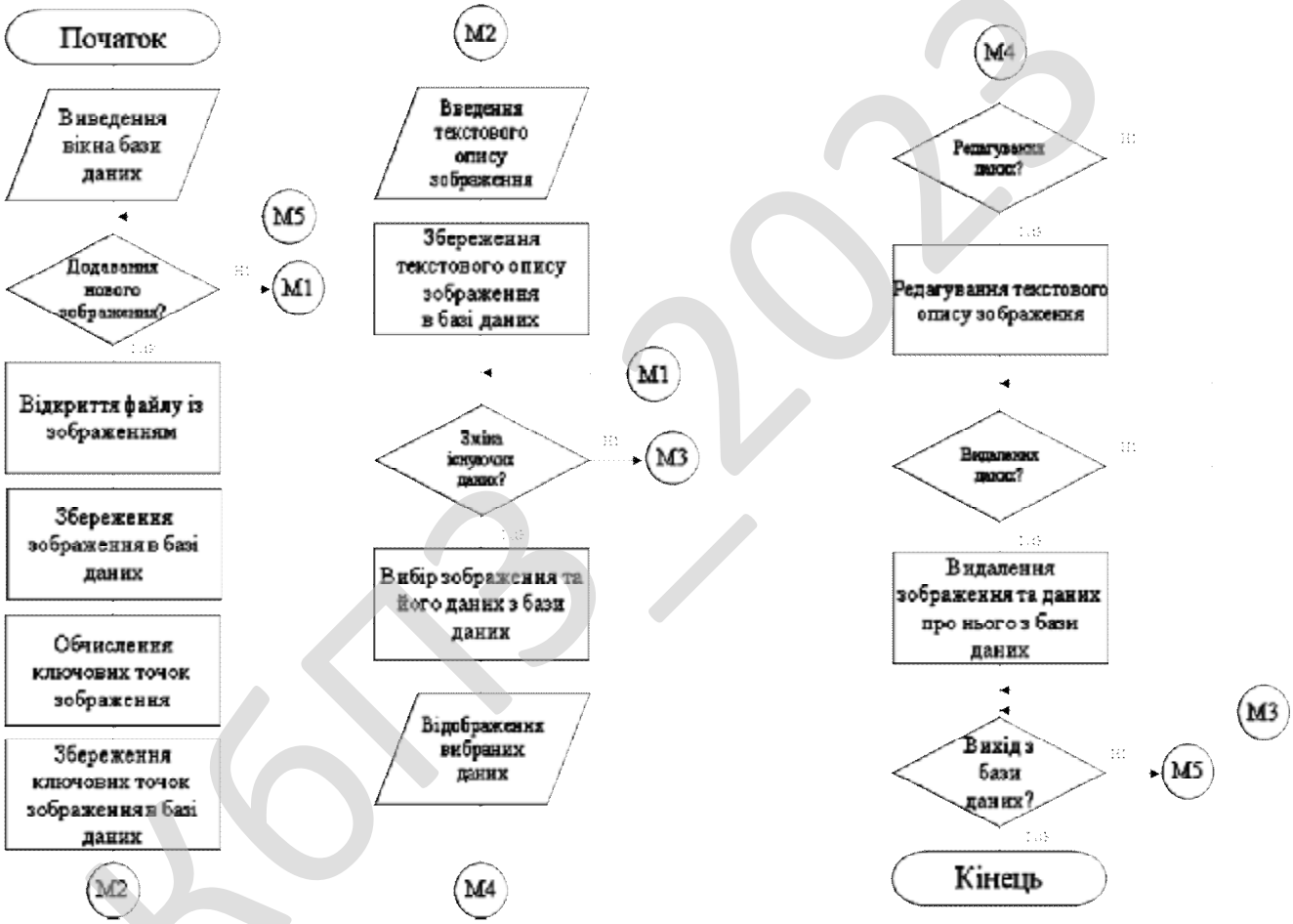

Рисунок 4.3 – Блок-схема підпрограми редагування бази даних

Зміна існуючих даних відбувається при виконанні наступних операцій:

– Відображення вибраних даних.

– Підтвердження редагування даних.

– Редагування текстового опису зображення.

– Вивід запиту на видалення даних.

– Видалення зображення та даних про нього із бази даних.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Threefish – в криптографії симетричний блоковий криптоалгоритм, розроблений автором Blowfish та Twofish, американським криптографом Брюсом Шнайером 2008 року для використання в хеш-функції Skein і як універсальну заміну наявним блоковим шифрам. Основними принципами розробки шифру були: мінімальне використання пам'яті, необхідна для використання в хеш-функції стійкість до атак, простота реалізації та оптимізація під 64-розрядні процесори.

Структура алгоритму

Threefish має дуже просту структуру і може бути використаний для заміни алгоритмів блочного шифрування, бувши швидким і гнучким шифром, що працюють в довільному режимі шифрування. Threefish S-блоки не використовує, заснований на комбінації інструкцій виключаючого або, складання і циклічного зсуву.

Як і AES, шифр реалізований у вигляді підстановочно-перестановочної мережі на оборотних операціях, не будучи шифром мережі Фейстел.

Алгоритм передбачає використання tweak-значення, свого роду вектора ініціалізації, дозволяючи змінювати таким чином значення виходу, без зміни ключа, що має позитивний ефект як для реалізації нових режимів шифрування, так і на криптостойкости алгоритму.

Як результат думки авторів, що кілька складних раундів часто гірше ніж застосування великого числа простих раундів, алгоритм має нетрадиційно велику кількість раундів – 72 або 80 при ключі 1024 біт, проте, за заявою творців, його

швидкісні характеристики випереджають AES приблизно вдвічі. Варто зауважити, що через 64-бітної структури шифру, дана заява має місцево лише на 64-розрядної архітектури. Тому, Threefish, як і Skein [1], заснований на ньому, на 32-розрядних процесорах показує значно гірші результати ніж на «рідному» обладнанні.

Ядром шифру є проста функція «MIX», перетворювальна два 64-бітових беззнакових числа, в процесі якої відбувається складання, циклічний зсув (ROL / ROR), і додавання по модулю 2 (XOR) .

Нижче представлений код MIX-функції для Threefish-1024 [2]:

```
<syntaxhighlight lang="C">
// Константи для циклічного зсуву
int R16 [8] [8] = {
  {55, 43, 37, 40, 16, 22, 38, 12},
  {25, 25, 46, 13, 14, 13, 52, 57},
  {33, 8, 18, 57, 21, 12, 32, 54},
  {34, 43, 25, 60, 44, 9, 59, 34},
  {28, 7, 47, 48, 51, 9, 35, 41},
  {17, 6, 18, 25, 43, 42, 40, 15},
  {58, 7, 32, 45, 19, 18, 2, 56},
  {47, 49, 27, 58, 37, 48, 53, 56},
};
// D – раунд, j – індекс в таблиці циклічного зсуву
void mix (int j, int d) {
  usigned long long rotl;
  y [0] = x [0] + x [1];
  rotl = R16 [d% 8] [j];
  y [1] = (x [1] << rotl) | (x [1] >> (64 – rotl));
  y [1] ^ = y [0];
}
</Source>
```

Процедура розшифрування обернена процедурі зашифрування і містить зворотну функцію DEMIX.

Кожен з 72 раундів Threefish-256 і Threefish-512 має чотири MIX перетворення, Threefish-1024 – вісім звернень до MIX функції.

Безпека

За заявою авторів, алгоритм має більш високий рівень безпеки, ніж AES. Існує атака на 25 з 72 раундів Threefish, в той час як для AES – на 6 з 10. Threefish має показник фактора безпеки 2.9, в свою чергу, AES всього 1.7 [3]

Для досягнення повної дифузії, шифру Threefish-256 досить 9 раундів, Threefish-512 – 10 раундів і Threefish-1024 – 11 раундів. Виходячи з цього, 72 і 80 раундів відповідно в середньому, забезпечать кращі результати, ніж існуючі шифри. [4]

У той же час, алгоритм має набагато простішу структуру і функцію перетворення, проте виконання 72-80 раундів, на думку дослідників, забезпечує необхідну стійкість. Вживаний розмір ключа від 256 до 1024 біт зводить нанівець можливість повного перебору паролів при так званій атаці грубою силою (brute force attack) на сучасному обладнанні.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Уявлення про інтерфейс програми можна отримати з рисунку 5.1, де наведено головне вікно програми із завантаженим зображенням, пошук якого потрібно здійснити. В окремих вікнах виводяться вихідне (завантажене) зображення, ідентифіковане зображення та текстовий опис.

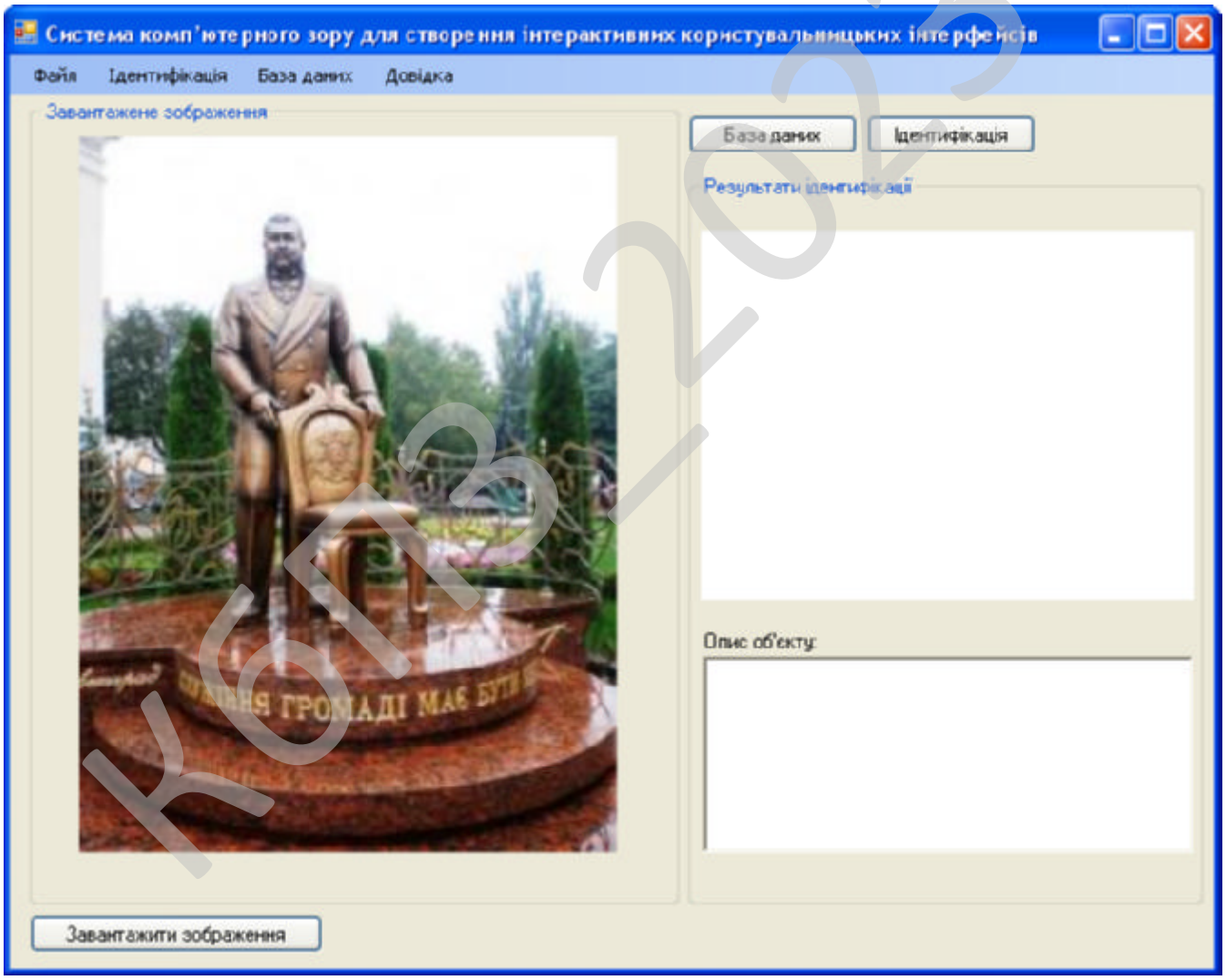

Рисунок 5.1 – Головне вікно програми (до застосування алгоритму пошуку SURF)

Управління програмою здійснюється за допомогою панелі меню, яке містить наступні пункти:

– Файл.

– Ідентифікація.

– База даних.

– Довідка.

Для завантаження зображення призначена також окрема кнопка внизу головного вікна.

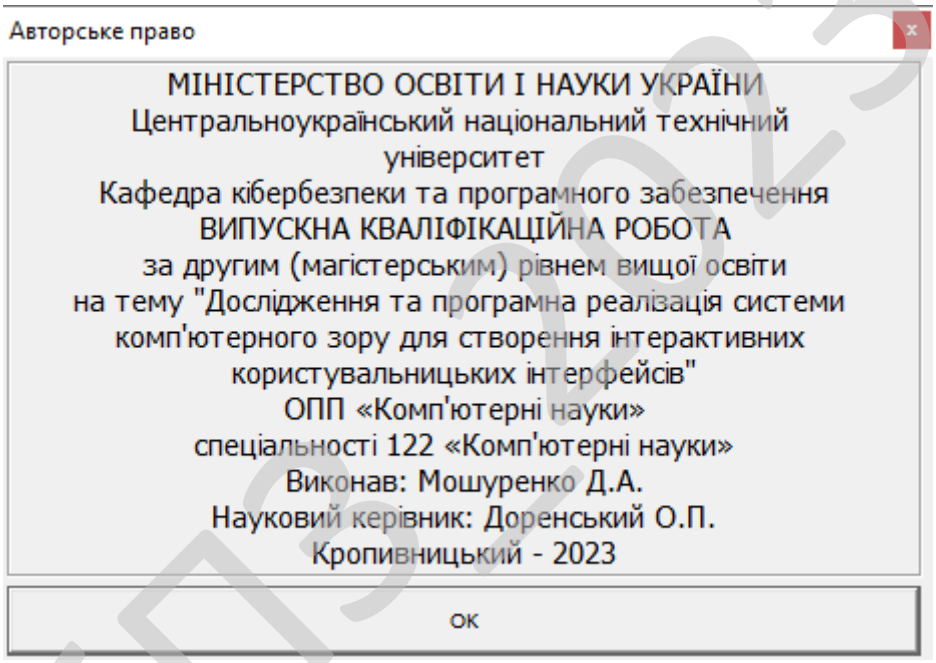

Рисунок 5.3 – Довідка

На рисунку 5.3 наведено вікно довідки про програму. В цьому вікні вказується, що програма є магістерською роботою та її тема, вказуються розробник, керівник та вихідні дані про місто та час виконання.

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Метою розробки є дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Об'єктом дослідження є процес комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Предметом дослідження є методи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Методи дослідження базуються на методах розпізнавання образів, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– Розроблено вітчизняний продукт комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 60 днів (три місяці).

В магістерській роботі було проведене дослідження та виконана програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів. Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт	N	1
2. Кількість екземплярів програм, шт	Ne	50
3. Запланований термін розробки, днів	Fpq	60 (3 місяці)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Б
6. Складність алгоритму (1, 2, 3)	–	2
7. Кількість макетів вхідної інформації	–	3

Продовження табл. 7.1

1	2	3
8. Кількість форм вихідної інформації.	–	4
9. Мова програмування (1-6)	–	2
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	2
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	2
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	2
17. Складність кінцевого програмного продукту (1-6)	–	2
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	2
20. Вимоги до швидкодії ПП (1-6)	–	2
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	2
23. Професійний рівень аналітиків (1-6)	–	2
24. Професійний рівень програмістів (1-6)	–	2
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	2
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження табл. 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПО для декількох серверів одночасно (1-6)	–	2
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн	–	50000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	55
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A \, \text{Size}^B , \qquad (7.1)$$

де A – коефіцієнт Боема, A=2,45; Size – загальний об'єм відлагодженого програмного коду, тис. рядків; B – показник ступеня, що визначається співвідношенням

$$B = 1{,}01 + 0{,}001\sum W_i \qquad (7.2)$$

де Wi – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B=1{,}01+0{,}001(2{,}43+3{,}64+3{,}38+3{,}95+2{,}73) = 1{,}026$$

$$T_{ном} = 2{,}45 \cdot 2{,}7^{1{,}026} = 6{,}78 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн}=T_{ном}\, ПВ_j, \qquad (7.3)$$

де $ПВj$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн}= 6{,}78\cdot(0{,}88\cdot0{,}93\cdot0{,}88\cdot0{,}91\cdot0{,}95\cdot1\cdot1\cdot0{,}87\cdot1{,}22\cdot1{,}16\cdot1{,}1\cdot1{,}1\cdot$$
$$\cdot1{,}12\cdot1{,}1\cdot1\cdot1{,}1\cdot1{,}1)=9{,}37 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3CT_{уточн}^{0{,}33+0{,}2(B-1{,}01)}S, \qquad (7.4)$$

де C – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); S – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПО згідно встановленим вимогам. Вибираємо в межах (25…350)%

$$T_{РП}=0{,}3\cdot2{,}66\cdot9{,}37^{0{,}33+0{,}2(1{,}026-1{,}01)}\cdot56=95 \text{ люд/день}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	9	Д7
Робочий проект	95	Ф 7.1-7.4
Впровадження	13	Д13
Всього	136	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \quad (7.5)$$

де Fpq – плановий фонд робочого часу одного спеціаліста, днів, Тпз – трудомісткість розробки програмного забезпечення люд-дні,

$$Ч = \frac{136 \cdot 1}{60\text{-}5} = 2,5 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	30	7	210	3,5
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	2	240	4
Принтер струминний	60	1	60	1
Сканер	20	1	20	0,33
Концентратор-маршрутизатор	30	3	90	1,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	250	625	10,42
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	44,08

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1,2} \quad (7.6)$$

$$Ф_{др}^{с} = \frac{44 \cdot 3}{1,2} = 110 \text{ год}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}} \quad (7.7)$$

$Ч_{ел}$=110/(60·8)=0,23 ставки

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів – електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	Кількість штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, серверу доступу АДСЛ (ОС Linux), Wi-Fi налаштування ADSL, VPN, PPPoE, Frame Relay	2	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	0,5	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	0,5	
	Забезпечення цілодобової роботи зв'язку клієнтів д мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	Кількість штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	1	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,25	
	Контроль взаєморозрахунків з постачальниками	0,25	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,25
	Створення графічних і стилістичних елементів сайту	0,5	
	Оформлення банерів і промо-сторінок	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,25
	Верстка друкованих видань	0,5	
	Додрукова підготовка макетів	0,25	
	Розміщення графіки і контенту на Інтернет сторінках	0,25	
Всього		2	

Складемо штатний розклад виконавців:

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньо-місячний оклад, грн.	Всього за період роз-робки, грн.
Керівник (IT-менеджер)	1	22000	66000
Продакт-менеджер	0,25	17000	12750
Інженер-програміст	2,5	20000	150000
Інженер-електронщик	0,23	15000	10350
Інженер-системотехнік	0,25	15000	11250
Адміністратор мережі	0,5	15000	22500
Системний програміст	0,25	15000	11250
Дизайнер WEB	0,25	16000	12000
Інженер-верстальник	0,25	15000	11250
Бухгалтер-економіст	0,5	17000	25500
Всього за період розробки	$R_{сп}$=5,98	-	$Ф_{роб}$=332850

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де Фроб – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{332850}{5,98 \cdot 60} = 928 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 8 робочих місць. Sy – питома площа на одне робоче місце, $м^2$; $Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних ТОВ науково-дослідницького консалтингового підприємства «Пектораль» ціна одного квадратного метра площі новобудови, вік якої не перевищує 25 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 37 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 8 \cdot 8 \cdot 20000 = 1280000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 128000 грн.

Балансова вартість інвентарю розраховується за нормою 3500 грн на одне робоче місце. Тобто

$$I_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де Цм – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 8 \cdot 3500 = 28000 \text{ грн}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7. Дані по оптовій ціні на обладнання та комплектуючі вибирались за прайсом фірми hotline за 16.10.23 – джерело https://hotline.ua

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		10947
Системний блок		7347
Процесор	AMD Ryzen 3 3200G (YD3200C5FHMPK) AM4, 4 ядра, 4 потоки, 3.6 GHz, 4.0 GHz, TDP – 65 Вт, 12nm	-
Системна плата	GIGABYTE A520M H DDR4, 64 ГБ, 5100 MHz, LAN – 1 Гбіт/с, DVI, HDMI, 1 x M.2 2280, 4 x Sata 6.0 Gb/s, Micro-ATX	-
Відеокарта	Radeon Vega 8	-
Жорсткий диск	SSD M.2 2280 240GB Apacer (AP240GAST280-1) 240 GB, TLC, M.2, SATA III (6Gb/s)	-
Оперативна пам’ять	DDR4 8GB 2666 MHz Kingston (KVR26N19S8/8)	-
DVD-привод	DVD -RW/+RW , LG SATA SuperMulti Bulk 22x, SecurDisc, black	-
Корпус	Gamemax MT520-450W Мультимедійний, Miditower, ATX, Mini – ITX, PSU – 450 Вт, Слотів расширення – 7	-
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-B, int. 3.5", 1*USB2.0+AUDIO+1394, multi: All Type Cards, black	-
інше	Клавіатура, мишка	-
Система охолодження	Deepcool BETA 10 Для процесорів – AMD, сокет – AM4, AM2, AM2+, AM3, AM3+, FM1, FM2	

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Монітор	LG W2363V-WF Wide LCD 2ms, 70 000:1, 300кд/м2, 170/160, D-Sub / Glossy White	3600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струменевий	Epson Stylus Photo P50 (C11CA45341) + USI cable	5500
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп’ютери	8	10947	8757,6	96333,6
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	114885,1

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відрахування, грн.
1	2	3	4
Група 3			
1. Будівлі	1280000	-	-
2. Передавальні пристрої	128000	-	-
Всього по групі	1408000	5	70400
Група 4			
3. Обчислювальна техніка	114885	-	-
Всього по групі	114885	50	57442,5
Група 5			
4. Вимірювальні пристрої	5190	-	-
5. Господарський інвентар	28000	-	-
Всього по групі	33190	25	8297,5
Нематеріальні активи			
6. Нематеріальні активи	50000	10	5000
Разом	$К_р$= 1606075		$А_р$= 141140

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_e}, \quad (7.11)$$

де Ne – Кількість екземплярів програм, шт.

$$З_о = 928 \cdot 136/50 = 2524 \text{ грн}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%

$$З_д = З_о \cdot Н_q \cdot 0{,}01, \quad (7.12)$$

де $Н_q$ – норматив додаткової зарплати, %

$$З_д = 2524 \cdot 10 \cdot 0{,}01 = 252{,}4 \text{ грн}$$

Відрахування на соціальні потреби за нормативом Нс=22% від суми основної та додаткової зарплати

$$С_{оц} = 0{,}01 \cdot Н_с (З_о + З_д), \quad (7.13)$$

де $Н_с$ – відрахування на соціальні потреби, %

$$С_{оц} = 0{,}01 \cdot 22(2524+252{,}4) = 611 \text{ грн}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом Нг=15% від основної зарплати

$$Г_{осп} = З_о \cdot Н_г \cdot 0{,}01, \quad (7.14)$$

де $Н_г$ – загальногосподарські витрати, %

$$Г_{осп} = 2524 \cdot 15 \cdot 0{,}01 = 379 \text{ грн}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де $З_{М1}$ – вартість паперу, грн., $З_{М2}$ – вартість запам'ятовуючих пристроїв, грн., $З_{М3}$ – вартість фарби, картриджів, тонеру, грн., $N_е$ – кількість екземплярів програм, шт.

Згідно виданих норм $n_{міс}$ приймаємо 1/6 пачки паперу на місяць розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=210 грн., визначаємо вартість паперу за період розробки $N_м$ =3 міс:

$$З_{М1} = Ц_п \cdot N_м \, n_{міс}. \quad (7.16)$$

$$З_{М1} = 210 \cdot 3 \cdot 1/6 = 105 \text{ грн}.$$

Згідно виданих норм до вартості запам'ятовуючих пристроїв входить вартість CD/DVD дисків в кількості 10 примірників:

$$З_{М2} = \sum Ц_{д}., \quad (7.17)$$

де Цд – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 41,2 грн/шт., DVD-R LG 4,7Gb, 16x speed Cake box – 41,2 грн/шт.

$$З_{М2} = 41{,}2 \cdot 10 = 412 \text{ грн.}$$

Згідно норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{М3} = \sum Ц_{з}., \quad (7.18)$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{М3} = 574+558+570=1702 \text{ грн.}$$

$$З_{М} = (105 + 412 + 1702)/50=44 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом (Нп = 15%) від основної зарплати виконавців

$$О_{п} = З_{о} \cdot Н_{п} \cdot 0{,}01, \quad (7.19)$$

де Нп – норматив витрат на освоєння нових мов програмування, %

$$О_{п} = 2524 \cdot 15 \cdot 0{,}01=379 \text{ грн}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 50 прим.)

$$А_{м} = \frac{А_{р} \cdot N_{міс}}{N_{е} \cdot 12}, \quad (7.20)$$

де Ар – загальна річна сума амортизаційних відрахувань, грн.

$$А_{м} = 141140 \cdot 3/(50 \cdot 12)=706 \text{ грн}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції

$$С_{п} = З_{о} + З_{д} + С_{оц} + Г_{осп} + З_{м} + О_{п} + А_{м}. \quad (7.21)$$

$$C_n = 2524+252,4+611+379+44+379+706 = 4895 \text{ грн.}$$

Визначимо плановий прибуток за рівнем рентабельності (Рп) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 55%

$$П_р = 0,01 \cdot P_n \cdot C_n, \quad (7.22)$$

де Рс – рівень рентабельності, %

$$П_р = 0,01 \cdot 55 \cdot 4895 = 2692 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн.
1. Основна зарплата виконавців	$З_о$	2524
2. Додаткова зарплата виконавців	$З_д$	252
3. Відрахування на соціальні потреби	$С_{оц}$	611
4. Загальногосподарські витрати	$Г_{осп}$	379
5. Витрати на матеріали	$З_М$	44
6. Освоєння нових операційних систем, мов програмування	$О_п$	379
7. Амортизація основних фондів	$А_м$	706
8. Повна собівартість програмного забезпечення	$С_п$	4895
9. Плановий прибуток	$П_р$	2692
10.Ціна підприємства $Ц_п=С_п+П_р$	$Ц_п$	7587
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	1517,4
12.Відпускна ціна програмної продукції Ц = Цп + ПДВ	$Ц$	10279

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн	
	Базовий	Новий
Вартість програмної продукції	–	10279
Всього капітальних витрат	–	10279

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на обслуговування	$З_р$	26572	11072
2. Витрати на електроенергію	$З_{ел}$	2455	2215
3. Витрати на амортизацію	$З_{ам}$	0	2570
Всього витрат за рік	$І$	29027	15857

Витрати на оплату праці:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_с), \qquad (7.23)$$

де Тр – кількість годин обслуговування системи за рік, *год.*; Зг – заробітна плата обслуговуючого персоналу, *грн / год*.

Після купівлі нового програмного забезпечення кількість годин обслуговування системи зменшилась з 180 годин до 75 годин на рік, тому витрати на обслуговування склали:

$$З_{р\,баз} = 180 \cdot 110 \cdot 1{,}1 \cdot 1{,}22 = 26572 \; грн.$$

до

$$З_{р\,нов} = 75 \cdot 110 \cdot 1{,}1 \cdot 1{,}22 = 11072 \; грн.$$

Витрати на електроенергію визначаються з урахуванням спожитої потужності (Пел) в кіловатах, часу експлуатації технічних засобів (Тр) в годинах та ціни однієї кіловат-години (Цел).

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \qquad (7.24)$$

Зел баз= $\cdot 0{,}475 \cdot 2455 \cdot 3{,}8 = 4431$ грн

Зел нов= $\cdot 0{,}475 \cdot 1227 \cdot 3{,}8 = 2215$ грн

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	25	–	10279	–	2569,75
Всього відрахувань	-	–	10279	–	2569,75

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою

$$E_в = (Ц_п - C_п) \cdot N_е - \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (7587 - 4895) \cdot 50 - (0,05 \cdot 1408000 + 0,5 \cdot 114885 +$$
$$+0,25 \cdot 33190 + 0,1 \cdot 50000) \cdot 3/12 = 99315 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р^*}{(Ц_п - C_п) \cdot N_е}, \quad (7.26)$$

де: $K_р^*$ – балансова вартість основних фондів розробника.

$$T_в = \frac{1606075}{(7587 - 4895) \cdot 50 \cdot 12 / 3} = 2,9 \text{ років}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н (K_н - K_б), \quad (7.27)$$

де $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно, $K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються

$$E_{сп} = (29027 - 15268) - 0,25 \cdot 10279 = 11189 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н} \quad (7.28)$$

$$T_{сп} = \frac{10279}{29027 - 15268} = 0,75 \ року$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	50
2. Повна собівартість розробленої програми	Грн.	4895
3. Ціна розробленої програми	Грн.	7587
4. Плановий прибуток від реалізації розробленої програми	Грн.	2692
5. Рентабельність програмної продукції	%	55
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	1606075
7. Загальний прибуток від реалізації програмної продукції	Грн.	134600
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	99315
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	2,9
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	10279
11.Величина економічного ефекту у користувача програмної продукції	Грн.	11189
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,75

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Техніка безпеки – це система правил і заходів, які допомагають запобігти травмам, хворобам і аваріям на робочому місці або в повсякденному житті. Знання техніки безпеки дуже важливе, бо воно рятує життя і здоров'я людей. Наприклад, якщо людина працює на шахті, то вона повинна знати правила безпеки у вугільних шахтах, щоб не потрапити під обвал або не підірватися на міні. Або якщо людина хоче навчитися програмувати, то вона повинна дотримуватися гігієнічних вимог і не сидіти за комп'ютером занадто довго, щоб не зашкодити своїм очам і спині.

Охорона праці та здоров'я у сфері ІТ – це комплекс заходів, які спрямовані на забезпечення безпечних і здорових умов праці для працівників, які використовують інформаційні технології, а також на запобігання травматизму, професійним захворюванням і стресу.

Охорона праці та здоров'я у сфері ІТ включає такі напрями, як:

– Ергономіка – це наука про адаптацію робочого середовища до фізичних і психологічних особливостей людини[2]. Ергономіка вимагає врахування таких факторів, як розміри, форми, кольори, освітлення, шум, температура, вологість, вентиляція, постава, рухи, пози, втома, навантаження на очі та ін. Ергономіка допомагає покращити комфорт, продуктивність і задоволення працівників.

– Комп'ютерна безпека – це захист комп'ютерних систем і даних від несанкціонованого доступу, зміни, знищення або блокування. Комп'ютерна безпека вимагає використання антивірусних програм, фаєрволів, паролей, шифрування, резервного копіювання та інших технологій. Комп'ютерна безпека допомагає запобігти крадіжці, шпигунству, шантажу, саботажу та іншим загрозам.

– Соціальна взаємодія – це процес спілкування між людьми у робочому колективі або через мережеві сервіси. Соціальна взаємодія вимагає дотримання

правил етикету, поваги, толерантності, співробітництва та конструктивного діалогу. Соціальна взаємодія допомагає покращити настрій, мотивацію, комунікацію та творчий потенціал працівників.

Правила охорони праці і здоров'я для програмістів:

– Регулярно роби перерви в роботі. Вставай із-за столу і розминай м'язи.

– Налаштуй яскравість і контрастність монітору так, щоб не напружувати очі.

– Використовуй ергономічну мишку і клавіатуру, які зручно лягають у руку і не виклкають болю.

– Слідкуй за своєю поставою. Сиди прямо і не нахиляйся до екрану.

– Захищай свій комп'ютер від вірусів, шпигунських програм і хакерів. Оновлюй ативірусне програмне забезпечення і не відкривай підозрілі файли і посилання.

– Не забувай про соціальну взаємодію. Спілкуйся з колегами, друзями і родиною. – Відвідуй заходи, які тебе цікавлять. Не ізолюй себе від світу.

– Люби свою професію, але не забувай про інші сфери життя. Розвивай свої захопленя, хоббі і таланти. Знаходь рівновагу між роботою і відпочинком.

Закон України “Про охорону праці” визначає основні принципи, завдання, права і обов’язки суб’єктів відносин з охорони праці, а також організаційні та правові основи державного управління і контролю за дотриманням законодавства про охорону праці.

Згідно з цим законом, ІТ компанії повинні впроваджувати такі заходи з охорони праці:

Створювати на підприємстві службу охорони праці або призначати відповідальних осіб, які забезпечують розроблення, реалізацію та контроль за дотриманням заходів з охорони праці.

Забезпечувати безпечні і нешкідливі умови праці для працівників, використвючи сучасні засоби техніки безпеки, санітарно-гігієнічні умови, засоби колективного та індивідуального захисту, оптимальні режими праці та відпочинку.

Проводити атестацію робочих місць на відповідність нормативно-правовим атам з охорони праці та аудит з охорони праці.

Проводити навчання та інструктаж з питань охорони праці, з надання першої медичної допомоги потерпілим від нещасних випадків і правил поведінки у разі виникнення аварії[5].

Забезпечувати лікувально-профілактичне обслуговування працюючих, санітарно-побутове обслуговування, пільги і компенсації для працівників, які працюють у важких і шкідливих умовах.

Нести відповідальність за порушення законодавства про охорону праці та зподіяння шкоди життю і здоров'ю працівників.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів.

Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С 40...60% 0,1...0,2 м/с

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року. Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у

таблиці 8.2.

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину 20... 40 $м^3$ на людину Більше 40 $м^3$на людину	Не менше 30 Не менше 20 Може біти використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів – малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення,

роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [21]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівця

Практичне значення заходів щодо підвищення працездатності випливає із закономірностей її динаміки і зводиться ось до чого:

збільшення фази стійкого стану у фонді робочого часу;

прискорення процесу впрацювання;

віддалення фази розвитку втоми;

забезпечення високої продуктивності праці за нормальних фізіологічних затрат.

Комплекс заходів щодо підвищення і збереження працездатності працівників на оптимальному рівні реалізується на техніко-організаційному, соціально-економічному, санітарно-гігієнічному, медико-біологічному, психологічному напрямках.

Вагомим фактором високої працездатності і продуктивності праці є оптимізація трудових навантажень на основі механізації і автоматизації виробничих процесів, удосконалення технології, скорочення і ліквідації важкої ручної праці. Доведено, що при правильній організації праці на легких роботах спостерігається найбільша тривалість фази стійкого стану, а на важких роботах вона нетривала.

Високий рівень працездатності безпосередньо залежить від умов праці, оскільки поліпшення їх супроводжується зменшенням енергетичних затрат організму на подолання несприятливого впливу факторів виробничого середовища.

Важливим напрямком підвищення працездатності працюючих є ритмізація трудових процесів, оптимізація темпу роботи, а також раціоналізація трудових

рухів на фізіологічній основі, що сприяє формуванню і закріпленню робочих динамічних стереотипів, а отже зменшенню м'язових і вольових зусиль. Ритмічна робота підвищує функціональні можливості організму, сприяє його тренованості і забезпечує економізацію енергетичних затрат. [1]

Багатьом програмістам постійно доводиться працювати з великою кількістю програм одночасно. Часте перемикання туди-сюди між IDE та довідкою суттєво зменшує продуктивність фахівця. Однак вирішення цієї проблеми досить просте та очевидне: встановлення більшої кількості моніторів.

Оптимальним варіантом є два монітори. Все ж таки це найпростіший з апаратної точки зору варіант. Крім того, якби їх було більше, то ними було б важче керувати, та й столі просто не вистачить місця на ще один монітор. Але тут ще залежить розміру моніторів. Є системи із 4 або 6 відносно невеликими екранами, які кріпляться на кронштейні. Але оптимальнім є два 27-дюймові монітори, на яких все добре видно, особливо коли працювати доводиться в основному з текстом [3].

8.4 Розрахункова частина

Завдання: розрахувати *штучне освітлення робочого приміщення.*

Початкові дані: ширина *робочого* приміщення: 5,4 м.; довжина –6 м.; висота – 3 м.

Розрахунок штучного освітлення проводемо за методом коефіцієнта використання світлового потоку.

Для того, щоб визначити потрібну кількість світильників, які повинні забезпечити нормований рівень освітленості, визначимо світловий потік, що падає на робочу поверхню за формулою:

$$F=ESKZ/n,$$

де: *F* – світловий потік, що розраховується, Лм;

E – нормована мінімальна освітленість, Лк; *E = 300 Лк;*

S – площа освітлюваного приміщення (у нашому випадку *S*=5,4 х 6 = 32,4 $м^2$);

Z – відношення середньої освітленості до мінімальної (зазвичай приймається рівним 1.1... 1.2, в нашому випадку *Z = 1,1*);

K – коефіцієнт запасу, що враховує зменшення світлового потоку лампи в результаті забруднення світильників в процесі експлуатації (його значення залежить від типу приміщення і характеру робіт, що проводяться в ньому, в нашому випадку *K = 1,5);*

n – коефіцієнт використання світлового потоку, (відношення світлового потоку, що падає на розрахункову поверхню, до сумарного потоку всіх ламп і обчислюється в долях одиниці; залежить від характеристик світильника, розмірів приміщення, забарвлення стін і стелі, що характеризуються коефіцієнтами відбиття від стін ($\rho_{стін.}$) і стелі ($\rho_{стелі}$), значення коефіцієнтів дорівнюють $\rho_{стін}$ = 50% і $\rho_{стелі}$ = 50% *[6]*.

Обчислимо індекс приміщення за формулою:

$$i=S/(h(A+B)),$$

де: *S* – площа приміщення, *S = 32,4* $м^2$;

h – розрахункова висота підвісу, *h* = 3 *м;*

A – ширина приміщення, А = 5,4 м;

B – довжина приміщення, В = *6* м.

Підставимо всі значення у формулу та визначемо індекса приміщення: *i=0,57.*

Знаючи індекс приміщення, за знаходимо *n = 0.29 (з табличних даних коефіцієнтів використання світлового потоку (n) світильників відповідного типу).* Підставимо всі значення у формулу, визначемо світловий потік: *F=34865 Лм.*

Для штучного освітлення приміщення використовуються *LED панель MAXUS ASSISTANCE PRO 80W 5000K WHITE (M1052480531)*, світловий потік яких $F_л$ = *8000* Лм.

Число світильників визначається по формулі:

$$N=F/F_л$$

де: F – світловий потік,

$F_л$ – світловий потік одного *світильника*.

$$N= 34865/ 8000=4,36 \text{ шт.}$$

Приймаємо необхідну кількість світильників 5 шт.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок захисного штучного освітлення. Розроблено заходи з охорони праці.

Список використаних джерел інформації

1. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

2. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2.007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

3. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

4. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508

5. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН

3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

6. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький: ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення: 16.06.2023).

7. Методичні рекомендації до виконання розділу «Заходи з охорони праці та техніки безпеки» у магістерській дисертації / Л.Д. Третякова; М-во освіти і науки України, Національний технічний університет україни «Київський політехнічний інститут» – Київ, КПІ, 2014. – 26 с. – Режим доступу до ресурсу: http://surl.li/dhulo (дата звернення: 16.06.2023).

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– Досліджена система комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

– На основі отриманих результатів досліджень створена програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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. 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 й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Threefish.

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.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 11189 грн. З урахуванням вартості розробки програми та обладнання, строк окуплення становить 0,75 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мошуренко Д.А. Дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів // Збірник праць молодих науковців ЦНТУ. – Вип. 14. – Кропивницький: ЦНТУ, 2023.

2. Pass G., Zabih R. Histogram refinement for content-based image retrieval // IEEE Workshop on Applications of Computer Vision, 1996, pp. 96-102.

3. Ma W.Y., Zhang H. Benchmarking of image feature for content-based retrieval // In IEEE 32nd Asilomar Conference on Signals, Systems, Computers, 1998, v. 1, pp. 253-257.

4. Deng Y., Manjunath B. S., Kenney Ch., Moore M. S., Shin H. An efficient color representation for image retrieval // IEEE Transactions on image processing, 2001, v. 10, no. 1, pp. 140-147.

5. Alasdair McAndrew. A Computational Introduction to Digital Image Processing. Chapman & Hall. 2021. 560 p.

6. PeterShirley, SteveMarschner. Fundamentals of Computer Graphics. 2009

7. Михайло Пічугін, Іван Канкін, Володимир Воротніков Комп'ютерна графіка. Навчальний посібник / Центр навчальної літератури 346 с. 2019р.

8. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.

9. Інженерна комп’ютерна графіка: підручник / В.В. Проців [та ін.] / М-во освіти і науки України, Нац. гірн. унт-т. – Дніпро: НГУ, 2017. – 247 с.

10. Проців В.В. Прикладна комп'ютерна графіка [Текст]: Навч. посібник / В.В. Проців, К.А. Зіборов, К.М. Бас, Г.К. Ванжа; М-во освіти і наук, Нац. гірн. ун-т. – Д.: НГУ, 2016. – 187 с.

11. Kopf, Johannes and Lischinski, Dani. Depixelizing Pixel Art (англ.) // ACM Trans. Graph.. — 2011. — Vol. 30, no. 4. — P. 99:1--99:8.

12. Giachetti, Andrea and Asuni, Nicola. Real-Time Artifact-Free Image Upscaling (англ.) // Trans. Img. Proc.. — 2011. — Vol. 20, no. 10. — P. 2760—2768.

13. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

14. 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,

15. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

16. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w

17. 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.

18. 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.

19. 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.

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

21. 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.

22. 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.

23. 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.

24. 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.

25. 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.

26. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

27. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

28. 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.

29. Zhurakovskyi, B., Tsopa, N., Batrak, Y., Odarchenko, R., Smirnova, T «Comparative analysis of modern formats of lossy audio compression». Workshop Proceedings, 2020, 2654, стр. 315-327.

30. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. Smirnov A.A., Kuzne tsov 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.

38. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». CEUR Workshop Proceedings Volume 2732, 2020, Pages 214-227.

39. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Хмарна інформаційна система оцінювання шорсткості з використанням дискретного частотного аналізу макрофотографій». IV міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 15-16 квітня 2021р. – Кропивницький: ЦНТУ. – 2021. – С. 30.

40. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у Кібербезпека та інформаційні технології: монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

41. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

42. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

43. О. Смірнов, Є. Деменко, О. Онікійчук, А. Арищенко, Л. Горбачова, «Формування псевдовипадкових послідовностей для приховування даних в зображеннях» Комп'ютерні науки та кібербезпека. № 4. С. 30-37. 2019.

44. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

45. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

46. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

47. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

48. Смірнов О.А., Кавун С.В., Коваленко О.В., Дрєєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.

49. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних

антивірусних системах. Збірник наукових праць "Системи обробки інформації". – Випуск 3 (140). – Х.: ХУПС – 2016. – С. 36-39.

50. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.

51. Смірнов О.А., Дрєєв О.М. Порівняння бітових щільностей при використанні різних методів кодування інформації. Збірник наукових праць "Системи обробки інформації". – Випуск 2 (118). т.2. – Х.: ХУПС – 2014. – С. 64-67

52. Смірнов О.А., Дрєєв О.М. Порівняння бітових щільностей при використанні різних методів кодування інформації. Збірник тез VI міжнародної науково-практичної конференції "Проблеми та перспективи розвитку IT-індустрії". м. Харків. 17-18 квітня 2014р. – Харків: ХНЄУ. – 2014. – С. 240.

53. Смірнов О.А., Коваленко О.В., Кожанова А.С., Левошко О.Л., Константинова Л.В. Основи системного програмування. Навчальний посібник. – Кіровоград: КНТУ 2013. – 257с.

54. Смірнов О.А., Дрєєв О.М., Доренський О.П. «Дослідження впливу ступеня стиснення зображень на оперативність їх доставки у телекомунікаційній системі. Збірник наукових праць "Системи обробки інформації". – Випуск 8(115). – Х.: ХУПС – 2013. – С. 234-239.

55. Смірнов О.А., Доренський О.П., Дрєєв О.М. Аналіз процесів стиснення та відновлення зображень на основі цифрових методів. Наука і техніка Повітряних сил Збройних Сил України. – Випуск 3(12). – Х.: ХУПС. – 2013. – С.122-127.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

					ВКРМ-122.23.0016.00.00.ТЗ			
Вим.	Арк.	№ документа	Підпис	Дата				
Розробив		Мошуренко Д.А.			Дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів	Літ.	Аркуш	Аркушів
Перевірив		Доренський О.П.				М	1	6
Н. Контр.		Коваленко А.С.				ЦНТУ КН-22М-1		
Затв.		Смірнов О.А.						

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 32-13 від 04.08.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) 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 з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи комп'ютерного зору для створення інтерактивних користувальницьких інтерфейсів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2023 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз умов праці на робочому місці ІТ-фахівця.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 106 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

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

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2023 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 22.12.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

_____ Доренський О.П.

Дослідження та програмна реалізація системи комп’ютерного зору для створення інтерактивних користувальницьких інтерфейсів

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2023 року

main.cpp – головний файл програми

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

//-------------------------------------------------------
/* Програма для розпізнання графічних об'єктів методом SURF.  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 зображенні (працює при динамічному
виконанні програми)
//  - 4, показати переміщення особи (працює при динамічному виконанні програми)
//  - 5, показати зміни між статичними зображеннями
*/


#define PROCEDURE 2

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

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

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

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

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

  // Виведення результату на екран
  showImage(img);

  return 0;
}

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

int mainVideo(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("Немає зображення");

  // 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 графічних об’єктів 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 графічних об’єктів методом SURF", img);

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

  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання графічних об’єктів методом SURF" );
  return 0;
}


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


int mainMatch(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("Немає зображення");

  // 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 графічних об’єктів 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 графічних об'єктів методом SURF", img);

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

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання графічних об'єктів методом SURF" );
  return 0;
}


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


int mainMotionPoints(void)
{
  // Ініціалізація пристрою отримання картинки
  CvCapture* capture = cvCaptureFromCAM( CV_CAP_ANY );
  if(!capture) error("Немає зображення");

  // Створюємо вікно
  cvNamedWindow("Програма розпізнання графічних об'єктів 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 графічних об’єктів методом SURF", img);

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

  // Редагуємо зображення з пристрою
  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання графічних об’єктів 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, що здійснює ідентифікацію об'єктів на відеозображенні

```
/*************************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*************************************************************/

#include "utils.h"

#include "surf.h"

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

const double gauss25 [7][7] = {

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

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

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

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

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

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

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

const double gauss33 [11][11] = {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
  int idx = 0;
  // розраховуємо відповідні для точок Хаара в межах радіусу 6*масштаб
  for(int i = -6; i <= 6; ++i)
  {
    for(int j = -6; j <= 6; ++j)
    {
      if(i*i + j*j < 36)
      {
        gauss = static_cast<float>(gauss25[id[i+6]][id[j+6]]);
        resX[idx] = gauss * haarX(r+j*s, c+i*s, 4*s);
        resY[idx] = gauss * haarY(r+j*s, c+i*s, 4*s);
        Ang[idx] = getAngle(resX[idx], resY[idx]);
        ++idx;
      }
    }
  }

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

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

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

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

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

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

//! Беремо модифікований дескріптор.

void Surf::getDescriptor(bool bUpright)
{
  int y, x, sample_x, sample_y, count=0;
  int i = 0, ix = 0, j = 0, jx = 0, xs = 0, ys = 0;
  float scale, *desc, dx, dy, mdx, mdy, co, si;
```

```
float gauss_s1 = 0.f, gauss_s2 = 0.f;
float rx = 0.f, ry = 0.f, rrx = 0.f, rry = 0.f, len = 0.f;
float cx = -0.5f, cy = 0.f; //Підобласть зосереджується для 4x4 блока гауса

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

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

i = -8;

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

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

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

    j = j - 4;

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

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

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

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

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

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

      }
    }
```

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

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

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

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

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

}


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

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

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

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

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

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

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

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

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

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

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

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

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

  return 0;
}
```

surf.h – файл заголовків

```
/*************************************************************
* --- Програма розпізнання графічних об'єктів методом SURF --- *
*************************************************************/

#ifndef SURF_H
#define SURF_H

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

#include <vector>

class Surf {

public:

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

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

private:

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

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

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

//! Розраховуємо значення в  2d 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 розпізнання графічних об'єктів методом SURF --- *
*
* *
*********************************************************/

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

#include "utils.h"

using namespace std;

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

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

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

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

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

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

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

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

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

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

IplImage* gray8, * gray32;

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

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

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

cvReleaseImage( &gray8 );
return gray32;
}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}

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

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

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

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

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

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

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

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

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


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

}
}

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

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

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

```
counter = 0;
}

// Інкрементуємо лічильник
++counter;

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

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

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

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

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

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

outfile << "\n";
}

outfile.close();
}

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

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

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

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

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

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

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

ipts.push_back(ipt);

}
}

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

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

utils.h – заголовочний файл

```
/*********************************************************
* --- Програма розпізнання графічних об'єктів методом SURF --- *
*********************************************************/

#ifndef UTILS_H
#define UTILS_H

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

#include <vector>


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

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

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

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

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

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

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

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

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

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

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

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

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

#endif
```

ipoint.cpp – підпрограма визначення базових точок

```
/*************************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*
*
*************************************************************/

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

#include "ipoint.h"

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

  matches.clear();

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

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

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

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

//
// Ця функція користується CV_RANSAC (
//

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

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

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

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

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

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

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

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

ipoint.h – заголовочний файл

```
/*********************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*
*                                                        *
*********************************************************/

#ifndef IPOINT_H
#define IPOINT_H

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

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

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

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

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

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

class Ipoint {

public:

  //! деструктор
  ~Ipoint() {};

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

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

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

  //! Визначає градацію
  float scale;

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

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

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

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

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

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


#endif
```

integral.cpp – робота з відеозображенням

```
/*************************************************************
* --- Програма розпізнання графічних об'єктів методом SURF --- *
* *
*************************************************************/

#include "utils.h"

#include "integral.h"

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

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

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

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

// звільняємо сіре зображення
cvReleaseImage(&img);

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

integral.h – заголовочний файл

```
/*************************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*
*                                                            *
*************************************************************/

#ifndef INTEGRAL_H
#define INTEGRAL_H

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

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

#ifdef max
  #undef max
#endif

#include <cv.h>

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


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

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

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

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

#endif
```

fasthessian.cpp – реалізація гессіана

```
/*********************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*                                                        *
*********************************************************/

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

#include <vector>

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



using namespace std;

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

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

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

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

  // Встановлюємо поточне зображення
  setIntImage(img);
}

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

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

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

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

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


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

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

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

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

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

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

  // Будуємо карту відповідностей
  buildResponseMap();

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  return 1;
}

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

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

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

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

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

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

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

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

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

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

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

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

  dI = cvCreateMat( 3, 1, CV_64FC1 );
  cvmSet( dI, 0, 0, dx );
  cvmSet( dI, 1, 0, dy );
  cvmSet( dI, 2, 0, ds );
```

```
  return dI;
}

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

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

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

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

  return H;
}

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

fasthessian.h – заголовочний файл

```
/*************************************************************
* --- Програма розпізнання графічних об'єктів методом SURF --- *
* *
*************************************************************/

#ifndef FASTHESSIAN_H
#define FASTHESSIAN_H

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

#include <vector>

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


class FastHessian {

public:

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

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

//! Деструктор
~FastHessian();

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

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

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

private:

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

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

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

//! 3x3x3 тест на екстремум
```

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

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

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

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

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

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

//! Число октетів
int octaves;

//! Число інтервалів між октетами
int intervals;

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

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


#endif
```

responselayer.h – заголовочний файл для шару відповідностей

```
/*************************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*************************************************************/


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

class ResponseLayer
{
public:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

kmeans.h – заголовочний файл

```
/*************************************************************
*  --- Програма розпізнання графічних об'єктів методом SURF ---
*
*************************************************************/

#include "ipoint.h"

#include <vector>
#include <time.h>
#include <stdlib.h>

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

class Kmeans {

public:

  //! Деструктор
  ~Kmeans() {};

  //! Конструктор
  Kmeans() {};

  //!
  void Run(IpVec *ipts, int clusters, bool init = false);

  //! Встановлюємо ipts до використання
  void SetIpoints(IpVec *ipts);

  //! Випадково поширюйте ''n'' кластерів
  void InitRandomClusters(int n);

  //! Призначемо Ipoints кластерам
  bool AssignToClusters();

  //! Розраховуємо нові центри кластерів
  void RepositionClusters();

  //! Функція вимірює відстань між  2 ipoints
  float Distance(Ipoint &ip1, Ipoint &ip2);

  //! Запам'ятовуємо вектор ipoints для цього руху
  IpVec *ipts;

  //! Запам'ятовуємо вектор центрів кластерів
  IpVec clusters;

};

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

void Kmeans::Run(IpVec *ipts, int clusters, bool init)
{
  if (!ipts->size()) return;

  SetIpoints(ipts);

  if (init) InitRandomClusters(clusters);

  while (AssignToClusters());
  {
    RepositionClusters();
  }
```

```
}

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

void Kmeans::SetIpoints(IpVec *ipts)
{
  this->ipts = ipts;
}

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

void Kmeans::InitRandomClusters(int n)
{
  // очищуємо вектор кластеру
  clusters.clear();

  // Запускаємо генератор випадкових чисел
  srand((int)time(NULL));

  // додаємо 'n' випадкових ipoints до списку кластерів й ініціалізуємо центри
  for (int i = 0; i < n; ++i)
  {
    clusters.push_back(ipts->at(rand() % ipts->size()));
  }
}

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

bool Kmeans::AssignToClusters()
{
  bool Updated = false;

  // цикл над усіма Ipoints і призначаємо кожну найближчому кластеру
  for (unsigned int i = 0; i < ipts->size(); ++i)
  {
    float bestDist = FLT_MAX;
    int oldIndex = ipts->at(i).clusterIndex;

    for (unsigned int j = 0; j < clusters.size(); ++j)
    {
      float currentDist = Distance(ipts->at(i), clusters[j]);
      if (currentDist < bestDist)
      {
        bestDist = currentDist;
        ipts->at(i).clusterIndex = j;
      }
    }

    // визначаємо чи змінила точка кластер
    if (ipts->at(i).clusterIndex != oldIndex) Updated = true;
  }

  return Updated;
}

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

void Kmeans::RepositionClusters()
{
  float x, y, dx, dy, count;

  for (unsigned int i = 0; i < clusters.size(); ++i)
  {
    x = y = dx = dy = 0;
    count = 1;

    for (unsigned int j = 0; j < ipts->size(); ++j)
    {
      if (ipts->at(j).clusterIndex == i)
```

```
      {
        Ipoint ip = ipts->at(j);
        x += ip.x;
        y += ip.y;
        dx += ip.dx;
        dy += ip.dy;
        ++count;
      }
    }

    clusters[i].x = x/count;
    clusters[i].y = y/count;
    clusters[i].dx = dx/count;
    clusters[i].dy = dy/count;
  }
}

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

float Kmeans::Distance(Ipoint &ip1, Ipoint &ip2)
{
  return sqrt(pow(ip1.x - ip2.x, 2)
            + pow(ip1.y - ip2.y, 2)
            /*+ pow(ip1.dx - ip2.dx, 2)
            + pow(ip1.dy - ip2.dy, 2)*/);
}

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