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

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи машинного зору для контролю якості на виробництві”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-3СК
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Коджебаш В.С.
«____» _____ 2024 р.

Керівник проекту
канд. фіз.-мат. наук, доцент
_____ Якименко Н.М.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2024 року

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

Коджебашу Володимиру Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи машинного зору для контролю якості на виробництві*

2. Керівник роботи *Якименко Наталія Миколаївна, канд. фіз.-мат. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 132-02 від 01.04.2024 року

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

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи машинного зору для контролю якості на виробництві*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

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

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

7. Дата видачі завдання « 17 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

Підпис керівника
_____ Якименко Н.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2024 р.

Підпис здобувача
_____ Коджебаш В.С.
(прізвище та ініціали)

АНОТАЦІЯ

Коджебаш В.С. Програмне забезпечення системи машинного зору для контролю якості на виробництві. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи машинного зору для контролю якості на виробництві.

Метою розробки є програмне забезпечення системи машинного зору для контролю якості на виробництві.

Результат роботи – програмна реалізація системи машинного зору для контролю якості на виробництві.

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

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

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

Програму розроблено в середовищі Visual C++.

Ключові слова: комп'ютерна інженерія, машинний зір

ABSTRACT

Kodzhebash V.S. Machine vision software for quality control in production. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the machine vision system for quality control in production.

The purpose of the development is the software of the machine vision system for quality control in production.

The result of the work is the software implementation of the machine vision system for quality control in production.

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

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

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

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

Keywords: computer engineering, machine 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. Як вважають експерти, відновлення промислової інфраструктури й розширення спектра додатків з урахуванням потреб держави й частки бізнесу приведуть до росту ринку систем і технологій машинного зору. Тому зараз самий час об'єднати зусилля українських і міжнародних компаній, державних і науково-дослідних підприємств і інститутів для розвитку української індустрії машинного зору.

До основних областей застосування машинного зору відносять контроль якості продукції, моніторинг і ідентифікацію об'єктів, керування технологічними процесами. Сучасне ПЗ для машинного зору реалізує сотні алгоритмів обробки, забезпечує обмін даними із промисловими ПЛК і ПК, здійснює відображення результатів і підтримує прийняття рішень на основі дані інспекції.

Машинний зір звичайно застосовується для керування якістю, при обліку сировини й матеріалів і для контролю за їхнім рухом, що в остаточному підсумку сприяє скороченню витрат і збільшенню обсягів виробництва. Воно забезпечує прискорене проведення інспекцій і підвищення їхньої якості, крім при цьому людський фактор: візуальний контроль заміняється на більше ефективну систему машинного зору. Причому автоматичний контроль можна впроваджувати на всіх етапах виробництва – як відомо, чим раніше виявляється дефект, тим дешевше обходиться його виправлення.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи машинного зору для контролю якості на виробництві.

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

– Огляд існуючих систем машинного зору для контролю якості на виробництві.

– Дослідження системи машинного зору для контролю якості на виробництві.

– Програмна реалізація системи машинного зору для контролю якості на виробництві.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі машинного зору для контролю якості на виробництві.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи машинного зору для контролю якості на виробництві, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система машинного зору дозволяє збільшити кількість контрольованих параметрів, проводити перевірку дуже дрібних деталей, вести статистику відхилень від норми для оптимізації процесу, вимірювати геометрію й обсяги об'єктів складних форм, здійснювати контроль маркування або підрахунок виробів із заданими характеристиками. У результаті вдається виявити брак на ранніх стадіях, забезпечити гарантовану якість продукції, скоротити простої й оптимізувати виробництво на підставі статистичних даних. Якість можна не тільки контролювати, але й управляти нею. Отримані чисельні значення й статистичні дані спрощують діагностику й усунення проблем, облік і контроль, підрахунок і сортування виробів.

Системи машинного зору розрізняються за вартістю й складністю, а обчислювальний модуль і камера можуть бути сполучені або розділені. Смарт-системи наділяються «інтелектом» – наприклад, можуть підраховувати кількість виробів.

У цих системах застосовуються різні типи процесорів: DSP, FPGA (або програмувальні логічні контролери, ПЛК), x86 або ARM, а також комбіновані варіанти. До останнього відносяться DSP+ARM (наприклад, TI KeyStone Multicore DSP), FPGA + ARM (XILINX Zynq), x86 + GPU (Intel, AMD), ARM + GPU (NVIDEA, Samsung, QUALCOMM і ін.). По сумі споживчих властивостей фахівці найвищу оцінку дають процесорам DSP, а також ARM у сполученні із графічними процесорами GPU. Привабливим рішенням для попередньої обробки зображень уважаються FPGA-камери. Технологія FPGA дозволяє програмувати такі камери для різних прикладних завдань.

Важливу роль у реалізації різних систем машинного зору грає вибір підходящої оптики. Наприклад, на високошвидкісних виробничих лініях можна використовувати бітелецентричний об'єктив з коллімірованим освітлювачем. Одержуване з його допомогою чітке зображення по краях об'єкта спрощує проведення точних вимірів і виявлення дефектів. Катадиоптрічний об'єктив дає повне (3600) зображення невеликих об'єктів. Його можна легко інтегрувати практично в будь-яку систему, використовувати для контролю кольору, перевірки точності різьблення пластикових пляшок, герметизації фіалів і т.п.

Спеціальна оптика використовується й при виробництві тонких і довгих об'єктів, наприклад медичних шприців. Існуючі системи забезпечують можливість огляду зовнішньої й внутрішньої поверхні об'єкта в одному зображенні високого дозволу. При рішенні деяких завдань макрооб'єктиви заміняють мікроскопи, дозволяючи вимірювати об'єкти з мікронною точністю. В одній системі найчастіше застосовується кілька камер машинного зору.

1.2 Область застосування

Сучасний мир комп'ютерних систем складно представити без технологій машинного, або комп'ютерного, зору. Нарощуючи свій науковий і практичний потенціал паралельно з удосконалюванням обчислювальної й техніки, що реєструє, комп'ютерний зір поступово завойовує всі нові технологічні рубежі. Високопродуктивні обчислювальні машини останнього покоління (до них відносяться й сучасні персональні комп'ютери) уже дозволяють вирішувати багато завдань обробки потоків цифрової відеоінформації й ухвалення рішення в режимі реального часу. І сьогодні, часом непомітно для більшості з нас, комп'ютерний зір досить міцно закріплюється в багатьох областях життєдіяльності людини, допомагаючи йому, а часом заміняючи його, рятуючи від монотонного, рутинного або, нерідко, пов'язаного з ризиком для життя праці.

Ні для кого не секрет, що комп'ютерний зір як технологія одержав найбільш широкий, повний і всебічний розвиток на Заході, особливо в США, у Південній Кореї й у Японії. Зв'язано це насамперед з потужною фінансовою підтримкою цього напрямку з боку уряду й інвесторів, що прогнозують за ним велике майбутнє. Причому уряд в основному підтримує розвиток технології в загальноосвітніх центрах, а інвестори забезпечують підтримку приватним високоперспективним компаніям. Найбільш яскравими прикладами таких добре фінансованих наукових центрів можуть служити Лабораторія Штучного Інтелекту Массачусетсского Технологічного Інституту (MIT Artificial Intelligence Laboratory), UC Berkeley Computer Vision Group, Vision and Autonomous Systems Center Університету Корнегі-меллона, Stanford Vision Laboratory і ряд інших. Прикладами підтримуваних приватних компаній можуть служити такі компанії, як Visionics, Eyematic і ін. Усього на Інтернет-сайті, що поєднує розроблювачів в області машинного зору, – Computer Vision Home Page зареєстровано близько 200 груп і наукових лабораторій, що працюють над даною проблематикою. Слід зазначити, що цим не вичерпується коло організацій, що займаються комп'ютерним зором, тому що існує велика кількість комерційних фірм, що спеціалізуються в області машинного зору й обробки зображень. Інформацію про їх можна знайти на спеціалізованих тематичних Інтернет-сайтах, присвячених окремим напрямкам даної технології. Іншими словами, розроблювачі різних технологій усередині самої технології комп'ютерного зору як би поєднуються в клуби по інтересах. Наприклад, що цікавляться досягненнями в області розпізнавання жестів можуть знайти досить докладну інформацію про дослідження, дослідницьких групах, комерційних додатках, патентах на відповідному спеціалізованому Інтернет-сайті – Gesture Recognition Home Page. Там же можна скачати деякі демонстраційні додатки й ознайомитися з останніми науковими публікаціями. Якщо ж читач воліє зайнятися технологіями, пов'язаними з розпізнаванням осіб, то йому пряма дорога у віртуальний клуб на іншому Інтернет-сайті – Face Detection and Recognition Home Page.

Слід зазначити, що все перераховане вище приведе до швидкого росту й удосконалювання технологій комп'ютерного зору. У цей час закордонні науково-дослідні й комерційні центри залучають велику кількість учених і висококваліфікованих програмістів, проводять розпаралелювані дослідження в різних областях машинного зору, домагаючись досить вагомих результатів.

Україна, як повноправний член світового економічного співтовариства, не залишилася осторонь від цього процесу. От уже кілька років на українському технологічному ринку також спостерігається тенденція підвищення інтересу до проблем комп'ютерного зору, причому як з боку керівників ряду IT-компаній і компаній, що працюють на ринку безпеки, так і з боку споживачів (користувачів) і студентів, що бажають спеціалізуватися в цій області. Реакцією на цей інтерес стала поява лабораторій, груп і комерційних структур, що відносять перед собою завдання розробки різного роду технологій і додатків для рішення проблем машинного зору. І якщо ще десятиліття назад ми були в ролі що доганяють, то на сьогоднішній день багато компаній – лідери в області передових технологій прагнуть на український ринок з метою придбання відповідних технологій комп'ютерного зору або розміщення замовлень на передові дослідження й розробки в цій області.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи машинного зору для контролю якості на виробництві, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Дослідження українського ринку розроблювачів технології машинного зору показує, що кількість фірм, що займаються комп'ютерним зором, відносно невелика. Розглянемо найбільш помітні із цих компаній і приведемо короткий опис деяких цікавих технологій комп'ютерного зору, які поставляються ними на вітчизняний і світовий ринки.

SPIRIT

Поряд з такими активно затребуваними в наші дні технологіями, як телефонія, обробка й розпізнавання мови, GPS-Технології, передача даних і т.д., компанія активно займається дослідженням, розробкою й просуванням комерційних додатків в області цифрової обробки й аналізу зображень. Примітним є той факт, що тільки за останні три роки компанія одержала охоронні грамоти на шість винаходів в області машинного зору.

Створена в рамках компанії лабораторія комп'ютерного зору – CV Lab – проводить R&D у наступних технологічних напрямках: біометричні системи ідентифікації й автентифікації користувача на основі вхідного відеопотоку, системи спостереження за особою людини як за просторовим об'єктом, системи розпізнавання жестів, системи детектування наявності руху й аналізу області руху на предмет класифікації об'єкта, що рухається, і ще ряд перспективних технологічних напрямків. У своїх розробках лабораторія опирається на аналіз перспективних потреб світового ринку у високонадійних і високоточних системах комп'ютерного зору. Тому більшості робіт, проведених у лабораторії,

передує глибокий маркетинговий аналіз. Серед клієнтів лабораторії ряд відомих закордонних компаній, таких як Toshiba, Panasonic, Samsung.

Окремо хочеться відзначити роботи компанії в області розробки систем розпізнавання міміки особи в реальному масштабі часу. Цю технологію, а також технологію спостереження за особою компанія ліцензувала фірмі SeeStorm для побудови систем відеоконференцій нового покоління. Основна ідея робіт у цьому напрямку – управляти аватаром (штучним тривимірним об'єктом) на підставі аналізу поводження людини перед комп'ютером. При цьому відбувається розпізнавання положення людини в просторі, визначення кутів повороту його голови, детектується його міміка. Ця інформація кодується й передається по мережі на приймаючу сторону. У такий спосіб досягається істотне зменшення інформаційного потоку в порівнянні із традиційними відеоконференціями, підвищується конфіденційність. Акцент у розробках зроблений на використання так званих Web-камер для керування аватаром. Іншими словами, користувачеві необхідний комп'ютер, гарнітура (навушники з мікрофоном) і Web-камера для того, щоб поринути в мир віртуального спілкування.

Поряд із цим напрямком у лабораторії проводяться серйозні дослідження й розробки в області створення біометричних систем, заснованих на аналізі зображень особи людини. Зокрема, у результаті багаторічних досліджень була розроблена унікальна технологія, що дозволяє системі приймати рішення про ідентифікацію в складних умовах яркістно-контрастної мінливості вхідних зображень, протидіяти пред'явленню фотографій, ураховувати ефект часткового загоражування області особи, протистояти зміні в зачісці, макіяжі й ряді інших факторів, що впливають. Технологія розроблялася й будувалася як альтернатива існуючим системам розпізнавання особистості, таким як FaceIt (Visionics, Inc.), BioID (DCS) і т.д. Ґрунтуючись на особливій інтегральній інформації про особу, вона формує його унікальне признаковий опис, що згодом використовується як біометричний ключ (або коду) для ухвалення рішення про розпізнавання.

Оригінальна система ухвалення рішення, емулююча ухвалення рішення людиною, дозволяє додатково підвищити характеристики надійності.

Рисунок 2.1 – Інтерфейс системи ідентифікації особистості FaceMe

Паралельно із цими розробками лабораторія проводить роботи в області створення системи розпізнавання особистості на основі тривимірних вимірів. Використання тривимірного рельєфу особи, відновлюваного по зображеннях, підвищує надійність системи безпеки, дозволяючи вирішувати завдання розпізнавання особистості з більше високими показниками надійності. Дана технологія припускає наявність спеціального стереокомплексу, що складається із двох камер, підсвічування й пристрої позиціонування особи.

Особливе місце в технологічному ряді фірми займають програми, призначені для розпізнавання жестів у реальному масштабі часу. Ця технологія

насамперед цікава своїми потенційними додатками, до яких відносяться: керування курсором PC за допомогою жестів, дистанційне керування в ігрових додатках, керування роботами, дистанційне керування побутовою технікою (телевізорами, відеомагнітофонами та ін.) замість пультів керування, запуск комп'ютерних додатків на основі розпізнавання визначених жестів, інтерпретатор жестів для глухонімих і нездатних до пересування людей (лежачі хворі) і т.п.

ІІТ

Основні технології, якими займається лабораторія ІІТ, включають: обробку зображень і аналіз сцен, ближню й далеку фотограмметрію, тривимірну візуалізацію й віртуальну реальність і ряд інших напрямків.

Найцікавішої й затребуваної як на українському, так і на міжнародному ринку технологією є технологія пошуку й зчитування штрихових кодів. У даній області, відповідно до отриманої інформації, лабораторією ІІТ досягнуті особливо високі результати. Суть даної технології полягає в наступному. Зображення об'єкта з нанесеним штриховим кодом реєструється відеокамерою й уводиться в персональний комп'ютер. У ході аналізу отриманого цифрового зображення наявні штрихові коди виявляються й зчитуються. Програмне забезпечення дозволяє детектувати, зчитувати й декодувати всі розповсюджені на сьогоднішній день штрихові коди, незалежно від їхнього розміру, положення на знімку й орієнтації. При цьому поверхня, на яку вони наносяться або можуть наноситися, може бути похилої, криволінійної або навіть зім'ятої, коди можуть бути частково забруднені або затерті, а прозоре впакування поверх кодів не є перешкодою для їхнього стійкого виявлення й розпізнавання. Коди розпізнаються на будь-якому складному тлі: програма легко відрізняє штрихові коди від будь-яких інших різновидів тексту й графіки, нанесених на об'єкт або навколишніх його.

Рисунок 2.2. – Результат роботи програмного забезпечення з розпізнаванню штрих-кодової інформації

Іншою сферою інтересів групи є розробка технології розпізнавання машиночитаемої інформації. Під машиночитаемою інформацією розуміються буквено-цифрові послідовності (машиночитаемі зони), що володіють спеціальною структурою, що забезпечує за рахунок надлишкового кодування й запису контрольних сум істотно більше високу вірогідність зчитування

інформації. При цьому, залежно від призначення, змістовна буквено-цифрова інформація може бути читаємою як (з метою візуального контролю), так і нечитаємою без спеціального декодера (у конфіденційних додатках). Основними достоїнствами розробленої технології є: інваріантність алгоритмів до поворотів зчитувального документа, стійке розпізнавання машиночитаемих символів при низькому дозволі (до 170-200 крапок на дюйм) і можливість навчання системи різним типам машиночитаемих шрифтів при розробці спеціалізованих додатків.

Продуктова лінійка лабораторії включає наступні розробки: систему аналізу й обробки зображень для персонального комп'ютера серії Pisoft Image Framework і цифрову фотограмметричну систему серії Z_Space. Перша адресована розроблювачам систем обробки зображень, а також може використовуватися для практичних, дослідницьких і навчальних цілей як інтегроване середовище роботи із зображеннями. Друга система призначена для одержання цифрової моделі рельєфу на основі стереопари цифрових знімків, створення ортофотопланоі і візуалізації елементів рельєфу з використанням тривимірної графіки.

CVision

Компанія орієнтована на створення програмних продуктів і рішень у наступних областях: керування виробництвом, медицина, відеоспостереження й векторизація зображень. Розглянемо найцікавіші розробки CVision з наявних матеріалів на її офіційному сайті. Відразу хочеться відзначити, що основне число проектів, представлених на згаданому сайті, розроблені для режиму реального часу.

Почати огляд продуктів і технологій даної компанії хочеться з опису двох додатків, пов'язаних з обробкою й аналізом медичних зображень. Перший додаток (або технологія) призначено для так званої системи сортування цифрової рентгенівської інформації. Зокрема, що поставляється додаток XraySort розроблено для класифікації рентгенівських медичних знімків. Відмінними рисами даної технології є те, що в ній досягаються високі характеристики

швидкодії й стійкість алгоритму до неповних зображень (тобто утримуюча лише частина шуканої інформації). Крім того, алгоритми дозволяють виявляти в організмі області, що мають відхилення від норми. Другий додаток – XraySortQuery – розроблено лабораторією для формування запитів у базах даних рентгенівських знімків.

Рисунок 2.3 – Інтерфейс програми XraySor

Запити для такої системи можуть бути зроблені в анатомічних термінах, а пошук і локалізація необхідних об'єктів на зображеннях можуть бути здійснені безпосередньо в процесі виконання запиту. Звичайний запит може мати такий вигляд: «дайте рентгенівські знімки гомілки з імплантованим стрижнем у верхній частині» або «дайте рентгенівські знімки черепа, де є присутнім половина особи» і т.п.

Поряд із цим лабораторія володіє різними технологіями розпізнавання символів, починаючи з розпізнавання номерних знаків і закінчуючи читанням символів на різних поверхнях об'єктів, зокрема для розпізнавання маркування чипів на складальній лінії. Відповідно до заяв розроблювачів цієї технології, вона може бути використана не тільки для розпізнавання номерів автомобілів і серійних номерів, але й для автоматичного розпізнавання друкованих текстів. Правда, під сумнівом залишається стабільність даної технології до різних шрифтів.

На закінчення хотілося б відзначити ще два продукти (технології), що поставляються даною групою. Це система сортування листів MailSort і технологія виявлення й аналізу руху.

Система сортування листів MailSort призначена для класифікації зображень поштових конвертів. Завдання полягає в пошуку підпослідовності поштових конвертів однакового типу в загальному потоці конвертів і ідентифікації відповідної поштової скриньки з подібними конвертами. Приналежність конверта до деякої поштової скриньки визначається по логотипах, печаткам і відповідним написам, а також по їхньому розташуванню відносно один одного на конверті. Якщо конверт не належить до жодного з існуючих класів, то створюється новий клас. У результаті роботи додатка вхідний набір зображень конвертів класифікується по поштових скриньках. Додаток візуалізує процес класифікації зображень і дозволяє переглянути кожну поштову скриньку по завершенні класифікації.

Технологія виявлення й аналізу руху розроблявся CVision для рішення завдань, пов'язаних з питаннями побудови систем безпеки. Завдання виявлення руху й спостереження за об'єктом складалася у виділенні на послідовності відеокадрів об'єкта, що рухається, і фіксації змін сцени (зони спостереження камери) протягом усього часу, поки об'єкт перебував у зоні видимості камери, до моменту, коли він цю зону покинули. Така технологія дозволяє виявити не тільки рух людини в заданому полі зору камери, але й, наприклад, зникнення якогось предмета, що перебував у зоні інтересу.

«Модуль»

В області систем комп'ютерного зору компанія активно проводить і просуває свої розробки на ринку так званих інтелектуальних транспортних систем. На сьогоднішній день із погляду сучасних систем комп'ютерного зору найцікавішими технологіями й продуктами, наявними в арсеналі компанії, є наступні:

– апаратно-програмний комплекс виміру характеристик транспортного потоку «Трафік-монітор»;

– система класифікації автомобілів;

– система розпізнавання дорожньої розмітки;

– система допомоги водієві автомобіля по запобіганню дорожньо-транспортних випадків, що по стереозображенню визначає дорожнє полотно, розмітку й перешкоди.

Розглянемо ці продукти докладніше.

«Трафік-монітор» – це компактний високопродуктивний обчислювач, виконаний в одному корпусі з відеокамерою й джерелом вторинного електроживлення, що має стандартний зовнішній інтерфейс, до якого можна підключатися по бездротовому або звичайному модемі. Один такий пристрій дозволяє здійснювати в реальному масштабі часу контроль до шести смуг руху, передаючи в центр керування рухом накопичену інформацію про кількість транспортних засобів, їхньому типі, швидкості, дистанції між ними й ступеня

завантаженості дороги. «Трафік-монітор» здатний розпізнавати п'ять типів транспортних засобів: мотоцикл, легковий автомобіль, вантажівка-мікроавтобус, автобус, довга вантажівка-трейлер. Дана система побудована на базі власної розробки НТЦ – сигнального процесора Л1879ВМ1. Цим обумовлена її більша гнучкість і універсальність у порівнянні з іншими аналогічними системами, тому що її можна досить легко перепрограмувати на рішення будь-яких інших завдань інтелектуальної обробки відеоінформації в режимі реального часу, наприклад для використання як інтелектуальний датчик у складі охоронних систем.

Іншим цікавим, на мій погляд, проектом НТЦ «Модуль» є розроблювальні система автоматичного визначення типу й вантажопідйомності автомобіля за інформацією від відеокамери й система розпізнавання дорожньої розмітки для запобігання дорожньо-транспортних випадків. Перша система розробляється для німецької компанії AGES Maut System, друга – для італійського автомобільного концерну FIAT. Система розпізнавання дорожньої розмітки повинна встановлюватися на машину й виконувати функції базового елемента системи допомоги водієві. У коло завдань системи так званий круїз-контролю будуть входити завдання моніторингу дистанції між даною машиною й іншими машинами в потоці (спереду, позаду й збоку) і інформування водія про небезпеку зіткнення. Особливої уваги по праву заслуговують технологія й діючий прототип системи запобігання дорожньо-транспортних випадків. В основу цієї перспективної системи закладений бінокулярний відеодатчик. Він розташований на транспортному засобі й здійснює в режимі реального часу наступні завдання: стереомоніторинг дорожнього полотна з метою виявлення статичних перешкод, стереомоніторинг розмітки й навколишніх транспортних засобів. Слід зазначити, що макетний зразок, що демонструвався на міжнародних виставках, даної системи одержав високі оцінки фахівців, що розробляють інтелектуальні транспортні системи майбутнього. Більше того, відповідно до наявної інформації, ряд автомобільних гігантів виразив готовність приступитися до тестування пілотних зразків такої системи.

Оптичне Розпізнавання Об'єктів

Для того щоб скласти більше ясне подання про рівень і глибину технологій, що поставляються на український і міжнародний ринки НПЦ ОРО, зупинимося докладніше на продукті КРИМНЕТ, що пропонують у якості готової біометричної системи. За наявною інформацією, з технологічної точки зору програмне забезпечення розглянутої біометричної системи складається із двох частин: модуля розпізнавання зображень і незалежного модуля пошуку інформації в базі даних по словесному описі зовнішності з наступним виводом цієї інформації на екран користувача.

Відповідно до заяв розроблювачів, алгоритми модуля розпізнавання зображень ГАБІТУС дозволяють одержати саме той результат, що близький асоціативному людському мисленню, у той час як більшість інших систем засновано на формальних статистичних обчисленнях. При цьому процес розпізнавання зводиться до пошуку найбільш схожого еталона (зображення, що зберігається в пам'яті нейронної мережі). Коли еталон заноситься на згадку, то між ним і вже збереженими в пам'яті комп'ютера зображеннями будуються асоціативні зв'язки у вигляді рівнів взаємної подібності, позитивно впливаючи на результат розпізнавання. Дана технологія дозволяє обробляти більші обсяги інформації у вигляді фотографій і фотороботів і призначена для пошуку осіб, найбільш схожих на пропоноване. При цьому ефективність пропонованого методу дає можливість реалізувати її на простих настільних PC без залучення потужних обчислювальних ресурсів.

Модуль словесного опису також є оригінальною розробкою НПЦ ОРО. Він дозволяє врахувати практично всі можливі випадки наявності або відсутності додаткової якісної інформації про те, як виглядає ідентифікуємий. Опис зовнішності будується на базі близько 300 параметрів, які використовуються для спрощення процедури розпізнавання.

Принцип дії КРИМНЕТ простий. Користувач системи вводить зображення невідомої особи й/або описову інформацію через спеціальну форму запиту, що по

локальній мережі або модемному зв'язку відправляється на сервер. Сервер спочатку відбирає зображення відповідно до описової інформації, а потім ГАБІТУС робить розпізнавання й видає обмежену ієрархію зображень зареєстрованих осіб на екран користувача. Відмінною рисою даного продукту (технології) від інших біометричних систем є те, що вона дозволяє здійснювати процедуру ідентифікації по фотороботі.

За офіційною інформацією, доступної на Інтернет-сторінці компанії, до теперішнього часу створені дві версії КРИМНЕТ: потужна система для баз даних до 500 000 зображень і полегшена версія для 50 000 зображень.

Як приклад використання своїх технологій НПЦ ОРО пропонує розважальне програмне забезпечення – Analogia. Суть даної програми полягає в пошуку найбільш схожих на користувача знаменитостей. Одержавши свою цифрову фотографію у фас за допомогою Web-камери, сканера або будь-яким іншим способом, кожний може підібрати собі «двійника» з миру зірок кіно, естради й т.п. Причому цю процедуру можна проробити в інтерактивному режимі з Інтернет-сторінки компанії на звичайному персональному комп'ютері.

Відеотест

Фірма Відеотест займається комплектацією й поставками комп'ютерних систем аналізу зображень, розробкою програмного забезпечення для аналізу зображень, впровадженням методик комп'ютерного аналізу в медицині, біології, геології, матеріалознавстві, криміналістиці й інших областях. Основне завдання, що ставить перед собою колектив компанії, – створення програмного забезпечення для роботи із цифровими зображеннями з метою їхнього перетворення, аналізу й архівування. Потреба в такого роду системах існує в багатьох сферах науково-дослідної діяльності, наприклад таких, як медицина й біологія, матеріалознавство, наука про землю та ін. Застосування комп'ютерних систем аналізу зображень піднімає роботу на сучасний рівень, роблячи її більше продуктивної, а результати – статистично достовірними.

Для цих цілей компанія пропонує набір програмних продуктів – так званих аналізаторів зображень.

Аналізатори зображень являють собою апаратно-програмні комплекси для рішення завдань, пов'язаних з уведенням, перетворенням і аналізом кольорових або чорно-білих цифових зображень. Такі комплекси призначені для проведення різних морфометричних вимірів і досліджень. Вони застосовуються в основному в медицині й біології й служать для зміни візуалізації вихідних зображень шляхом різного роду перетворень, проведення аналізу й вимірів на цих зображеннях і архівування. Типовими прикладами розв'язуваних за допомогою даного комплексу завдань є морфометрія гістологічних зрізів, аналіз патологічних змін кліток крові, оцінка ядерно-клітинного (цитоплазменного) відносини, побудова й аналіз еритроцитарної гістограмми (наприклад, для уточнення діагностики природи анемії), підрахунок і аналіз тромбоцитів, цитофотометрія, аналіз поведінкових особливостей тварин (траєкторія руху, швидкість, прискорення) і інші медичні додатки. В області матеріалознавства такими прикладами є гранулометричний аналіз, аналіз тріщинуватості, фазовий аналіз і т.п.

«Промінформ»

Компанія «Промінформ» відома своїм апаратно-програмним комплексом «Сова-2», у який входить технологія машинного зору.

Основне призначення комплексу – автоматична ідентифікація державних реєстраційних знаків, розпізнавання кольору й вимір швидкості руху автотранспортних коштів, автоматична перевірка лічених державних реєстраційних знаків по базах даних різного рівня й призначення, у тому числі по базах федерального, регіонального й оперативного розшуку, ведення бази даних автотранспорту, що проїхав через пост, обладнаний даним комплексом.

Даний відеокомплекс працює на підставі наступних принципів. Відеокамери розташовуються над дорогою на висоті 6 м на видаленні від 300 до 1000 м від стаціонарного поста. Довжина зони контролю становить 10 м. Для

забезпечення роботи АПК «Сова-2» у нічний час доби зони контролю висвітлюються. Кількість установлюваних відеокамер рівняється кількості контрольованих смуг руху автотранспорту. Сигнал від відеокамер передається на контролери розпізнавання. Передача відеосигналу може здійснюватися як по кабельній лінії, так і будь-яким бездротовим способом. Колір автотранспортного засобу визначається в області над місцем розташування державного реєстраційного знака. Швидкість руху автотранспортних коштів виміряється по швидкості зміни положення автотранспортного засобу в кадрі зображення. Результати обробки по локальній мережі передаються на консоль оператора.

На закінчення хочеться відзначити, що розглянутими прикладами не вичерпуються ні можливості технологій комп'ютерного зору, ні число компаній і груп, що займаються даною проблематикою. Зокрема, за рамки даної роботі вийшли групи, що займаються розробкою й побудовою так званих геоінформаційних систем (ГІС).

З 1995 року дослідницькі групи й фірми, що займаються цим розділом машинного зору, об'єднані в ГІС-асоціацію, що на сьогоднішній день нараховує більше 450 членів. У числі завдань, розв'язуваних геоінформаційними системами, – просторове відновлення рельєфу місцевості, розпізнавання об'єктового состава місцевості й побудова тривимірних цифрових карт за інформацією, отриманої за допомогою аерокосмічних засобів.

До найбільш відомих у світі фотограмметричних систем відносяться такі апаратно-програмні комплекси, як Leica і Intergraph, що поставляються разом з потужними робочими станціями. Це досить дорогі системи, і дозволити їх собі можуть деякі компанії. З розвитком обчислювальної техніки усе популярніше стають менш дорогі системи, що дозволяють проводити обробку зображень на персональних комп'ютерах.

Ще один напрямок в області машинного зору – побудова систем розпізнавання символів.

Підбиваючи підсумок, можна із упевненістю заявити, що українські технології комп'ютерного зору не уступають, а багато в чому й перевершують закордонні аналоги. Найчастіше компаніям, що розвивають ці технології, не вистачає всесвітньо відомого імені. Тому й інвестиції в них, як правило, роблять неохоче. Однак не викликає сумнівів, що високий рівень технологій і висока кваліфікація українських фахівців уже в недалекому майбутньому приведуть до домінування на світовому ринку саме українських технологій комп'ютерного зору.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програми мною була використана мова програмування Visual C++. У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка додатків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконний режим,

технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні додатки, а також додатки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої додатки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-додатки на базі об’єктно-орієнтованого підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення

тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

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

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

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи машинного зору для контролю якості на виробництві.

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

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

б) вибрати та обґрунтувати методику побудови системи 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

Машинний зір – це форма роботизованої технології, яка використовує цифрові зображення та камери, підключені до комп’ютера, щоб отримати інформацію. Він працює, знімаючи показання з цифрових камер або обладнання для отримання зображень і застосовуючи їх до алгоритмів. Це дозволяє машинам успішно ідентифікувати та каталогізувати різні об’єкти в навколишньому середовищі. Апаратний компонент машинного зору складається з цифрової камери, джерел світла, лінз і, можливо, інших компонентів, таких як відбивачі та фільтри. Усі вони працюють разом, щоб захопити зображення. Програмне забезпечення для машинного зору працює шляхом обробки та класифікації зображень, отриманих камерою, готуючи їх до аналізу.

Технологія машинного зору зараз стає все більш популярною в процесах контролю якості, наприклад для виявлення дефектів. Його також можна використовувати для вимірювання розмірів об'єктів або для порівняння об'єктів із зразком. Ця технологія не тільки підвищує ефективність і точність, але й зменшує потребу в участі людини, підвищуючи безпеку. Його можна використовувати в багатьох галузях промисловості, включаючи автомобільну, аерокосмічну та медичну промисловість.

Важливість контролю якості в промисловому виробництві

Контроль якості (QC) є життєво важливим аспектом виробничої промисловості, який охоплює всі види діяльності та процеси, що здійснюються для забезпечення відповідності продукції необхідним стандартам. Він передбачає систематичну та сувору перевірку сировини, компонентів і кінцевої продукції. Завдяки ретельному моніторингу кожного етапу виробничого процесу контроль якості гарантує, що товари відповідають їхнім характеристикам,

надійності та безпеці.

Неможливо підкреслити важливість контролю якості у виробничій промисловості, оскільки він безпосередньо впливає на продуктивність продукту, витрати на технічне обслуговування, задоволеність клієнтів і репутацію бренду, впливаючи на успіх бізнесу. Підприємства з високим контролем якості насолоджуються мінімізацією відходів, скороченням переробки, надійністю продукції та, таким чином, отримують конкурентну перевагу на своїх ринках, зрештою сприяючи довгостроковій лояльності та довірі клієнтів.

Основні етапи контролю якості в промисловому виробництві

Галузь або продукт значною мірою впливають на методи контролю якості, які використовуються в компанії. Щоб гарантувати, що належний продукт досягне свого споживача, тестування проводиться відповідно до стандартів, характерних для кожного бізнесу. Тестування на різних етапах виробництва допомагає виявити помилки виробничого процесу та вжити коригувальні дії, необхідні для запобігання їх повторенню.

Кожна стадія виробничого процесу включає різноманітні перевірки якості, і цей контроль якості може виконуватися повністю або частково залежно від країни, сектору, клієнта та продукту:

– **Зразок перед виробництвом**: перший етап тестування включає перевірку та оцінку вхідної сировини/компонентів. Це вкрай важливо, оскільки дефекти та невідповідності в сировині/компонентах можуть негативно вплинути на якість кінцевого продукту.

– **Перевірка сировини/Технічна перевірка**: другий етап випробування проводиться в лабораторіях. Сировина або продукт вибираються для проведення більш ретельного/глибшого аналізу якості. Результати цього аналізу також можуть бути надані зацікавленим сторонам.

– **In-line Inspection**: Під час виробничого процесу здійснюються контроль якості та перевірки, які часто називають поточною інспекцією. Це передбачає перевірку продукту на різних рівнях складальної лінії, щоб виявити та усунути

проблеми до того, як вони посиляться. Це економить час, гроші та ресурси компанії та допомагає компанії вжити необхідних заходів безпеки для майбутніх продуктів. Регулярна оцінка продукту під час виробничого процесу допомагає точно визначити несправність. Це зменшує відходи, зміни та повторне виготовлення всього продукту.

– **Позачергова перевірка**: перевірка відбувається, як тільки продукт залишає будь-яку виробничу лінію. Подібно до процесу внутрішньої інспекції, цей метод забезпечує більшу гнучкість у пошуку помилок і реалізації планів виправлення.

– **Остаточна перевірка**: це остання можливість для інспекторів/аудиторів якості запобігти відправці неякісного продукту, щоб вчасно вирішити проблеми з якістю, щоб запобігти виробництву такого продукту в майбутньому. Ця перевірка зазвичай проводиться, коли 100% замовлення виготовлено та приблизно 80% упаковано. Цей тест дозволяє аудитору вирішити, чи давати зелений сигнал для відвантаження цих продуктів.

– **Перевірка завантаження**: оскільки більшість промислових товарів маркуються та упаковуються перед відправкою клієнту. Останній етап перевірки також виконується під час маркування, пакування та завантаження. Цей останній крок гарантує відповідальне управління товарами та безпечне завантаження для безпечного транспортування.

Обмеження традиційних ручних методів перевірки

Традиційні процеси, які використовуються в якості, покладаються на перевірку операторами та схильні до помилок. Крім того, існують інші обмеження цих процесів, наприклад:

– Високі витрати: процеси ручного контролю якості вимагають багато трудових ресурсів, що може різко підвищити витрати на виробництво.

– Забирає багато часу: процеси ручного контролю якості займають багато часу та вимагають висококваліфікованого персоналу, щоб гарантувати, що продукти відповідають стандартам. Це може призвести до затримок у

виробництві та плануванні.

– Схильність до помилок: хоча процеси ручного контролю якості виконуються висококваліфікованим персоналом, людська помилка завжди є ризиком. Навіть найдосвідченіший персонал може робити помилки, що призводить до браку продукції.

– Обмежене охоплення: ручний процес контролю якості може охоплювати лише обмежену область, що ускладнює перевірку всього.

– Відсутність гнучкості: процеси ручного контролю якості не дуже гнучкі і не можуть легко адаптуватися до різних типів продукції.

Машинний зір для контролю якості: основні переваги

Традиційно контроль якості на виробництві покладався на ручні перевірки операторами, які можуть бути трудомісткими, суб'єктивними та схильними до помилок. Завдяки потребі у високоякісних продуктах зі зниженою нормою прибутку та суворим нормам безпеки людей процеси контролю якості пройшли автоматизацію, а сфера контролю якості переживає трансформаційну еволюцію. У той час як минулі системи контролю якості значною мірою покладалися на ручне втручання, нові та футуристичні системи використовують штучний інтелект, який підтримується машинним зором для виявлення навіть найменших недоліків. Ці нові розробки революціонізують виробничі лінії та підривають весь виробничий сектор.

Машинний зір пропонує кілька переваг перед традиційними методами контролю якості.

– По-перше, це підвищує точність перевірки за рахунок усунення людської помилки. Камери та алгоритми можуть виявляти дефекти та аномалії з більшою точністю, ніж людське око. Це гарантує, що схвалені лише продукти, які відповідають зазначеним стандартам якості.

– По-друге, машинний зір покращує ефективність перевірки шляхом автоматизації процесу. Перевірка вручну може зайняти багато часу, особливо для складних або повторюваних завдань. Системи машинного зору можуть

працювати безперервно та на високій швидкості, значно скорочуючи час перевірки та підвищуючи загальну продуктивність.

– По-третє, машинний зір дозволяє збирати та аналізувати дані в реальному часі. Збираючи та аналізуючи дані перевірок, виробники можуть визначати тенденції якості, контролювати виробничі процеси та приймати обґрунтовані рішення для оптимізації виробничих операцій.

– Крім того, машинний зір пропонує підвищену гнучкість і масштабованість. Системи можна легко запрограмувати для задоволення різних вимог перевірки та адаптації до мінливих потреб виробництва. Ця гнучкість дозволяє виробникам легко інтегрувати машинний зір в існуючі процеси контролю якості без значних збоїв.

Можливості зростання для процесів контролю якості на основі машинного зору

Ринок процесів контролю якості на основі машинного зору швидко зростає під впливом різних факторів. По-перше, зростаюча складність і різноманітність продукції, що виготовляється, вимагає більш досконалих методів контролю. Системи машинного зору можуть легко адаптуватися до різних типів продукції та виконувати інспекції з високою точністю, що робить їх ідеальними для таких галузей, як автомобільна, електронна, фармацевтична, харчова та напоїв.

По-друге, зростає попит на покращення якості та безпеки продукції. Дефектні продукти можуть призвести до відкликання, невдоволення клієнтів і навіть загрози безпеці. Машинний зір допомагає виробникам виявляти та усувати дефекти на ранніх етапах виробничого процесу, забезпечуючи надходження на ринок високоякісних і безпечних продуктів.

Деякі галузі, де спостерігають зростання контролю якості на основі машинного зору:

1. **Автомобільна промисловість**: автовиробники все частіше використовують системи машинного зору на виробництві для контролю якості,

від перевірки деталей автомобіля до забезпечення чистоти. Ця технологія використовується, щоб гарантувати, що деталі зібрані правильно та відповідають технічним характеристикам.

2. **Харчові продукти та напої**: виробники харчових продуктів і напоїв часто використовують системи машинного зору для контролю якості, наприклад, класифікації фруктів і овочів за солодкістю або перевірки упаковки на наявність помилок.

3. **Медицина**: компанії, що займаються медичним обладнанням, використовують системи машинного зору для перевірки та контролю якості. Ця технологія може виявляти дефекти в хірургічних інструментах та інших медичних пристроях.

4. **Фармацевтика**: фармацевтичні фірми використовують системи машинного зору для таблетування, наповнення капсул, сортування ліків і перевірки відповідності; ця технологія допоможе забезпечити безпеку та точність ліків.

5. **Виробництво електроенергії**: системи на основі машинного зору використовуються в галузі виробництва електроенергії для перевірки та обслуговування силових турбін і відповідних компонентів.

6. **Нафта та газ**: нафтові та газові компанії використовують машинний зір, щоб перевірити підводи та трубопроводи на наявність пошкоджень. Ці системи також використовуються для моніторингу процесів безпеки на нафтопереробних заводах.

Кілька компаній активно використовують машинний зір у своїх процесах контролю якості. В автомобільній промисловості машинний зір широко використовується для перевірки різних компонентів, таких як двигуни, панелі кузова та електронні системи. Такі компанії, як Cognex Corporation, Keyence Corporation і Omron Corporation, надають системи машинного зору та рішення, спеціально розроблені для контролю якості автомобілів.

Audi оптимізує перевірку якості в пресовому цеху за допомогою штучного

інтелекту

Ринок контролю якості на основі машинного зору є свідком сплеску активності стартапів і технологічного прогресу. Такі стартапи, як Inspekto, VITRONIC і Kneron, розробляють інноваційні рішення машинного бачення, які пропонують просту інтеграцію, доступність і підвищену точність.

Звіти про дослідження ринку також передбачають значне зростання ринку машинного зору для процесів контролю якості. Такі фактори, як дедалі більше впровадження концепцій Industry 4.0, прогрес ШІ та алгоритмів глибокого навчання, а також потреба в покращенні контролю якості, сприяють цьому зростанню.

Машинний зір революціонізує процеси контролю якості в промисловому виробництві. Його точність, ефективність, можливості аналізу даних у реальному часі та гнучкість роблять його ідеальним рішенням для галузей промисловості, які прагнуть покращити якість продукції та відповідати нормативним вимогам. У зв'язку з розширенням ринкових можливостей і розвитком технологій, процеси контролю якості на основі машинного зору будуть готові до значного зростання в найближчі роки. Крім того, нормативні вимоги та стандарти якості стають суворішими в різних галузях. Машинний зір пропонує надійний і послідовний метод виконання цих вимог і підтримки відповідності, зменшуючи ризик штрафних санкцій і шкоди репутації.

3.2 Розробка структурної схеми

Структурна схема системи зображена на рисунку 3.1. Системи машинного зору дозволяють автоматизувати контроль продукції, що випускається, і керування виробничими процесами шляхом аналізу візуальної інформації. Для формування зображень використовуються промислові відеокамери. Для визначення зображень використовують CBIR-системи.

Рисунок 3.1 – Структурна схема системи

Звичайний пошук на основі тексту не є достатнім для обробки цих мультимедійних даних у веб-сховищах, оскільки типовий розмір зображення надзвичайно великий порівняно з текстовим вмістом. Отже, пошук і отримання зображень із цих веб-сховищ є ще більш складним і складним процесом. Отже,

існує потреба в ефективній системі пошуку зображень, яка вимагає мінімальної участі людини. Традиційно для процесу пошуку зображень використовувалися ключові слова, ім’я файлу та розташування, пов’язані із зображеннями. Зокрема, цей метод відомий як метод пошуку зображень на основі тегів (TBIR) [1]. Система TBIR не працює з фактичною інформацією зображення, і це головний недолік цього підходу. Іншою проблемою, пов'язаною з системою TBIR, є процес анотації бази даних, який вимагає участі людини. Щоб вирішити всі вищезазначені проблеми, дослідники запровадили нову систему пошуку зображень, яка використовує фактичну помітну візуальну інформацію зображень. Цей новий підхід відомий як підхід до пошуку зображень на основі вмісту (CBIR) [2], [3], [4], [5]. Він використовує фактичну інформацію про зображення, тобто функції візуального зображення для процесу пошуку зображення. Функції примітивного візуального зображення – це інформація про колір, текстуру та форму зображення, відома як низькорівневі функції зображення, які застосовуються в комбінованому та ієрархічному порядку для отримання відповідних зображень. У процесі CBIR кожне зображення представлено вектором ознак, який містить усі ці ознаки візуального зображення низького рівня. Як наслідок, виділення низькорівневих візуальних ознак і побудова вектора ознак у CBIR є основними завданнями, оскільки продуктивність пошуку будь-якої системи CBIR залежить від якості вектора ознак. У даній роботі ми зосереджені на покращенні якості вектора ознак для розробки ефективної схеми CBIR.

Більшість зображень містять однорідні та неоднорідні області інтенсивності, де однорідні області інтенсивності містять групу пікселів, що мають однакові значення інтенсивності. Неоднорідні області інтенсивності – це ті області зображення, де група пікселів демонструє різні значення інтенсивності. Частина зображення, яка охоплює області максимальної неоднорідної інтенсивності, зазвичай привертає більшу частину візуальної уваги та вважається областю об’єкта (ObR). Ця об’єктна область зображення містить

суттєву життєво важливу інформацію зображення порівняно з рештою зображення, тобто необ’єктною областю (NObR). Усі ці розрізнення різних областей зображення можуть бути реалізовані за допомогою методів виявлення помітних областей. Цей дискримінаційний механізм візуальної концентрації можна ефективно та ефективно застосовувати в CBIR для досягнення кращої точності пошуку. Керуючись цими фактами, ми використали карту помітності на основі сигнатур зображення, щоб розрізнити зображення на зображення ObR і NObR. У зображенні ObR кожен піксель містить унікальні сигнатури зображення, тому, використовуючи техніку виділення ознак на рівні блоку, ми можемо ефективно витягувати кращі візуальні характеристики. Тому в цій роботі ми використали розкладання сингулярного значення (SVD) на рівні блоку, щоб зафіксувати максимальні сигнатури енергії кожного блоку зображення. Згодом у зображенні NObR більшість пікселів несуть аналогічні сигнатури зображення, тому для виділення важливих низькорівневих особливостей візуального зображення достатньо лише аналізу кольору чи текстури всього фонового зображення. Фільтр Габора є одним із поширених методів аналізу текстурних візерунків на зображенні. Однак аналізу текстури на основі однієї орієнтації недостатньо для представлення всіх текстурних візерунків NObR. Таким чином, ми застосували фільтр Габора до базових кольорових компонентів зображення NObR у комбінації різних масштабів і кількох орієнтацій. Далі ми об’єднали візуальні об’єкти ObR і NObR, щоб побудувати остаточний вектор об’єктів.

Отже, остаточний вектор ознак містить усі види локальних візуальних функцій, покращуючи якість вектора ознак для додатків CBIR. Однак важливість ознак ObR над NObR у кінцевому векторі ознак не є жорсткою, оскільки це характеристика, що залежить від зображення. Щоб вирішити цю проблему, ми також запропонували схему зіставлення подібності зображень на основі однорідності, яка окремо використовує функції ObR і NObR для обчислення значень подібності на основі запропонованої карти однорідності.

У літературі більшість систем CBIR використовують різні функції низького рівня незалежно або комбінацію різних функцій низького рівня в їх належній пропорції для процесу пошуку зображення. У низькорівневих характеристиках колірні характеристики вважаються одними з основних домінуючих характеристик зображення, оскільки людське око здатне розрізняти величезну кількість кольорів, присутніх на зображенні. Кольорова гістограма [6] є одним із найпоширеніших підходів до отримання статистичної інформації з колірних ознак. Гістограма просто дає інформацію про розподіл інтенсивності зображення. Однак неодноразово спостерігалося, що два різні зображення дають подібні гістограми, хоча їх кольорові композиції можуть здаватися абсолютно різними. Щоб подолати ці проблеми, багато дослідників використовували кольорові корелограми в запропонованій схемі CBIR [7]. Використання кольорових корелограм усуває неузгодженість, наявну в просторовій кольоровій інформації. Деякі дослідники також вивчали використання вектора когерентності кольорів, спільної гістограми та гістограми зображення з різною роздільною здатністю в процесі пошуку зображення [8]. Guang та ін. [9] запровадили систему пошуку зображень на основі гістограми кольорової різниці (CDH). CDH має ексклюзивну властивість підраховувати перцепційно узгоджену різницю кольорів між двома точками.

Текстура – це ще одна широко використовувана функція візуального зображення, яка надає інформацію щодо поверхневих візерунків зображення. Використання дискретних вейвлет-перетворень (DWT) [10] для аналізу текстур у трьох різних напрямках (тобто 0°, 45° та 90°) є одним із найпоширеніших методів виділення ознак текстури. Однак DWT не дуже ефективний через відсутність спрямованості. Щоб вирішити вищезазначену проблему, багато дослідників використовували складні вейвлет-перетворення (CWT) для вилучення особливостей текстури в шести різних напрямках (тобто ±15°,±45° і±75°) [11]. У 2013 році Rahul et al. [12] повернув комплексний вейвлет-фільтр на 45° і використав його з подвійним деревом CWT для виділення

особливостей текстури в 12 різних напрямках (тобто ±15°,±30°,±45°,±60°,±75° і±120°). У 2016 році Ghanshyam та ін. [13] використовували адаптивне тетролетне перетворення для отримання зображення текстури. Tetrolet transform досліджує всі можливі просторові структури, присутні на зображенні.

Форма різних об'єктів на зображенні надає інформацію про форму зображення. Інформація про краї різних форм є найпоширенішим способом представлення елементів форми. У літературі дослідники запропонували багато алгоритмів для визначення можливих країв зображення, як-от карта канні-країв, карта нечітких різниць-країв, карта країв Собела тощо. Усі ці алгоритми виділяють лише крайові пікселі зображення. Гістограма орієнтації країв (EOH) [14] є одним із найпоширеніших підходів для вилучення ознак форми з карт країв. EOH дає кількість пікселів у п'яти різних орієнтаціях (тобто 0°, 45°, 90°, 135° та піксель без країв). Тільки кількості пікселів недостатньо для представлення просторової геометрії зображення, і це головний недолік EOH. Щоб подолати цю проблему, дослідники використали детальну EOH [15]. Інші методи виділення ознак форми, представлені в літературі, це кривизна краю, максимальна довжина краю, підхід до заповнення водою тощо [16].

Лише однієї характеристики візуального зображення недостатньо, щоб описати всю можливу просторову інформацію, присутню в зображенні. У результаті дослідники поєднують різні функції візуального зображення для кращої продуктивності пошуку. Julesz та ін. [17] ввів поняття аксіом для теорії текстонів. Ця техніка просто поєднує функції кольору та текстури для процесу пошуку зображення. Пізніше було запропоновано багатотекстонну гістограму (MTH) [18] з використанням тієї ж концепції аксіоми. MTH поєднує ознаку текстури, отриману з матриці спільного розташування, і функцію кольору, витягнуту з колірних гістограм. Ця концепція аксіоми була використана Джханваром та ін. [19], щоб запропонувати матрицю спільної зустрічі мотивів (MCM) для процесу пошуку зображень. Деякі інші системи CBIR, засновані на комбінованих візуальних характеристиках, це модифікована матриця спільного

появи колірного мотиву (MCMCM), інтегрована матриця спільного виникнення кольору та інтенсивності (ICICM) тощо [20]. У 2016 році Varish et al. [6] використовували функцію кольору та текстури в ієрархічному порядку для процесу CBIR. У своїй схемі автори використовували нерівномірне бінове розкладання схеми колірної гістограми для виділення колірних ознак. Крім того, ці функції кольору використовувалися для зменшення простору пошуку зображень. На наступному ієрархічному рівні вони використовували функції текстури на основі комплексного вейвлет-перетворення подвійного дерева для виконання остаточного завдання пошуку. Пізніше Прадхан та ін. [21] запропонували трирівневу ієрархічну систему CBIR, яка використовувала особливості текстури, форми та кольору для кращої продуктивності пошуку. У своїй схемі автори запропонували гістограму кореляції колірного каналу та гістограму крайових з'єднань для виділення особливостей кольору та форми. Одночасно вони використовували схему, засновану на тетролетному перетворенні, для вилучення особливостей текстури об'єкта. У своїй роботі вони розглянули всі можливі комбінації ієрархічних функцій для зменшення простору пошуку зображень у різних програмах CBIR.

У більшості обговорюваних схем функції низького рівня були безпосередньо витягнуті з усієї частини зображення. Тоді як у багатьох випадках основна візуальна інформація міститься лише в обмеженій частині зображення. Як наслідок, виділений вектор ознак страждає від проблеми семантичного розриву. Щоб вирішити цю проблему, дослідники використали багато моделей помітності, щоб знайти помітну частину зображення для більш ефективного виділення ознак для CBIR. У 2015 році Zhang et al. [22] запропонував модель на основі сумки слів (RoI-BoW) для сегментації зображення та пошуку зображень. Спочатку Zhang et al. отримані ROI зображень за допомогою схеми різниці Гауса (DoG) на основі ключових точок. Пізніше вони використовували різні ROI, щоб побудувати Bag-of-visual-words (BoW). Пізніше Zhang et al. [23] запропонував систему пошуку зображень на основі розширеної

помітної області (ESR) для подальшого підвищення ефективності пошуку. У даній схемі автори поєднали помітну область з навколишньою територією, щоб створити ESR. Крім того, Zhang et al. використовував функції Gabor, SIFT і HSVH для створення візуального BoW для представлення та пошуку зображень. Ці дві схеми показали ефективні результати пошуку зображень Coral. Однак ці схеми покладаються на навчальні дані та можуть бути неефективно екстраполювати або передбачати несподівані сценарії та невідомі набори даних. У 2017 році Jian et al. [24] запропонував виявлення помітних патчів і метод направленого виділення патчів для виділення ознак. Пізніше Noh et al. [25] виконали великомасштабний пошук зображень, використовуючи уважний локальний дескриптор ознак, який використовує згорткову нейронну мережу (CNN) з увімкненим механізмом уваги. Пізніше в 2019 році Teichmann et al. [26] витягнув помітні області із зображень орієнтирів і об'єднав їх, щоб сформувати більш помітне представлення, використовуючи регіональне агреговане вибіркове збігове ядро (R-ASMK) для більш ефективного пошуку зображень. Нещодавно Wei et al. [27] представили систему CBIR, керовану помітністю, в якій вони використовували схему виявлення помітності разом із функціями CNN, щоб отримати більше семантичних характеристик зображення для CBIR.

Крім усіх локальних і помітних регіональних ознак зображення, багато дослідників використовували глобальні характеристики зображення для класифікації та пошуку зображень. Семантичний шаблон, пакет візуальних слів, виявлення об'єктів, онтологія об'єктів, модель на основі глибокої згорткової нейронної мережі (CNN) тощо [28], [29] є поширеними підходами до роботи з характеристиками глобального зображення. У 2017 році Караоглу та ін. [30] запропонували іншу систему CBIR на основі семантичних характеристик зображення високого порядку. У своїй схемі вони побудували текстові та візуальні підказки для класифікації та пошуку зображень. У 2018 році Мен та ін. [31] використовували схему регіонального відображення ознак згортки (RCMF) разом із підходом на основі згорткової нейронної мережі (CNN) для

виділення ознак і класифікації зображень у програмах CBIR. Мен та ін. [31] також запропонував нову схему об'єднання регіональних ознак для остаточного пошуку зображень. У 2019 році Лі та ін. [32] запропонував генеративну байєсівську модель для застосування CBIR, яка спільно використовувала багатопроекційну та багатофункційну схему навчання в процесі мережевого навчання для класифікації зображень. Таким чином воно вивчало різні представлення різних категорій зображень. Здебільшого всі ці обговорювані функції семантичного зображення високого порядку використовувалися в програмах виявлення об'єктів, розпізнавання об'єктів, класифікації зображень і додатків пошуку зображень на основі семантичних ознак. Системи CBIR, які використовують ці семантичні характеристики глобального або високого порядку, здебільшого виконують завдання класифікації зображень, за яким слідує остаточне завдання пошуку зображення. У цих схемах спочатку визначається клас зображення запиту за допомогою будь-якого відповідного навченого класифікатора зображень. Далі завдання пошуку зображення виконується лише для визначеного класу зображень. Ця схема суттєво скорочує кінцевий простір пошуку зображення для завдання пошуку зображення. Але ця схема CBIR на основі класифікації вимагає дуже великої кількості зображень для процесу навчання класифікатора. У той же час процес навчання мережі займає багато часу. Ці схеми також потребують нотації бази даних зображень, що є ще одним великим недоліком цих схем. Ці схеми або отримують весь набір відповідних зображень, або весь набір нерелевантних зображень у процесі пошуку. Для справжньої класифікації ці схеми виділяють увесь набір подібних зображень, а для помилкової класифікації ці схеми виділяють увесь набір несхожих зображень. Це ще один серйозний недолік цієї схеми.

Щоб вирішити ці проблеми, у даній роботі ми використали звичайний підхід CBIR, згідно з яким результати пошуку містять значну кількість подібних зображень щодо зображення запиту. У даній роботі ми використали схему виявлення об'єкта на основі характерної помітності, щоб отримати більше

помітних ознак об’єкта, що покращує якість вектора ознак. Ми також провели детальний порівняльний аналіз між запропонованою системою та іншими суміжними системами. Ми також порівняли ефективність пошуку запропонованої схеми з добре відомою схемою CBIR на основі VGG-16 CNN [33], щоб продемонструвати прийнятність і надійність запропонованої системи.

Як обговорювалося в підрозділі огляду літератури, багато систем CBIR виділяють низькорівневі характеристики зображення безпосередньо із зображень. Як наслідок, витягнутий вектор ознак страждає від проблем із семантичними розривами. Дослідники також використовували машинне навчання та моделі CNN для виконання CBIR на основі класифікації. У багатьох випадках ці підходи не дозволяють отримати потрібні зображення через проблему неправильної класифікації. Окрім цих методів, у літературі також існує багато CBIR на основі виявлення помітних областей, таких як [22], [23], [24], але ці системи витягують характеристики лише з виявлених областей. У той же час ці системи в основному покладаються на навчальні дані, щоб знайти помітну область зображення. Як наслідок, ці системи CBIR можуть не ефективно екстраполювати або прогнозувати несподівані сценарії та невідомі набори даних. Сьогодні дослідники вбудовують виявлення помітних областей в архітектуру CNN [25], [26], щоб підвищити продуктивність пошуку. Але ці схеми виділяють характерні ознаки лише з виявлених областей і відкидають фонові особливості. Однак у багатьох випадках довідкова інформація також відіграє важливу роль у розумінні семантичного значення зображення.

У даній роботі, щоб вирішити ці проблеми, ми використали стандартний підхід CBIR на основі декомпозиції помітних областей, який обіцяє, що результати пошуку містять значну кількість подібних зображень щодо зображення запиту. Запропонована схема виділяє різні характеристики з об’єктних (ObR) і необ’єктних (NObR) областей зображення. Оскільки ці функції не вносять однаковий внесок у семантичну інформацію зображення, у даній роботі було введено зіставлення зображень на основі карти однорідності. Ця

схема генерує оцінку подібності, враховуючи важливість функцій ObR і NObR, що в кінцевому підсумку покращує точність пошуку. Основні внески цієї роботи такі:

– Для зменшення проблеми семантичного розриву була запропонована схема декомпозиції ObR і NObR на основі ознак помітності. Ця схема декомпозиції області не потребує навчання та добре працює в неочікуваних сценаріях і невідомих наборах даних.

– Зображення ObR містить пікселі з унікальними підписами зображення, тому техніка виділення ознак на рівні блоку може ефективно витягувати кращі візуальні характеристики. Отже, ми використали розкладання сингулярного значення (SVD) на рівні блоку, щоб зафіксувати сигнатури максимальної енергії кожного блоку зображення.

– Зображення NObR містить пікселі з аналогічними підписами зображення. Таким чином, аналіз кольору або текстури NObR є достатнім для захоплення важливих характеристик фонового зображення. Тому ми застосували фільтр Габора над основними кольоровими компонентами зображення NObR у комбінації різних масштабів і кількох орієнтацій.

– Значимість ознак ObR порівняно з ознаками NObR не є жорсткою, оскільки це характеристика, яка залежить від зображення. Щоб вирішити цю проблему, була запропонована схема зіставлення подібності зображень на основі однорідності, яка окремо використовує функції ObR і NObR для обчислення значень подібності на основі запропонованої карти однорідності.

– Нарешті, ми використали комбіновані та окремі функції ObR та NObR на семи наборах даних, щоб продемонструвати ефективність запропонованої системи CBIR.

3.3 Розробка функціональної схеми

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

Система реалізує наступні функції:

1. Інтерфейс користувача сервісу машинного зору для контролю якості на виробництві.

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

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

4. Введення ключового слова-тега для пошуку зображення.

5. Блок уточнення запиту:

6. Блок введення тегів, для опису конкретного зображення.

7. Вибір баз зовнішніх серверів:

8. Вибір фільтрів:

– 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.

13. Блок виробничих функцій:

– Зчитування текстового маркування.

– Зчитування 1D і 2D кодів.

– Простежування продукції.

– Контроль наявності об'єктів.

– Вимір геометричних розмірів.

– Порівняння зі зразком.

– Підрахунок об'єктів.

– Ідентифікація об'єктів.

– Контроль кольору.

– Зір промислових роботів.

– Контроль якості – «комплексна інспекція».

– 3D інспекція.

Зчитування текстового маркування

Системи машинного зору дозволяють проводити зчитування текстового маркування з поверхні заготівель і з готової продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Оптичне розпізнавання символів являє собою важливий для забезпечення відслідковування й контролю якості продукції елемент багатьох завдань в автомобільній і харчовій промисловості, виробництві споживчих товарів, фармацевтичної й електронної промисловості, а також на ринку поштових послуг.

Особливості:

– розпізнавання символів різної ширини, символів з нахилом;

– розпізнавання дотичних символів фіксованих шрифтів;

– розпізнавання символів на зашумленному тлі;

– розпізнавання рядків зі змінною довжиною;

– розпізнавання до 600 рядків в хвилину.

Зчитування 1D і 2D кодів

Системи машинного зору дозволяють проводити зчитування будь-яких 1D і 2D кодів з поверхні заготівель і готової продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Зчитування 1D і 2D кодів являє собою важливий для забезпечення відслідковування й контролю якості продукції елемент багатьох завдань в автомобільній і харчовій промисловості, виробництві споживчих товарів, фармацевтичної й електронної промисловості, а також на ринку поштових послуг.

Особливості:

– всеспрямоване зчитування кодів;

– одночасне зчитування декількох кодів;

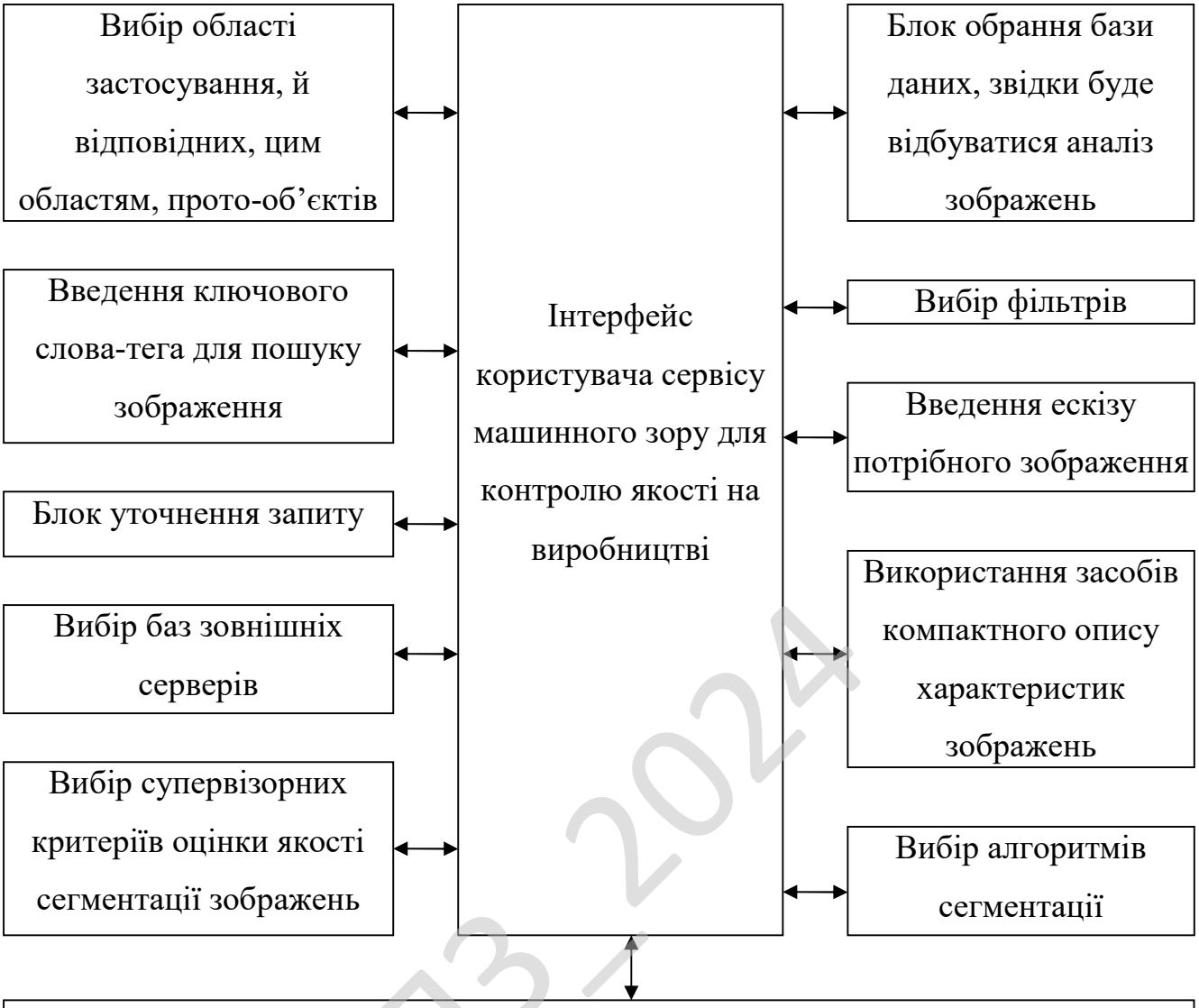
– зчитування кодів при значних перспективних перекручуваннях;

– зчитування кодів, нанесених будь-яким способом (струминний, ударно-крапковій, лазерний, прямій маркування);

– зчитування з будь-якого типу поверхонь;

– зчитування ушкоджених, розмитих, погано надрукованих кодів;

– зчитування до 7 200 кодів в хвилину.



Блок виробничих функцій:

- Зчитування текстового маркування.
- Зчитування 1D і 2D кодів.
- Простежування продукції.
- Контроль наявності об'єктів.
- Вимір геометричних розмірів.
- Порівняння зі зразком.
- Підрахунок об'єктів.
- Ідентифікація об'єктів.
- Контроль кольору.
- Зір промислових роботів.
- Контроль якості – «комплексна інспекція».
- 3D інспекція.

Рисунок 3.2 – Функціональна схема системи.

Простежування продукції

Виробники деяких споживчих товарів повинні мати системи, що створюють інформаційний слід, що тягнеться за кожним виробом по ланцюжку поставок. Для забезпечення безпеки продукту й ефективності заходів щодо відкликання товару, виробник повинен мати можливість швидко ідентифікувати потенційно небезпечний виріб, що не відповідає заявленим критеріям, і виявити його місце розташування в ланцюжку виробництва.

Особливості:

– ідентифікація виробу за допомогою зчитування кодів або текстового маркування;

– зчитування будь-яких 1D, 2D кодів з поверхонь різного типу;

– зчитування будь-якого текстового маркування;

– передача даних на наступні крапки контролю;

– відстеження продукції за допомогою спеціального програмного забезпечення Track&Trace;

– аналіз якості кодів і маркування на відповідність міжнародним стандартам.

Контроль наявності об'єктів

Системи машинного зору дозволяють перевіряти комплектність, виконувати контроль наявності й положення елементів заготівель і готової продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Особливості:

– контроль комплектності й розміщення продукції;

– контроль наявності й положення елементів продукції;

– необмежена кількість елементів для контролю;

– комплексна інспекція продукції;

– всебічний контроль продукції.

Вимір геометричних розмірів

Системи машинного зору дозволяють проводити безконтактні виміри геометричних розмірів і форми заготівель, готової продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Особливості:

– Високоточний вимір розмірів критично важливих деталей незалежно від зміни орієнтації деталі й умов навколишнього висвітлення.

– Субпіксельна точність вимірів.

Порівняння зі зразком

Системи машинного зору дозволяють проводити безконтактне зіставлення форми й зовнішнього вигляду виробів з еталонним зразком для виявлення відхилень від заданих характеристик продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Порівняння зі зразком також дозволяє робити класифікацію продукції безконтактним методом для подальшого їхнього розподілу або обліку.

Особливості:

– одночасна обробка декількох об'єктів;

– надійне й точне виявлення деталей у різних місцях розташування;

– обробка значних відмінностей в орієнтації, розмірі й зовнішньому вигляді деталей;

– максимальна продуктивність і надійність;

– порівняння по «золотому шаблоні» попіксельно.

– Надійність алгоритмів забезпечує якість аналізу поза залежністю від наступних факторів:

– відмінностей у контрастності об'єктів;

– зміни умов висвітлення;

– різниці у фокусі зображень;

– погіршення зовнішнього вигляду об'єктів;

– часткового перекриття об'єктів.

Підрахунок об'єктів

Системи машинного зору дозволяють виконувати підрахунок, контроль наявності/відсутності заготівель і готової продукції в процесі виробництва безпосередньо на конвеєрі або виробничій лінії, незалежно від способу їхнього розміщення й кількості.

Особливості:

– автоматизований підрахунок готових виробів;

– автоматизований підрахунок виробів в упакуванні;

– контроль наявності/відсутності виробів;

– гнучке настроювання під різні типи продукції.

Ідентифікація об'єктів

Системи машинного зору дозволяють проводити ідентифікацію будь-яких об'єктів по різних відмітних ознаках у процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

По відмітних ознаках можна унікально ідентифікувати об'єкт контролю, перевірити його відповідність заданим параметрам.

Особливості:

– Ідентифікація по наступних ознаках:

– Геометрична форма.

– Геометричні розміри.

– Колір.

– Наявність отворів і інших геометричних ознак.

– Зміст текстового маркування або штрих-коду.

– Етикетка.

– Комплектність складених виробів і багато чого іншого.

– Перевірка результатів складання по декількох ознаках у процесі виробництва.

Контроль кольору

Системи машинного зору дозволяють на основі кольору й колірної моделі об'єктів визначати їхнє місце розташування й наявність, проводити вимір і підрахунок у процесі виробництва безпосередньо на конвеєрі або виробничій лінії.

Особливості:

– надійне й точне визначення кольору різних об'єктів;

– визначення якості об'єкта на основі сполучення декількох кольорів;

– можливість вивчення кольорів одним клацанням миші усуває необхідність у розумінні різноманіття кольорів;

– доступні різні інструменти роботи з кольором: лінійка, підрахунок об'єктів, виділення плям, визначення площі, зіставлення кольорів і т.д.;

– перетворення кольорових зображень у чорно-білі для виконання додаткових видів перевірки з використанням чорно-білих фільтрів.

Зір промислових роботів

Системи машинного зору, використовувані разом з роботами, дозволяють істотно розширити спектр розв'язуваних завдань на виробництві.

Розв'язувані завдання системи «зір+робот»:

– Переміщення продукції.

– Завантаження/розвантаження паллет або піддонів.

– Класифікація об'єктів.

– Відбраковування виробів.

– Інспекція об'єктів із всіх ракурсів.

Переваги використання машинного зору для роботів:

– Керування роботом у процесі виробництва.

– Всебічний контроль продукції.

– Точність роботи.

– Запобігання випадкових зіткнень роботів.

– Усунення необхідності купувати високоточне встаткування.

– Можливість обробляти різні об'єкти без складного перенастроювання.

Особливості рішень:

– Нелінійне калібрування – поліпшує точність і повторюваність, дозволяє боротися з дисторсією (перекручуваннями зображення).

– Наявність готових драйверів для всіх роботів і приклади коду для роботів.

– Наявність інструментів світового класу для виявлення, інспекції й виміри об'єктів.

– Використання алгоритму PatMax® – кращого галузевого високоточного, стабільного алгоритму для виявлення нефіксованих об'єктів.

Контроль якості – «комплексна інспекція»

Системи машинного зору дозволяють виконувати комплексну інспекцію об'єктів будь-якої складності, незалежно від їхнього положення, безпосередньо на конвеєрі або виробничій лінії.

Комплексна інспекція дозволяє одночасно відслідковувати ряд характеристик продукції, що приводить до значного зниження вихідного браку.

Особливості:

– Комплексна інспекція об'єктів на основі використання декількох інструментів контролю.

– Зчитування тексту й штрихкоду.

– Виявлення дефектів продукції.

– Зіставлення з еталоном.

– Вимір розмірів.

– Перевірка кольору.

– Підрахунок об'єктів і багато чого іншого.

– Виконання аналізу можливо на одній камері.

3D інспекція

Відповідність стандартам і відслідковуваність підвищують безпека пацієнта й знижують потенційну відповідальність.

Незалежно від продукту, що випускається, упакування впливає на сприйняття споживачем якості й вартості. Якщо в упакуванні перебуває не то виріб, або має місце неправильна доставка, страждає рівень задоволеності покупця й образ торговельної марки. Виробники, які надають пріоритет відповідності впакування й відслідковуваності продукту, домагаються значної конкурентної переваги.

Відповідність, відслідковуваність, підтвердження дійсності:

– Унікальність кодування кожного лота, партії або елемента для визначення часу й місця виробництва може допомогти вам досягти більше ефективного й менш дорогого відкликання товару.

– Збільшення споживчої безпеки.

– Запобігання контрафакції.

– Зупинка паралельних угод і продажів по недозволених каналах.

– Мінімізація втрат.

– Ефективність заходів щодо відкликання товарів.

– Захист торговельної марки.

– Контроль виробничого процесу й керування поставками Чим краще й точніше системи відстеження й контролю походження, тим швидше ви можете ідентифікувати й вирішити проблеми. Якщо чистий прибуток зменшується, зниження вартості виробництва й поставок, найчастіше може збільшити різницю між прибутком і збитками.

– Поліпшення процесу керівництва роботами без зупинки виробництва.

– Зменшення числа найменувань.

– Оптимізація доступності й використання виробничого встаткування.

– Мінімізація продажів некондиційної продукції.

– Повний спектр рішень в області перевірки й розпізнавання.

Приклади використання:

– Перевірка правильності етикетки й місця її розташування.

– Виявлення помилок у рулоні етикеток.

– Перевірка наявності кришок і схоронності пломб на пляшках.

– Узгодження впакування з видом продукції.

– Читання кодів і перевірка тексту на еластичній пакувальній плівці й пакетах.

– Читання одномірних і двомірних кодів і тексту на етикетках.

– Читання штрихкодів і тексту на плоских, вигнутих і блискучих поверхнях.

– Читання й перевірка двомірних кодів, надрукованих безпосередньо на поверхні продукту або контейнера.

– Колірне сортування й перевірка.

– Автоматичне виробництво з можливістю сортування продукції або орієнтування її на розлив або впакування.

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврської дипломної роботи, наведена на рисунку 3.3.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування).

Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

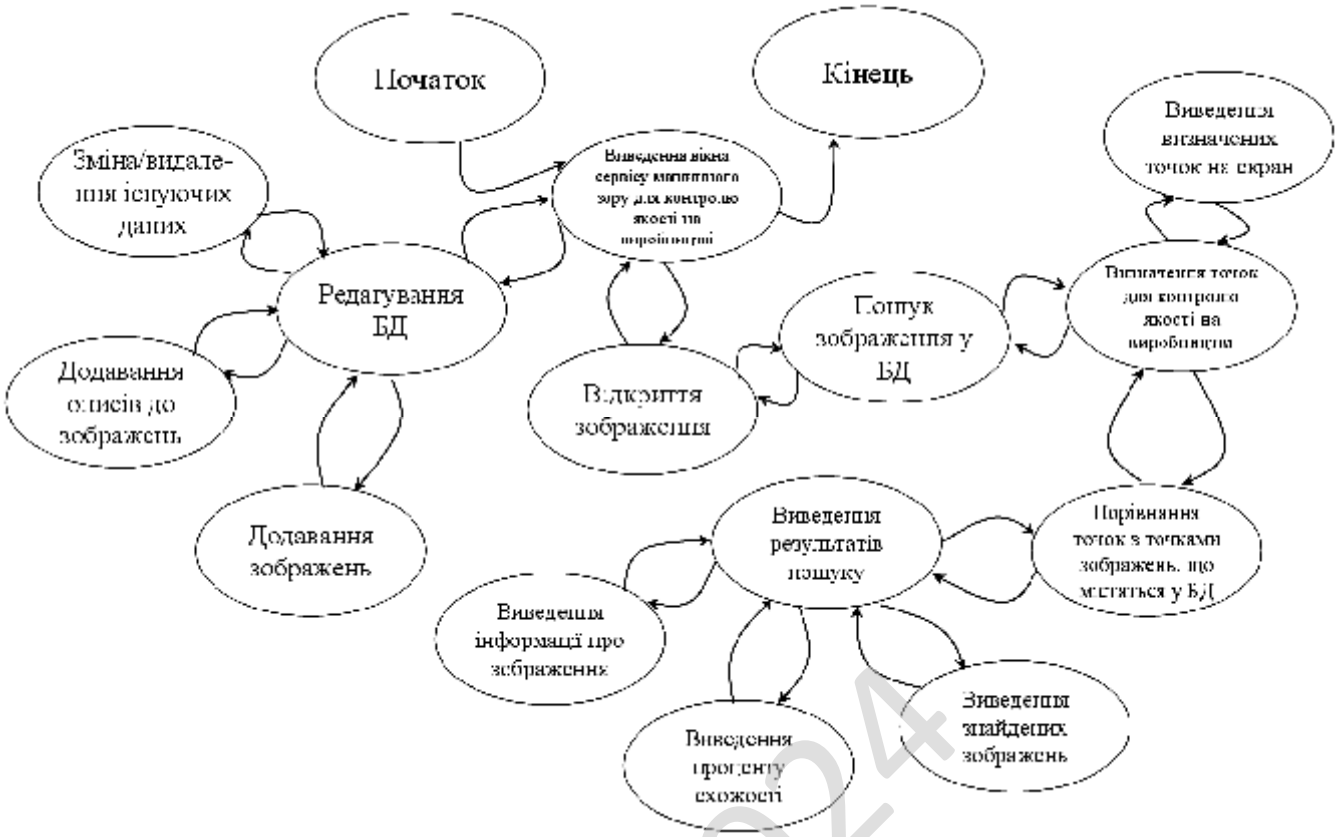


Рисунок 3.3 – Діаграма взаємодії процесів

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

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

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю сервісу машинного зору для контролю якості на виробництві.

На рисунку 4.1 наведено блок-схему основної програми, на рисунку 4.2 зображено роботу підпрограми.

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються.

Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

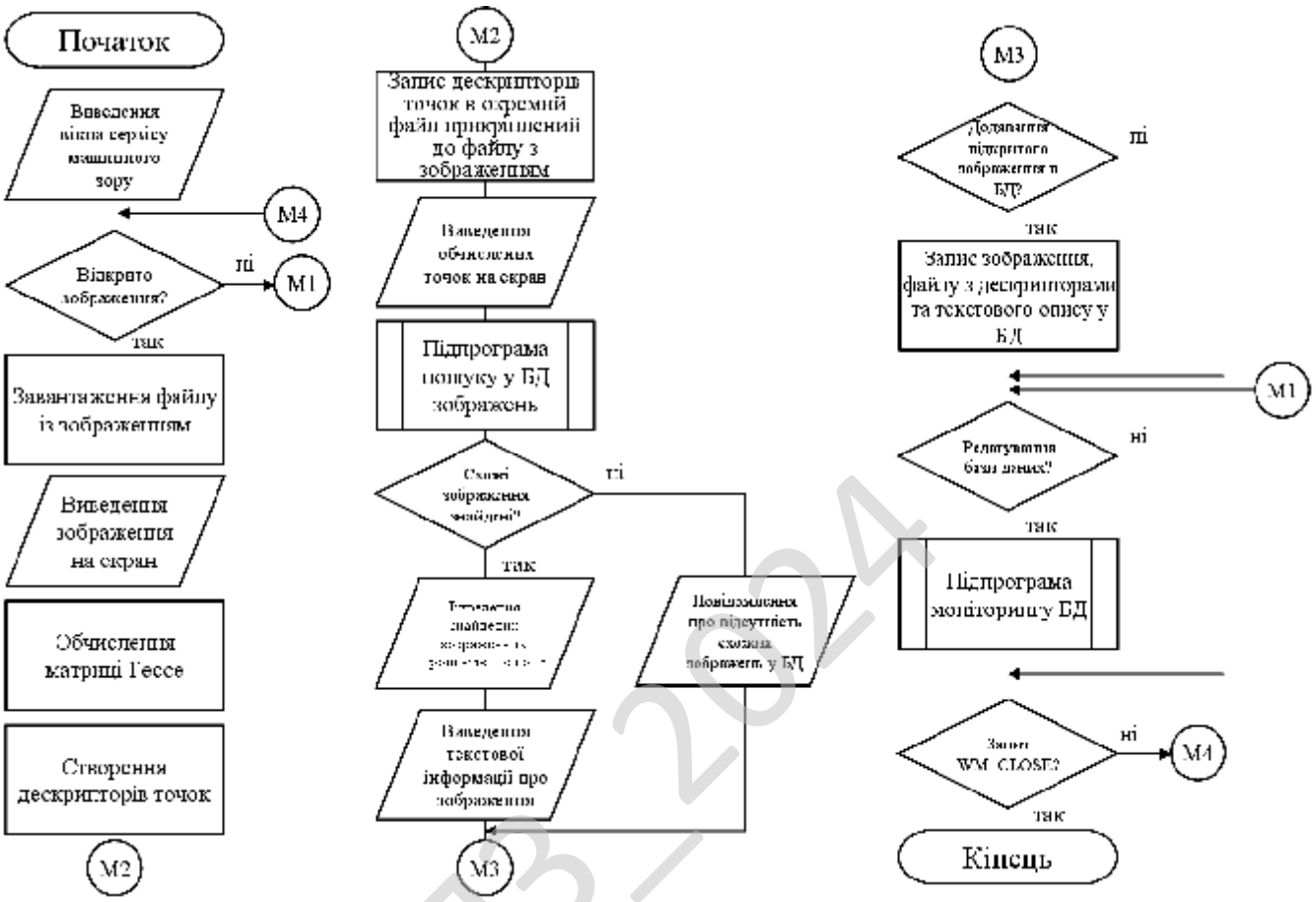


Рисунок 4.1 – Блок-схема основної програми

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

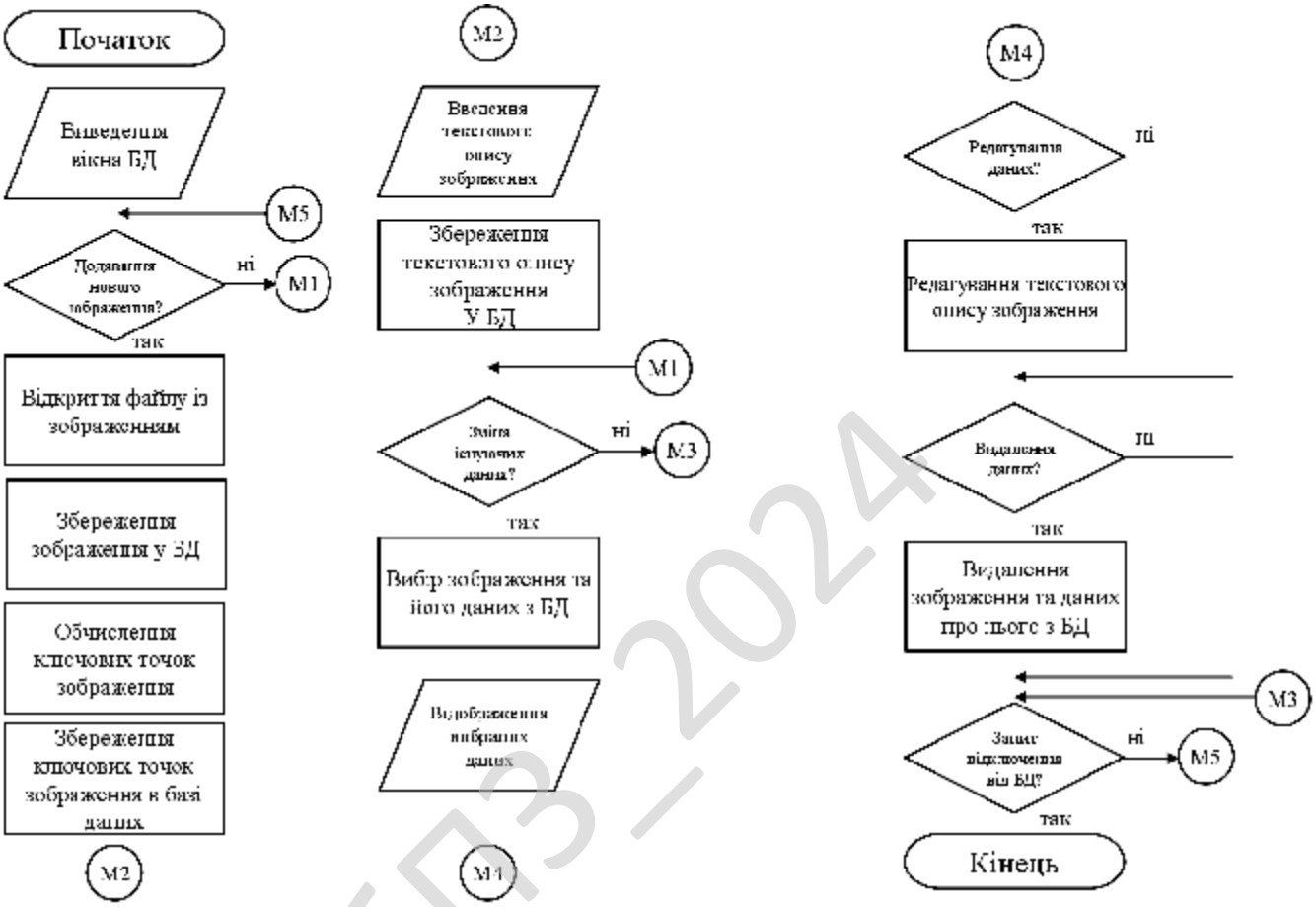


Рисунок 4.2 – Блок-схема роботи підпрограми

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу.

Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); діаграма класів.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії

мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі

випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з

класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

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.

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

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи.

Розроблене програмне забезпечення сервісу машинного зору для контролю якості на виробництві складається з наступних функціональних блоків:

– Навігаційне меню: Файл; БД; Ідентифікація; Довідка.

– Вікна відображення завантаженого зображення чи зображення отриманого з Веб-камери.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

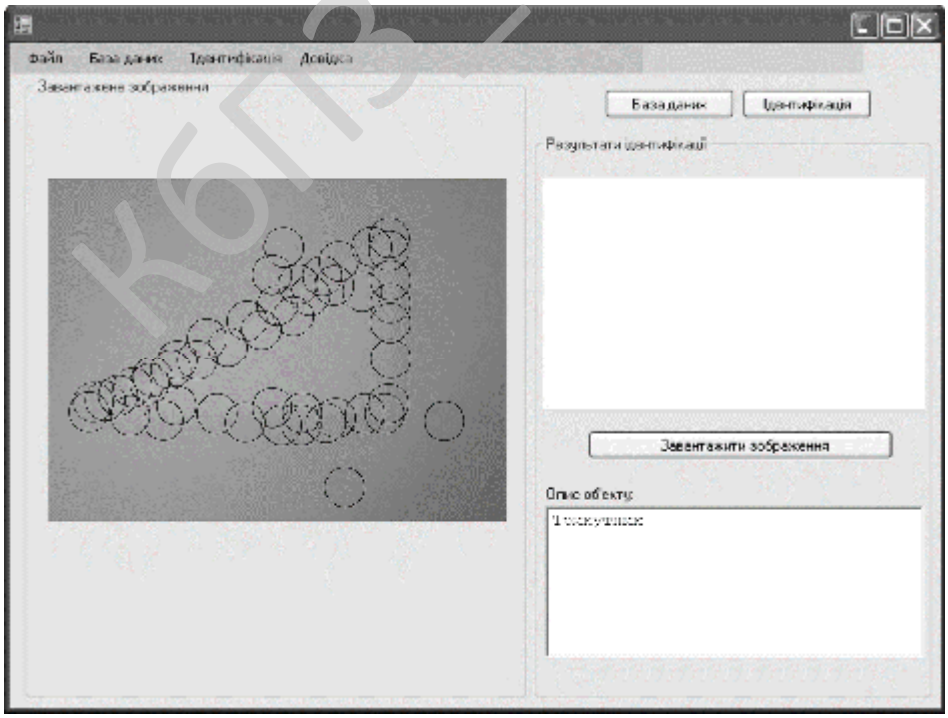


Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

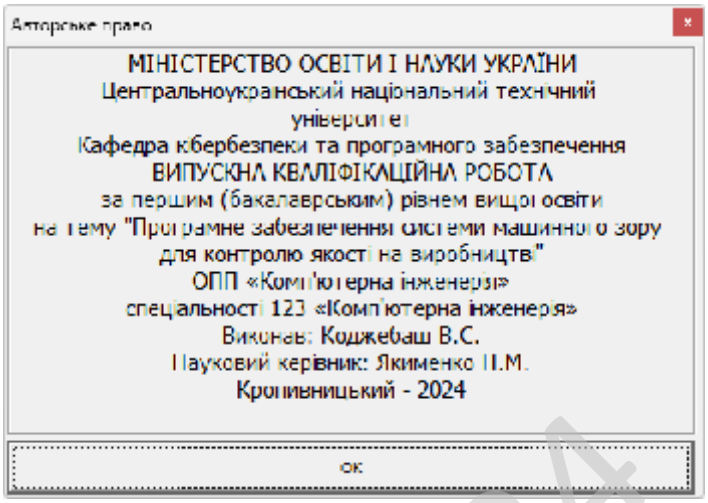


Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом чорної скриньки.

Основне місце програми тестів «чорної скриньки» – інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору ОТ.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функцій;

– Помилок інтерфейсу;

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних;

– Помилок характеристик (необхідна ємність пам'яті і т.д.);

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – proprietary software.

Програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримує обмежені права користування ним: може бути заборонено або закрито доступ до коду (вивчення), внесення змін, тиражування, розповсюдження та перепродаж. Програмне забезпечення вважається власницьким, якщо наявне хоча б одне з перелічених обмежень.

Найчастіше основним методом захисту майнових прав на власницьке ПЗ, поза ліцензійною угодою, власник обирає закриття сирцевого коду, захищаючи свій продукт від модифікації і вбудовуючи системи обмеження користування через авторизацію. Таке програмне забезпечення називається закритим. Проте, код власницького продукту може бути і відкритим, але власник може обмежити права користувача умовами користувацької ліцензії.

Власницьке програмне забезпечення та комерційне програмне забезпечення не є синонімами – власницьким може бути і безплатне (тобто, некомерційне) програмне забезпечення.

На противагу власницькому ПЗ існує вільне програмне забезпечення, автори і власники якого дозволяють вивчати, модифікувати і поширювати свій продукт. Саме визначення власницького програмного забезпечення виникло в результаті діяльності громадського руху вільного програмного забезпечення (представленого Фондом вільного програмного забезпечення та іншими організаціями) і осмислення умов свободи користування програмами. Визначенням власницького програмного забезпечення є не невідповідність хоча б одній з базових умов вільного програмного забезпечення. Сама назва власницьке ПЗ підкреслює визначальне значення власника у способі використання і можливостях розвитку цього програмного забезпечення.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи машинного зору для контролю якості на виробництві.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем машинного зору для контролю якості на виробництві.

– Досліджена система машинного зору для контролю якості на виробництві.

– На основі отриманих результатів досліджень створена програмна реалізація системи машинного зору для контролю якості на виробництві.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання машинного зору для контролю якості на виробництві.

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

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

Програма реалізована на мові високого рівня Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи машинного зору для контролю якості на виробництві. 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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Foreman J.W. Data Smart: Using Data Science to Transform Information into Insight 1st Edition. – Wiley, 2013. – 432 p.

2. Hurbans R. Grokking Artificial Intelligence Algorithms. – Manning, 2020. – 631 p.

3. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.

4. Kotu V., Deshpande B. Data Science: Concepts and Practice. – Elsevier Science, 2018. – 953 p.

5. Knowledge Base A Complete Guide - 2021 Edition // The Art of Service - Knowledge Base Publishing, 2020. – 306 p.

6. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

7. Mattmann C. Machine Learning with TensorFlow, Second Edition. – Manning, 2020. – 1124 p.

8. Mueller J.P., Massaron L. Machine Learning For Dummies. – Wiley, 2016. – 714 p.

9. Teofili T. Deep Learning for Search. – Manning, 2019. – 695 p.

10. Rungta K. TensorFlow in 1 Day: Make your own Neural Network. – Publishdrive, 2019. – 587 p.

11. Weidman S. Deep Learning from Scratch: Building with Python from First Principles. – O'Reilly. 2019. – 252 p.

12. Rajasekaran S., Vijayalakshmi Pai G.A. Neural networks, fuzzy logic, and genetic algorithms: synthesis and applications (with cd-rom) Kindle Edition. – PHI, 2013. – 628 p.

13. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems,* 2023, 729 LNNS, pp. 104–112.

14. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

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

16. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

17. 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, pp. 1-12.

18. 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.

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

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

21. 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.

22. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». CEUR Workshop Proceedings Volume 3101, 2021, Pages 192-207.

23. 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.

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

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

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

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., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in

Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

36. 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.

37. 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.

38. Smirnov, S., Bulekbaeva, G., Kikvidze, O.G., Lakhno, V., Brzhanov, R., Tabylov, A. «Computer simulation in the MathCAD package of plastic deformation of the deposited layer on the flat surface of the part». Journal of Theoretical and Applied Information Technology Volume 97, Issue 20, 2019, Pages 2467-2484. (Scopus).

39. 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.

40. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

41. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». Сучасні інформаційні системи. 2021. Т. 5, № 4. С. 79-95

42. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для

підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

43. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

44. 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.

45. Смірнов, С.А., Смірнова, Т.В., Минайленко, Р.М., Доренський, О.П., Сисоєнко С.В. «Хмарна автоматизована система інтелектуальної підтримки прийняття рішень для технологічних процесів». Вісник Черкаського державного технологічного університету. Технічні науки. №4, 2020, С. 84-92.

46. 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.

47. 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).

48. О.А. Смірнов, Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура № 1 (11). с. 48-57, 2019.

49. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу

відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

50. Смірнова Т.В., Дрєєв О.М., Смірнов О.А. Експертна система оптимізації процесу відновлення та зміцнення поверхонь деталей типу «вал» електродуговим напиленням. Системи управління, навігації та зв'язку, № 2 (54). С. 149-154, 2019.

51. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

52. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). - Полтава: ПолтНТУ. - 2017. - С. 112-115.

53. Смірнов О.А., Коваленко О.В. Використання псевдобулевих методів бівалентного програмування для управління ризиками розробки програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 1 (37). - Полтава: ПолтНТУ. - 2016. - С. 98-103

54. Смірнов О.А., Лисенко І.А. Формалізація процесу проектування тестових наборів. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 3 (48). - Харків: ХУПС. - 2016. - С.96-100.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи машинного зору для контролю якості на виробництві.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 132-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи машинного зору для контролю якості на виробництві.

4 Джерела розробки

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

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи машинного зору для контролю якості на виробництві;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

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

5.4 Вимоги до архітектури

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

5.5 Вимоги до надійності

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

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

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

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

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

5.8 Вимоги до інформаційної і програмної сумісності

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

5.8.1 Обладнання

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

5.8.2 Мова програмування

Середовище Visual C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 85 аркушів.

8 Етапи розробки

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

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

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 13.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Якименко Н.М.

Програмне забезпечення системи машинного зору для контролю якості на виробництві

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 39

Літера: РП

Кропивницький – 2024 року

main.cpp – головний файл програми

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

//-------------------------------------------------------
/* Програма для розпізнання графічних об'єктів методом SURF для сервісу машинного
зору для контролю якості на виробництві Visual C++.  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 для сервісу
машинного зору для контролю якості на виробництві Visual C++ знайшла: " <<
ipts.size() << " особливі точки" << std::endl;
  std::cout<< "Програма розпізнання графічних об’єктів методом SURF для сервісу
машинного зору для контролю якості на виробництві Visual C++ 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 для
сервісу машинного зору для контролю якості на виробництві Visual C++",
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 для
сервісу машинного зору для контролю якості на виробництві Visual C++", img);

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

  cvReleaseCapture( &capture );
  cvDestroyWindow( "Програма розпізнання графічних об’єктів методом SURF для
сервісу машинного зору для контролю якості на виробництві Visual C++" );
  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 для
сервісу машинного зору для контролю якості на виробництві Visual C++",
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 для
сервісу машинного зору для контролю якості на виробництві Visual C++", 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 для
сервісу машинного зору для контролю якості на виробництві Visual C++" );
  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 для
сервісу машинного зору для контролю якості на виробництві Visual C++",
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 для
сервісу машинного зору для контролю якості на виробництві Visual C++", 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 для
сервісу машинного зору для контролю якості на виробництві Visual C++" );
  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 для сервісу машинного зору для контролю якості на виробництві Visual C++ ---
*
*********************************************************/

#include "utils.h"

#include "surf.h"

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

const double gauss25 [7][7] = {

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

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

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

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

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

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

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

const double gauss33 [11][11] = {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

//! Беремо модифікований дескріптор.

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

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

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

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

i = -8;

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

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

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

    j = j - 4;

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

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

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

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

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

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

      }
    }
```

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

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

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

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

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

}


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

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

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

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

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

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

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

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

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

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

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

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

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

  return 0;
}
```

surf.h – файл заголовків

```
/*************************************************************
* --- Програма розпізнання графічних об'єктів методом SURF для сервісу машинного зору для контролю якості на виробництві Visual C++ --- *
*************************************************************/

#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 для сервісу машинного
зору для контролю якості на виробництві Visual C++ --- *
*
* *
*********************************************************/

#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ --- *
*********************************************************/

#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 для сервісу машинного
зору для контролю якості на виробництві Visual C++ ---
*
*
*
*********************************************************/

#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 для сервісу машинного
зору для контролю якості на виробництві Visual C++ ---
*
*
*                                                        *
*********************************************************/

#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ --- *
* *
*************************************************************/

#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ ---
*
*
*                                                            *
*************************************************************/

#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 для сервісу машинного
зору для контролю якості на виробництві Visual C++ ---
*                                                            *
*************************************************************/

#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ --- *
* *
*************************************************************/

#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ ---
*
*************************************************************/


//#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 для сервісу машинного зору для контролю якості на виробництві Visual C++ ---
*
*************************************************************/

#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)*/);
}

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