Кіровоградський державний педагогічний університет
імені Володимира Винниченка

Кафедра інформатики

КУРСОВА РОБОТА

з прикладної математики

на тему: **Розробка та реалізація алгоритму розпізнавання силуетів на статичному зображенні**

Студента III курсу 36 групи
спеціальності 6.040203 Інформатика
Олійника Владислава Михайловича

Керівник: Болілий В.О.,
кандидат фізико-математичних наук,
доцент

Національна шкала _____
Кількість балів: _____Оцінка: ECTS _____

Члени комісії

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали

м. Кіровоград – 2016 рік

ЗМІСТ

ВСТУП

Актуальність роботи. Комп’ютерний зір – спосіб з’ясувати, що саме знаходиться на зображенні. Така проста задача, яку людина кожного дня успішно розв’язує незліченну кількість разів, для комп’ютера представляється досить складною, незрозумілою та малодослідженою. На сьогодні людство лише починає робити несміливі маленькі кроки до того, щоб навчити комп’ютер робити хоча б частку того, що вміє робити людина, а зокрема людське око та людський зір. [18]

Як наукова дисципліна, комп’ютерний зір відноситься до теорії та технології створення штучних систем, що отримують інформацію із зображень. Ціль полягає в формуванні істинної інформації про об’єкти реального світу на основі аналізу цих зображень.

Як технологічна дисципліна, комп'ютерний зір прагне застосувати теорії та моделі комп'ютерного зору до створення систем комп'ютерного зору. [19] Прикладами таких систем можуть бути:

- системи доповненої реальності (приклад: Microsoft HoloLens[4]);
- системи організації інформації (наприклад, для індексації баз даних зображень Google Inside Search [2]);
- системи моделювання об'єктів або навколишнього середовища (аналіз медичних зображень, топографічне моделювання);
- системи взаємодії (наприклад, пристрої введення для системи людино-машинного взаємодії Kinect[8]);
- системи управління процесами (промислові роботи, автономні транспортні засоби);
- системи відео спостереження.

Останні два напрями дуже тісно пов’язані із проблемою розпізнавання людини на графічному зображенні.

У даній курсовій роботі, розглядається реалізація програми яка вміє розпізнавати силуети об’єктів на зображенні.

Мета: дослідити сучасний стан розвитку систем комп’ютерного зору, створити програму, що здатна розпізнавати силуети об’єктів, які контрастно виражені на зображенні та підраховувати їх кількість. Надати можливість візуалізації роботи програми.

У відповідності з метою роботи були представлені наступні задачі:

- розглянути сучасні досягнення в сфері комп’ютерного зору, зокрема у сфері розпізнавання силуетів;
- розглянути теорії розпізнавання образів;
- розглянути відкриту бібліотеку комп’ютерного зору OpenCV;
- змоделювати алгоритм, що дозволяє розпізнавати силуети об’єктів на зображенні;
- створити власну програму на основі цього алгоритму, для розпізнавання силуетів на зображеннях наступного характеру:
    - фон зображення світлий, об’єкти – темні;
    - фон зображення може бути неоднорідним;
    - кольори фону зображення мають бути світлими;
    - тон кольорів фону має бути приглушеним;
    - об’єкти на зображенні мають бути чітко та контрастно виділені максимально темним кольором;
- програма повинна мати графічний інтерфейс та ілюструвати принцип роботи використаного алгоритму розпізнавання та підрахунку силуетів.

Об’єкт дослідження: технології комп’ютерного зору.

Предмет дослідження: розпізнавання силуетів засобами комп’ютерного зору.

Структура та об’єм. Курсова робота складається із вступу, двох розділів, списку використаних джерел (24 найменування), та додатків. Основний зміст викладено на 28 сторінках (без додатків).

У **вступі** описано вибір об’єкту дослідження та його актуальність, обґрунтовано мету та завдання.

У **першому розділі** проведено аналіз сучасних досягнень у сфері комп’ютерного зору, наведені теоретичні відомості щодо теорії розпізнавання образів, теорії поняття комп’ютерної графіка, однієї із бібліотек комп’ютерного зору та середовища програмування.

У **другому розділі** описано процес ідентифікації вимог до програмного продукту, викладено етапи проектування та розробки програмного продукту. Виконано огляд кінцевої програми, написано коротку інструкцію користування програмним додатком.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ДОСЯГНЕНЬ В СФЕРІ КОМП'ЮТЕРНОГО ЗОРУ

1.1. Теорія розпізнавання образів

Теорія розпізнавання образів — розділ кібернетики, що розвиває теоретичні основи й методи класифікації і ідентифікації предметів, явищ, процесів, сигналів, ситуацій і т. п. об'єктів, які характеризуються кінцевим набором деяких властивостей і ознак.

Задача розпізнавання образів — це задача віднесення вихідних даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних. [24]

У процесі біологічної еволюції багато тварин за допомогою зорового й слухового апарата вирішили задачу розпізнавання образів досить добре. Створення штучних систем розпізнавання образів залишається складною теоретичною й технічною проблемою. Необхідність у такому розпізнаванні виникає в самих різних областях — від військової справи й систем безпеки до оцифровування різних аналогових сигналів. [22]

Загалом виділяють два напрями в розпізнаванні образів:

- Вивчення, пояснення та моделювання здібностей до розпізнавання, якими володіють живі істоти.
- Розвиток самої теорії та методів побудови пристроїв, призначених для розв'язання окремих задач у прикладних цілях. [21]

Існує досить багато вже перевірених часом методів розпізнавання образів, наприклад:

1. Оптичне порівняння – метод перебору вигляду об'єкта під різними кутами, масштабами, зсувами й т. д.
2. Знаходження контуру об’єкта і дослідження його властивостей та форми.
3. Використання штучних нейронних мереж.

Найпоширенішими сферами практичного використання методів розпізнавання є: **технічна діагностика** на виробництві, як приклад – перевірка

вигляду деталі і прийняття певних заходів при виявлені дефектів; **медична діагностика**, діагностування різноманітних захворювань, на основі аналізу кардіограм, рентгенівських знімків чи знімків МРТ; **розпізнавання літер,** одна із найпоширеніших і найрозповсюджених проблем сьогодення у сучасному світі, так звані «капчі[1]» наочний тому приклад; **охоронні системи,** наприклад ідентифікація людини серед рухомих об’єктів на камерах спостереження, та аналіз цієї людини за певними критеріями та ознаками. [23]

Інформативність ознак — величина, яка кількісно характеризує придатність ознак (або їх набору) X для розпізнавання класів об'єктів.

У розпізнаванні образів як інформативність ознак використовуються умовна ентропія, ймовірність помилки розпізнавання, дисперсійна міра і інші величини. [17]

1.2. Бібліотека комп’ютерного зору OpenCV.

OpenCV[2][5] — бібліотека функцій та алгоритмів комп'ютерного зору, для обробки зображень і чисельних алгоритмів загального призначення з відкритим вихідним кодом.

Бібліотека розроблена компанією Intel і нині підтримується Willow Garage та Itseez. Бібліотека написана мовою C++ і поширюється під ліцензією BSD, що означає, що вона може вільно використовуватися в академічних та комерційних цілях.[3]

Бібліотека надає засоби для обробки і аналізу зображень, їх вмісту, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстеження руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях.

На перших етапах розвитку OpenCV основними були наступні задачі:

[1] CAPTCHA (англ. «**c**ompletely **a**utomated **p**ublic **t**uring test to tell **c**omputers and **h**umans **a**part» – повністю автоматизований публічний тест Тюринга для розрізнення комп'ютерів і людей.

[2] OpenCV (англ. Open Source Computer Vision Library, бібліотека комп'ютерного зору з відкритим кодом)

- Розвивати дослідження у напрямку комп'ютерного зору, забезпечуючи добре оптимізований та відкритий код бібліотеки.
- Поширювати знання у сфері комп'ютерного зору, забезпечуючи загальну інфраструктуру, яку б могли розвивати розробники, таким чином код ставатиме більш легким для сприйняття та обміну.
- Розвивати засновані на роботі з комп'ютерним зором комерційні додатки, створюючи не залежну від платформи, оптимізовану та безкоштовну бібліотеку. Для цього використовувалася ліцензія, яка не вимагала від таких комерційних додатків бути відкритими.

На сьогодні бібліотека містить понад 2500 оптимізованих алгоритмів, серед яких повний набір як класичних так і практичних алгоритмів машинного навчання і комп'ютерного зору [6]. Алгоритми OpenCV застосовують найрізноманітніших сферах, наприклад: аналіз та обробка зображень, системи з розпізнавання обличчя, ідентифікації об'єктів, розпізнавання жестів на відео, відстеження переміщення камери, побудова 3D моделей об'єктів, створення 3D хмар точок зі стерео камер, склеювання зображень між собою, для створення зображень всієї сцени з високою роздільною здатністю, система взаємодії людини з комп'ютером, пошук схожих зображень із бази даних, усування ефекту червоних очей при фотозйомці зі спалахом, стеження за рухом очей, аналіз руху, ідентифікація об'єктів, сегментація зображення, трекінґ відео, розпізнавання елементів сцени і додавання маркерів для створення доповненої реальності та багато інших.

Всі модулі бібліотеки зазначені на сайті некомерційної організації в розділі «OpenCV API Reference» [1].

1.3. Комп’ютерна графіка

Комп'ютерна графіка — це графіка, тобто зображення, яке створюється, перетворюється, оцифровується, обробляється і відображається засобами обчислювальної техніки, включаючи апаратні і програмні засоби.

У навчальному посібнику[16] надається таке означення: терміном комп’ютерна графіка позначають генерацію або оцінку графічних об’єктів ПЕОМ, маніпулювання ними, а також установлення зв’язку між графічними об’єктами і неграфічною інформацією, що знаходиться у файлах.

Комп’ютерну графіку можна розбити на області: зображувальна комп’ютерна графіка, аналіз зображення та перцептивна графіка (аналіз сцен).

1.4. Поняття растрової графіки

Комп'ютерна графіка	Зображувальна комп'ютерна графіка	**Об'єкти**: штучно створені зображення (символи, лінії) **Задачі**: побудова об'єкта та генерація зображення; ідентифікація об'єкта та отримання інформації. **Інструменти**: графопобудовники таінтерактивні системи
	Обробка і аналіз зображень	**Об'єкти**: фотознімки **Задачі**: поліпшення якості зображення (подавлення шуму, збільшення контрасності); оцінка зображення; розпізнавання образів **Інструменти**: скануючі пристрої, спеціальне програмне забезпечення.
	Перцептивна графіка (аналіз сцен)	**Об'єкти**: зображення сформовані машиною, або виділені з фотографії **Задачі**: розпізнавання прототипів і деяких взаємозв'язків між прототипами **Інструменти**: евристичні алгоритми, спеціальне програмне забезпечення

Растрову графіку застосовують під час розробки електронних і поліграфічних видань. Ілюстрації, виконані засобами растрової графіки, рідко

створюють вручну, за допомогою комп'ютерних програм. Частіше всього застосовують відсканованої ілюстрації, підготовлені художником на папері, чи фотографії. Останнім часом для введення растрових зображень у комп'ютер знайшли широке застосування цифрові фото- і відеокамери. Відповідно, більшість растрових графічних редакторів орієнтовані не стільки на створення зображень, скільки на їх обробку.

Растр – це прямокутна сітка точок, формуючих зображення на екрані монітора.

Кожна точка растру характеризується двома параметрами: своїм положенням на екрані і своїм кольором, якщо монітор кольоровий, чи ступенем яскравості, якщо монітор чорно-білий.

Растрова графіка являє собою зображення у вигляді масиву цифр. Тому при великому збільшенні всі зображення виглядають як мозаїка (сітка), яка складається з найменших комірок.

Сама сітка отримала назву растрової карти (bitmap), а її формуючі елементи називаються пікселем.

Пікселі (pixel) – це найменший елемент зображення, відтворюючий комп'ютером.

Відмінними особливостями пікселя є його: однорідність (всі пікселі за розміром однакові) і неподільність (всередині пікселя не може бути ніяких більш дрібних елементів).

Якщо пікселі досить малі, то око сприймає «піксельну мозаїку» як одне ціле зображення.

Під час масштабування растрових зображень виникають спотворення – **східці** (aliasing).

Растрові редактори дозволяють частково прибрати ці спотворення за рахунок застосування спеціальних алгоритмів обробки. Ці операції називаються anti-aliasing. [20]

1.5. Силует

Силует — узагальнене площинне однотонне зображення постаті або предмета, форма яких окреслюється тільки контуром. Іншими словами: силует — зображення якогось об'єкта лише контуром при збереженні характерних рис, що дають змогу розпізнати цей об'єкт. Назва походить від прізвища французького міністра фінансів Етьєна де Силуета, який був настільки скупим, що його карикатуру зобразили лише у вигляді затемненого абрису.

Зазвичай в мистецтві силует виконується чорною тушшю на білому, витинається з темного паперу й наклеюється на ясне тло (може бути й навпаки). Виражальна скупість силуету вимагає від художника або декоратора пошуку лаконічних, легко зрозумілих форм. При створенні силуету важливими є рівновага площин, поєднання в абрисі різких та плавних контурів. Лаконічність силуету часто використовують при створенні емблем, алегорій, символів тощо.[9] Рідше силует може виступати виражальним засобом у художній фотографії, графіці.

1.6. Середовище Qt

1.6.1. Загальні відомості про Qt

Qt[7] – кросплатформовий інструментарій розробки програмного забезпечення мовою програмування C++ (але не тільки, Qt підтримує щонайменше ще 10 мов). Qt дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем, просто компілюючи текст програми спеціально для кожної операційної системи без зміни сирцевого коду (звісно, можна зробити статичне лінкування[1] бібліотек, але це значно збільшить розмір виконуваного файлу, оскільки для коректної роботи потрібно буде злінкувати чимало бібліотек). Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє

[1] англ. link – «посилання»

керувати нитями, працювати з мережею та забезпечує кросплатформовий доступ до файлів.

Чим Qt такий унікальний? Qt використовує Meta Object Compiller (MOC) – система попередньої обробки початкового коду (загалом, Qt – це бібліотека не для чистого C++, а для його «діалекту» (якщо так можна виразитись). Qt, за допомогою MOC, перекладає цей «діалект» на чистий C++ для подальшої компіляції будь-яким стандартним C++ компілятором). Qt комплектується графічним середовищем розробки графічного інтерфейсу QTDesigner (як MFC, тільки нормальний). QTDesigner дозволяє створювати діалогові вікна і форми «мишею» та «в два кліки». Ідеологія створення форм у Qt базується на використанні менеджерів розташування, котрі надають «гумовий» дизайн, при якому розмір і розташування елементів форм визначаються автоматично, що значно прискорює розробку графічного інтерфейсу, так як програмісту не треба витрачати зайвий дорогоцінний час на «совгання» кнопочок та інших елементів по вікну створюваної форми.

1.6.2. Загальні бібліотеки Qt, що необхідні для розробки GUI додатку

QtCore [10]

Головний модуль, ядро Qt. Всі інші модулі Qt залежать від цього модуля і не будуть працювати без нього.

QtGui [14]

Модуль QtGui розширює QtCore додаючи функції GUI.

QApplication [11]

Клас qapplication управляє потоками контролю застосування GUI та здійснює його основні налаштування.

Клас Qapplication містить основний цикл обробки подій, де обробляються всі події від віконної системи та інших джерел. Він також відповідає за ініціалізацію і завершення програми, та забезпечує керування сеансами. Крім того, qapplication обробляє більшість загальносистемних параметрів програми в цілому.

Для будь-якого додатку із графічним інтерфейсом з використанням Qt, існує як мінімум один об'єкт qapplication, незалежно від того, яка кількість вікон у цього додатка.

QMainWindow [13]

Клас QMainWindow надає головному вікні програми. Головне вікно забезпечує основу для побудови користувацького інтерфейсу програми. В Qt є QMainWindow і пов'язані з ним класи для управління головним вікном. QMainWindow має свій власний макет, в який можна додавати QToolBars[1], QDockWidgets[2], а QMenuBar[3], і QStatusBar[4] (рис 1.6.2.1). Макет має центр, який може бути зайнятий будь-яким віджетом.

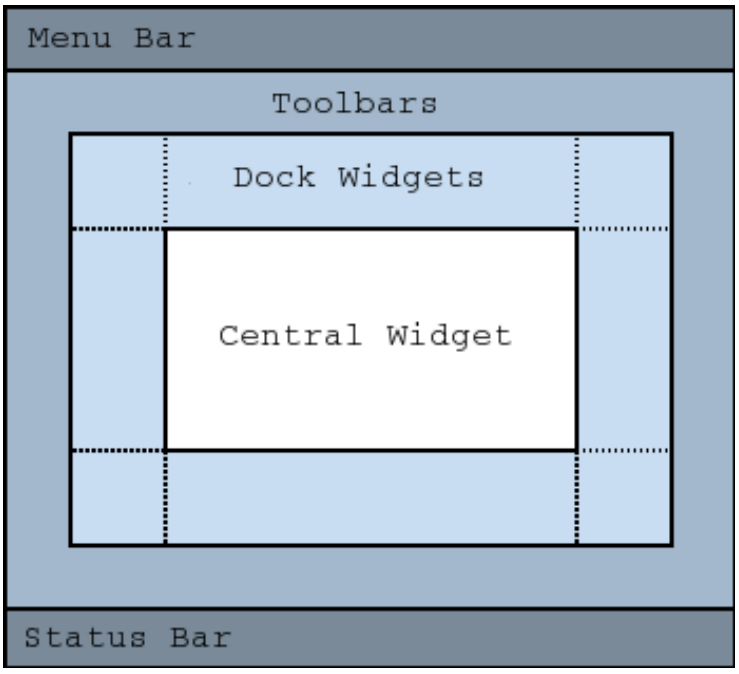

Рис. 1.6.2.1. Структура вікна QMainWindow

QWidget [15]

Клас qwidget є базовим класом для всіх об'єктів користувальницького інтерфейсу. Віджет – це атомарна одиниця інтерфейсу: він отримує

[1] QToolBars – клас успадкований від QWidget, забезпечує реалізацію рухливої панелі, яка містить набір елементів управління.
[2] QDockWidgets – клас успадкований від QWidget, надає віджет, який можна закріпити всередині QMainWindow або переміщати як вікно верхнього рівня на робочому столі.
[3] QMenuBar – клас успадкований від QWidget, забезпечує реалізацію горизонтального меню.
[4] QStatusBar – клас успадкований від QWidget, забезпечує реалізацію горизонтального статус-бару.

контролер миші, клавіатури та інших подій від вікона, і «малює себе» на екрані.

Віджет, який не вбудований в батьківський віджет (не має бітьківського віджета), називається **вікном**. Як правило, вікна мають **рамку** і **заголовок вікна**, хоча можна створювати вікна без такого декору, використовуючи відповідні прапорці). В Qt, QMainWindow і різні підкласи QDialog є найбільш поширеними типами вікон.

Кожен конструктор віджета приймає один або два стандартних аргументи:

- **QWidget *parent = 0** – батько нового елементу. Якщо прапорець дорівнює 0 (за замовчуванням), новий віджет буде вікном (не маэ батьківського елемента). Якщо прапорець не нульовий, то це буде дочірній елемент, і цей об’єкт буде обмежуватись полями батьківського (якщо власноруч не встановити відповідний флаг WindowFlags).
- **Qt::WindowFlags f = 0** (при наявності) встановлює прапорець вікна; значення за замовчуванням підходить практично для всіх віджетів. Встановлення прапорця у ненульове положення майже не потрібне для більшості задач.

Клас qwidget має багато функцій, але деякі з них не мають безпосередньої функціональность. Існує безліч підкласів, які забезпечують реальні можливості, такі як QLabel, екземпляр об'єкта qpushbutton, QListWidget, qtabwidget, і т.д..

РОЗДІЛ 2. ОПИС, ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

2.1. Ідентифікація вимог

1. Програма повинна бути реалізована на мові програмування C++.
2. Програма повинна працювати із файлами мінімум одного із зазначених форматів: *.jpg, *.bmp, *.gif, *.png.
3. Програма повинна розпізнавати силуети об’єктів на зображенні*.
4. Програма повинна підраховувати кількість силуетів об’єктів на зображенні.

   Під зображенням мається на увазі наступне:

   - Графічне зображення в одному із вищезазначених форматів.
   - Фон зображення світлий[1], об’єкти – темні[2].
   - Фон зображення може бути неоднорідним та різних кольорів.
5. Програма повинна мати функцію, що дозволяє візуалізувати процес роботи програми.

2.2. Технічне завдання

1. ВСТУП

   1.1. Назва програми:

   Повна назва – Silhouettes Counter (Лічильник силуетів)

   1.2. Область використання:

   Silhouettes Counter призначений для підрахунку кількості силуетів об’єктів на зображенні. Код програмного додатку не прив’язаний до конкретної операційної системи. Для роботи програми необхідні бібліотеки Qt[3].
2. ПРИЗНАЧЕННЯ РОЗРОБКИ

   Розробка програмного додатку є основою курсової роботи.
3. ЗАДАЧІ ПРОГРАМИ

[1] Під «світлим» розуміється колір, яскравість (brightness) якого більше 75%.
[2] Під «темним» розуміється колір, яскравість (brightness) якого менше 25%.
[3] Qt – крос-платформовий інструментарій розробки програмного забезпечення мовою програмування C++.

Виходячи із вимог, що були виявлені, було поставлено наступні задачі:

3.1. Підтримка мінімум одного з наступних графічних форматів:

3.1.1. ***.jpg** – JPEG (Joint Photographic Experts Group) — формат файлу зображень растрової графіки, що використовує стиснення з втратами.

3.1.2. ***.bmp** – bitmap-формат або DIB (Device Independent Bitmap) – формат файлу зображень растрової графіки, в якому зображення зберігається у вигляді двовимірного масиву пікселів.

3.1.3. ***.gif** – GIF (Graphics Interchange Format) – 8-бітний растровий графічний формат, що використовує до 256 чітких кольорів із 24-бітного діапазону RGB. Лише статичне зображення.

3.2. Підготовка зображення:

Очищення зображення від зайвої інформації (бінаризація).

3.3. Розпізнавання об’єктів:

Розпізнавання об’єктів на вже очищеному зображенні.

3.4. Підрахунок кількості силуетів об’єктів.

3.5. Візуалізація роботи алгоритму очищення зображення та розпізнавання об’єктів. (Для візуалізації використовувати зображення розміром, що не перевищує 320*240px)[1]

2.3. Розробка програмного додатку

2.3.1. Визначення підходу до вирішення проблеми.

Існує досить багато вже перевірених часом методів розпізнавання образів, наприклад:

1. Оптичне порівняння – метод перебору вигляду об'єкта під різними кутами, масштабами, зсувами й т. д.
2. Знаходження контуру об’єкта і дослідження його властивостей та форми.
3. Використання штучних нейронних мереж.

[1] Приблизний розмір зображення був виведений експериментально, про це далі по тексту роботи. За основу розміру був взятий формат QVGA.

Проте очевидно, що кожен із цих методів має як свої позитивні сторони, так і недоліки. Так метод перебору вимагає від оператора заздалегідь підготовлене зображення і «правильну» відповідь. За допомогою цього метода машина «навчається» розпізнавати і знаходити заданий об’єкт під різними кутами, масштабами, зсувами і т.д.. Для моєї задачі цей метод не є ідеальним, так як мені потрібно створити універсальну програму для розпізнавання будь-яких, нових, різних образів на зображенні.

Метод із використання штучних нейронних мереж досить складний як і для вивчення, так і для реалізації. Окрім цього, цей метод вимагає або великої кількості прикладів задачі розпізнавання (знову ж таки із правильними відповідями), або спеціальної структури нейронної мережі, що враховує специфіку даної задачі. Цей метод також не повністю задовольняє мої потреби для вирішення поставленого завдання.

Найближчим з усіх виявився метод знаходження контуру об’єкта і аналізу його властивостей.

На даному етапі, все, що потрібно для створення програми із зазначеними у вступі вимогами, це знайти об’єкт, перевірити чи задовольняє він певним умовам відбору, та підрахувати кількість таких об’єктів.

Для створення програми, я вирішив не використовувати готовий код, тобто якісь зовнішні фреймворки, бібліотеки комп’ютерного зору, чи будь-що написане до мене і не мною, що стосувалося б саме комп’ютерного зору.

Було вирішено власноруч «придумати» досить просту, але водночас ефективну логіку роботи програми. Ця ціль була досягнена. Логіка програми полягає в наступному:

- В програму завантажується файл зображення.
- Проходить процес бінаризації (очищення зображення).
- Проходить процес пошуку силуетів.

2.3.2. Опис і принципи роботи програми

Завантаження файлу зображення.

Завдяки бібліотекам від Qt, а саме за допомогою бібліотеки QImage[12], завантажене зображення представляється як двовимірний масив пікселів із певними характеристиками (колір у форматі RGB, яскравість, насиченість тощо.)

Бінаризація зображення.

Двовимірний масив пікселів перетворюється у бінарний масив наступним чином: програма аналізує кожен піксель за певним принципом, і двовимірний масив пікселів із різними властивостями перетворюється на бінарний масив.

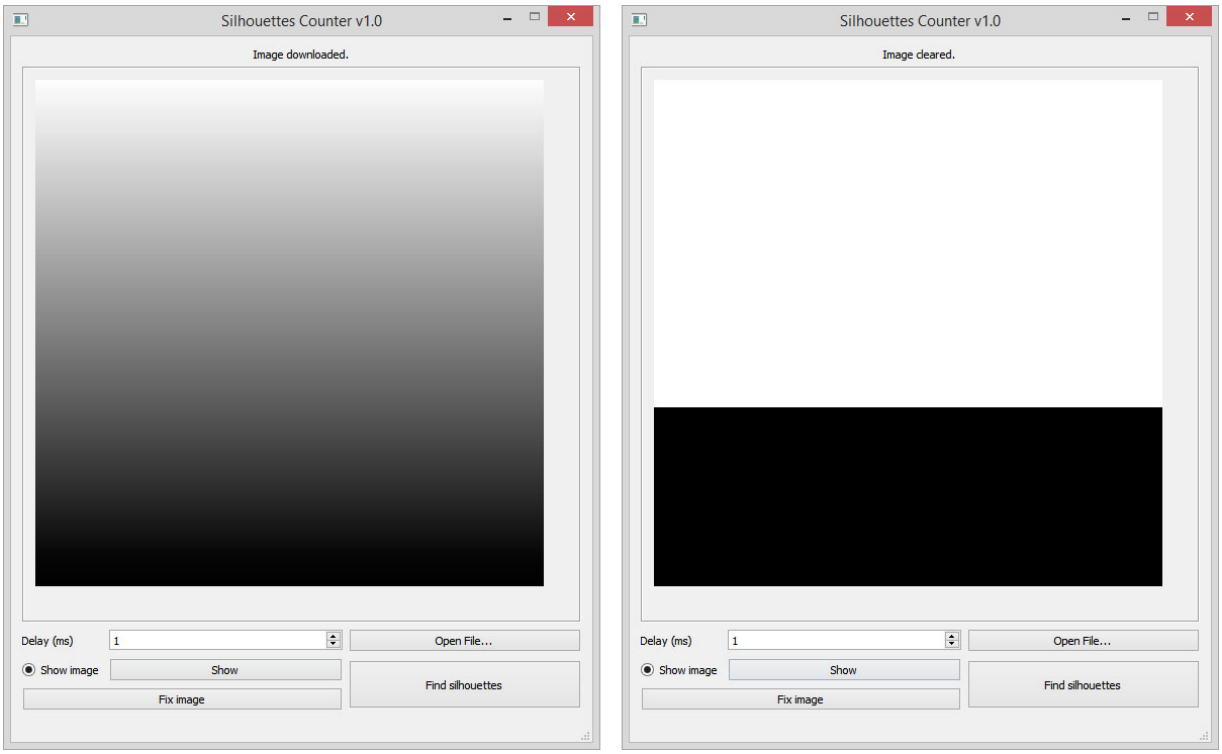

Рис.2.3.2.1. Оригінал (зліва) та результат бінаризації (справа), чорно-білого градієнту.

Після даного етапу маємо бінарний масив таких самих розмірів, але із пікселями лише двох видів: білі, на місці, де «на думку програми» не має об’єкта, і чорні – де він є.

«Думає» програма за наступним принципом:

- Якщо значення параметру lightness[1] поточного пікселю більше чи рівне 90, то цей піксель перетворюється на білий піксель. Тобто програма вважає, що цей піксель належить фону зображення.
- Якщо значення параметру lightness поточного пікселю менше за 90, то цей піксель перетворюється на чорний піксель. Тобто програма вважає, що цей піксель належить об’єкту, який ми розпізнаємо на зображенні.

Пошук об’єктів.

На даному етапі маємо двовимірний масив чорних і білих пікселів.

Рис.2.3.2.2.Процес пошуку силуетів на зображенні.

Програма починає аналізувати пікселі. Логіка процесу розпізнавання силуетів полягає у аналізі всіх пікселів нашого зображення. Програма знаходить силуети один за одним аналізуючи все зображення. Оскільки наше

[1] В представлення зображення за допомогою класу QImage, кожен піксель має набір властивостей. Параметр lightness (англ. освітленість) лежить в межах від 0 (чорний, темний, не освітлений) до 255 (світлий, засвічений).

зображення – це набір точок, кожен силует можна представити у вигляді графа. Коли аналізатор «натикається» на силует (на граф із чорних точок), програма починає «обхід» цього графа за наступним алгоритмом:

Розглянемо випадок, коли курсор (1) знаходиться на пікселі чорного кольору. Колір інших пікселів невідомий.

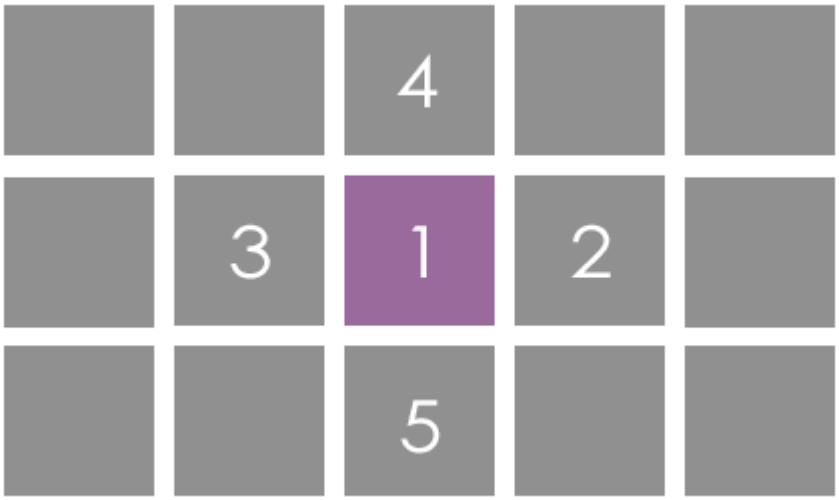

Рис.2.3.2.3. Схема «бачення» сусідніх пікселів (2,3,4,5) програмним «курсором» (1).

- 1. Якщо піксель (1) – чорний, помічаємо піксель червоним кольором, і курсор переміщується на 1 піксель вправо, на піксель (2).
- 2. Якщо піксель (2) – чорний, програма повертається до кроку №1.
- 3. Якщо піксель (2) – білий, курсор переміщується на 1 піксель вліво (на піксель (1)).
- 4. Після повернення «з права», курсор переміщується на 1 піксель вправо, тобто на піксель (3).
- 5. Якщо піксель (3) – чорний, програма повертається до кроку №1.
- 6. Якщо піксель (3) – білий, курсор переміщується на 1 піксель вправо (на піксель (1)).
- 7. Після повернення «з ліва», курсор переміщується на 1 піксель вгору, тобто на піксель (4).
- 8. Якщо піксель (4) – чорний, програма повертається до кроку №1.

- 9. Якщо піксель (4) – білий, курсор переміщується на 1 піксель вниз (знову до пікселя (1)).
- 10. Після повернення «з верху», курсор переміщується на 1 піксель вниз, на піксель (5).
- 11. Якщо піксель (5) – чорний, програма повертається до кроку №1.
- 12. Якщо піксель (5) – білий, курсор повертається на 1 піксель вверх (на піксель (1)).
- Всі напрями від пікселя (1) перевірено, програма робить висновок, що силует повністю знайдено.
- Програма інкрементує лічильник силуетів.
- Курсор переходить на 1 піксель вліво, або на новий рядок, якщо це кінець рядка пікселів.
- Аналіз зображення продовжується.

Червоні пікселі для курсора еквівалентні білим (програма пропускає червоні пікселі через те, що якщо вони червоні, то вони вже були перевірені раніше).

На початку роботи частини коду, що відповідає за пошук силуетів в програмі оголошується локальна змінна, яка буде містити кількість пікселів поточного силуету. При кожному «маркуванні» чорного пікселя в червоний, ця змінна-лічильник інкрементується. Таким чином, після повного аналізу поточного силуету лічильник містить число пікселів, які займає даний силует. Це число порівнюється із «магічним» числом що задано всередині програми. Аналізатор відкидає всі силуети, розмір яких менший на 0.5% від всього розміру зображення. Тобто при зображенні розміром 320*240px, чорні цятки, розміром менші за 384 пікселі – не враховуються лічильником силуетів. Дане число (0.5%) було виведено експериментальним шляхом.

Бінаризація зображення, частково, стала проблемою. Оскільки початкове зображення не ідеальне, після процесу бінаризації може з’явитись так званий «шум» – чорні цяточки поблизу силуета (див. Рис.2.3.2.4.).

Такі «плями» з’являється не часто, проте має бути механізм, що правильно їх аналізує. Саме через це було прийнято рішення відсікати, і не рахувати такі «плями». Оскільки розмір таких чорних цяток досить малий відносно всього зображення (як вже було сказано – 0.5% від кількості пікселів зображення), такими плямами можна знехтувати при підрахунку кількості силуетів.

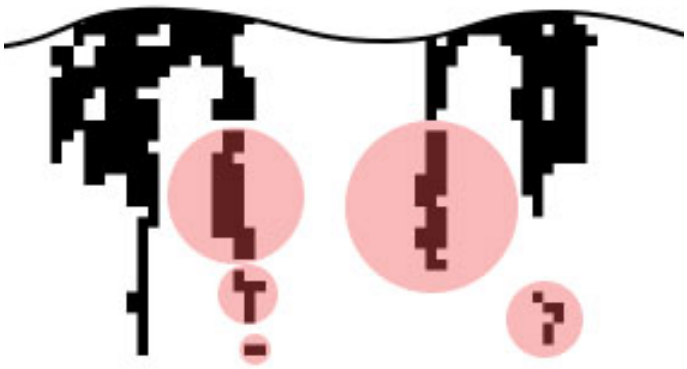

Рис.2.3.2.4. «Шум» на зображенні після процесу бінаризації.

Після завершення роботи алгоритму пошуку силуетів, випливає діалогове вікно, що сповіщає про завершення роботи програми та містить інформацію про кількість силуетів знайдених на зображенні (див. Рис.2.3.2.5).

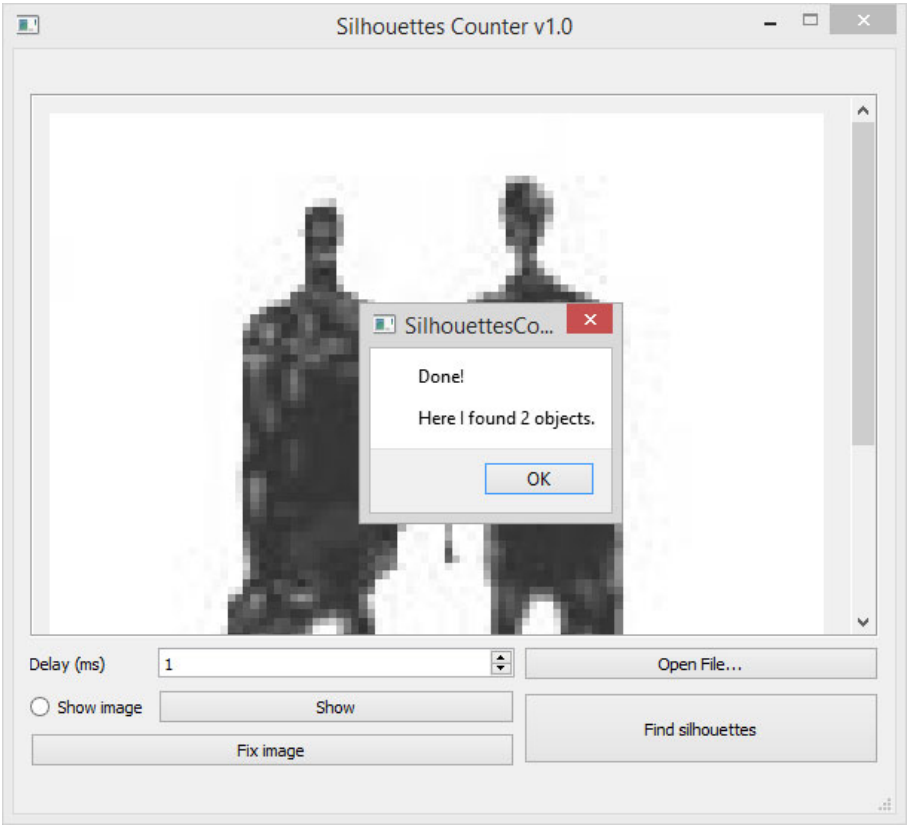

Рис.2.3.2.5. Кінцеве вікно програми та діалогове вікно.

2.4. Огляд продукту та інструкція користувача.

Готовий програмний продукт отримав назву «Silhouettes Counter».

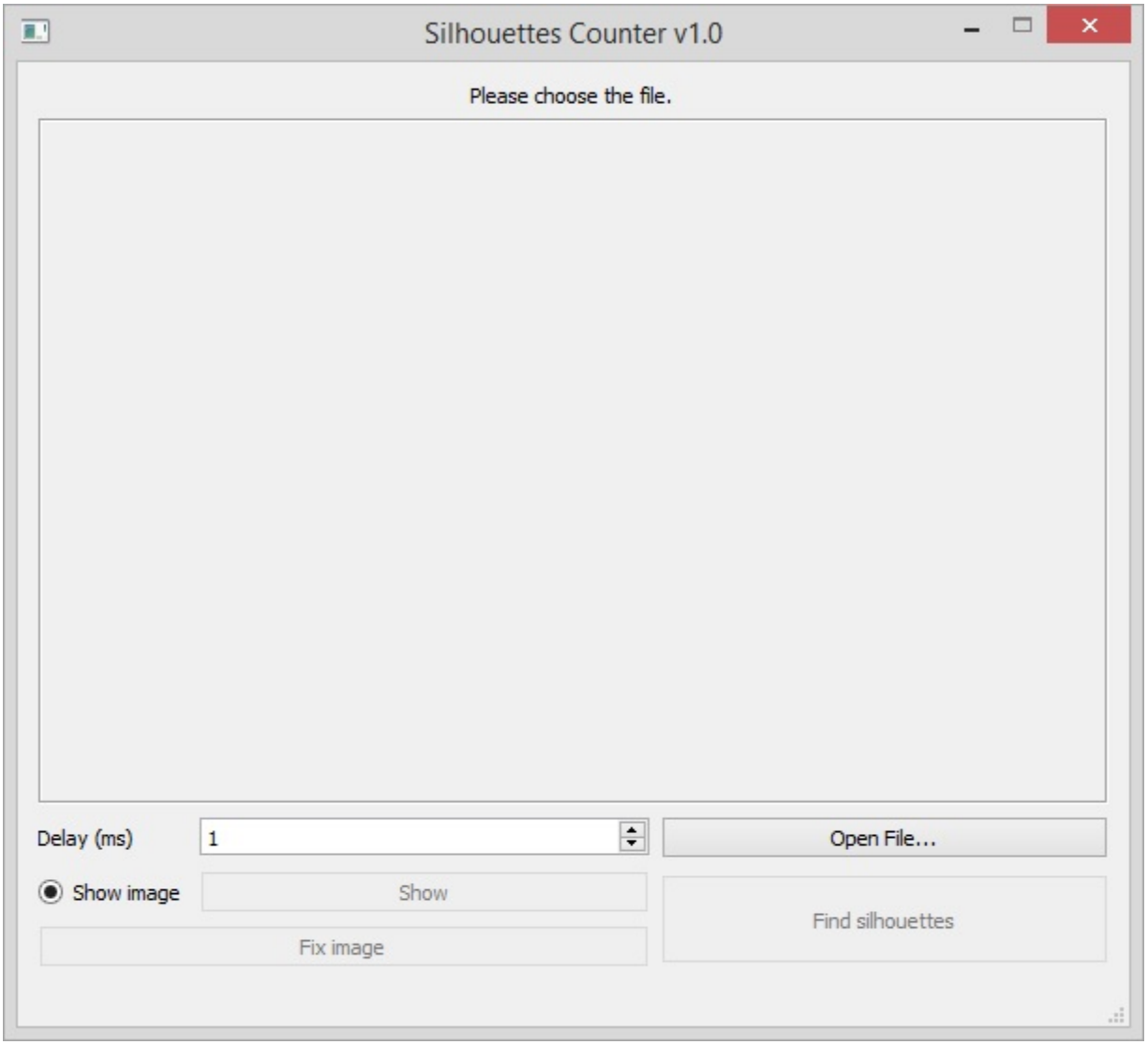

Рис.2.4.1. Стартовий вигляд вікна програми при запуску.

Весь програмний код продукту, тестові зображення, виконуваний exe-файл у zip-архіві, викладений на веб-сервісі github.com в репозиторії github.com/VladOliynyk/SilhouettesCounter. Також програмний код доступний в додатках А-Е.

2.4.1. Головне вікно програми.

Головне вікно програми містить наступні компоненти: рядок заголовку, кнопку згортання, кнопку закриття програми, робочу область (див. Рис.2.4.1.1.а). В свою чергу, робоча область містить в собі: заголовок, вікно попереднього перегляду, поле для виставлення затримки між ітераціями

циклу процесу баніразації зображення та процесу пошуку силуетів, кнопку завантаження файлу, перемикач відображення зображення, кнопку примусового відображення зображення, кнопку запуску процесу бінаризації, кнопку запуску процесу пошуку силуетів (див. Рис.2.4.1.1.б).

Рис.2.4.1.1.а. Головне вікно програми.
1 – рядок заголовку, 2 – кнопка згортання, 3 – кнопка закриття, 4 – робоча область програми.

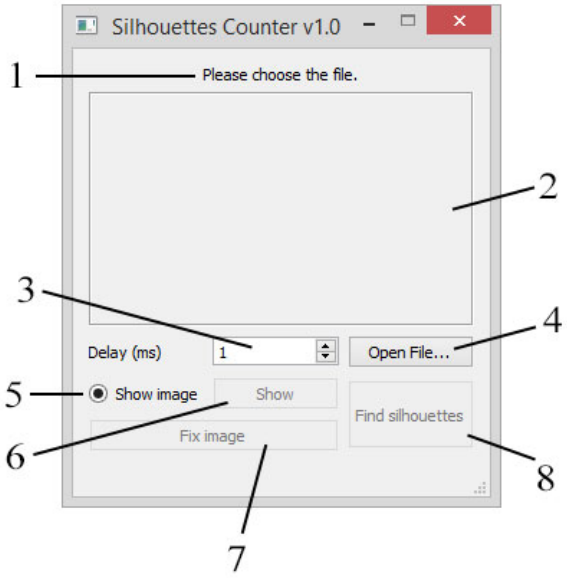

Рис.2.4.1.1.б. Робоча область програми.
1 – заголовок, 2 – вікно попереднього перегляду, 3 – поле для виставлення затримки, 4 –кнопка завантаження файлу, 5 – перемикач відображення, 6 – кнопка «показу» зображення, 7 – кнопка запуску процесу бінаризації, 8 – кнопка запуску процесу пошуку силуетів.

2.4.2. Інструкція користувача.

На початку роботи програми доступною є лише кнопка вибору файлу («Open File…»). Кнопки «Show» та «Fix image» э неактивними.

Завантаження (відкриття) файлу.

Для того, щоб завантажити файл у програму, потрібно натиснути на кнопку «Open File…». Після цього відкриється стандартне діалогове вікно вибору файлу із жорсткого диску комп’ютера (програма підтримує наступні формати графічних файлів: *.bmp, *.jpg, *.gif). Після вибору файлу натисніть

«Відкрити» у діалоговому вікні вибору файлу. Файл буде завантажено в програму, про що буде свідчити напис над вікном попереднього перегляду (перед вибором файлу там буде міститись повідомлення «Please choose the file.», після коректного вибору файлу повідомлення зміниться на «Image downloaded.»). Якщо зображення завантажено не коректно, або не завантажено через будь-які обставини, повідомлення буде наступним: «Please choose the file!».

Після коректного завантаження файлу, кнопки «Show» та «Fix image» стануть активними.

Початкові налаштування.

Якщо потрібно, можна виставити затримку (у мілісекундах), між ітераціями роботи циклу програми (незалежно, це бінаризація зображення чи пошук силуетів). Затримка потрібна для спрощення та покращення якості демонстрації логіки програми.

Якщо потрібно показувати зображення і процес роботи програми, перемикач «Show image» залишають увімкненим.

Якщо показувати зображення і процес роботи програми не потрібно, перемикач «Show image» вимикають.

Кнопка «Show»

Кнопка «Show» показує поточне зображення на полі для попереднього перегляду. Цю кнопку використовують у наступних випадках:

- після завантаження зображення, для перевірки коректності вибору зображення;
- після завершення процесу бінаризації, якщо показ зображення було вимкнуто;
- після завершення процесу пошуку силуетів, якщо показ зображення було вимкнуто;

Кнопка «Fix image»

Кнопка «Fix image» відповідає за початок роботи процесу бінаризації зображення. Кнопка «Fix image» стає доступною після коректного завантаження зображення в програму.

Після натискання на кнопку, буде запущено процес бінаризації зображення. Якщо перемикач показу зображення увімкнено, на полі попереднього перегляду буде змінюватись картинка, показуючи процес бінаризації.

Кнопка «Find silhouettes»

Кнопка «Find silhouettes» відповідає за початок роботи процесу пошуку силуетів. Кнопка «Find silhouettes» стає доступною після завершення процесу бінаризації зображення.

Після натискання на кнопку, буде запущено процес пошуку силуетів. Якщо перемикач показу зображення увімкнено, на полі попереднього перегляду буде змінюватись картинка, показуючи процес пошуку бінаризації.

Поле затримки («Delay»)

Поле затримки є активним протягом всієї роботи програми. Змінювати значення затримки можна під час виконання будь-якого процесу. Якщо перемикач показу зображення увімкнено, програма буде робити затримку відповідної довжини (у мілісекундах) після кожної ітерації процесу бінаризації, чи процесу пошуку силуетів.

ВИСНОВКИ

Комп’ютерний зір – теорія та технологія створення машин, які можуть проводити виявлення, стеження та класифікацію об'єктів. Розробка систем комп’ютерного зору є надзвичайно важливою складовою науково-технологічного розвитку. Комп'ютерний зір, обробка зображень, і машинний зір — тісно пов'язані області.

Комп’ютерний зір потрібен на виробництві, при керуванні роботами, для автоматизації процесів, в певних навчальних, медичних і військових додатках, при спостереженні, при вивченні будь-яких астрономічних об’єктів і, звісно, при роботі з персональними комп'ютерами. У всіх цих областях комп’ютерний зір виконує одну глобальну функцію – аналіз зображення. На основі проаналізованої і зібраної інформації робляться певні висновки і приймаються рішення. Якщо в сфері навчання чи домашнього користування, помилки при роботі додатків і сервісів комп’ютерного зору можуть бути допустимі, в областях точної науки, медицини, в військовій справі помилки не допускаються. Саме тому технології комп’ютерного надзвичайно вимогливі до якості їх розробки.

У курсовій роботі було розглянуто сучасний стан досягнень в області комп’ютерного зору, зокрема досягнення в сфері систем та способів обробки та аналізу зображень. Розглянуто основний метод створення простої системи розпізнавання образів на контрастному зображенні.

У відповідності із поставленими задачами було:

- розглянуто сучасні досягнення в сфері комп’ютерного зору, а саме у сфері розпізнавання силуетів;
- розглянуто теорію розпізнавання образів (частково);
- розглянуто основні відомості про бібліотеку OpenCV – відкриту бібліотеку комп’ютерного зору;
- змодельовано та реалізовано алгоритм, що дозволяє розпізнавати силуети об’єктів на зображенні;

- на основі алгоритму створено власну програму, для розпізнавання силуетів на зображеннях наступного характеру:
    - фон зображення світлий, об’єкти – темні;
    - фон зображення може бути неоднорідним;
    - кольори фону зображення мають бути світлими;
    - тон кольорів фону має бути приглушеним;
    - об’єкти на зображенні мають бути чітко та контрастно виділені максимально темним кольором;
- програма має графічний інтерфейс, також в програмі реалізована візуалізація принципів роботи програми і використаного алгоритму бінаризації зображення та розпізнавання і підрахунку силуетів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. About OpenCV. [Електронний ресурс]. Режим доступу: http://docs.opencv.org/2.4/modules/refman.html (30 березня 2016)
2. Google Inside Search. [Електронний ресурс]. Режим доступу: https://www.google.com/intl/es419/insidesearch/features/images/searchbyimage.html (1 квітня 2016)
3. Learning OpenCV / Gary Bradski, Adrian Kaehler. – O'REILLY Media, Inc.,1005 Gravenstein Highway North, Sebastopol. USA. 2008.
4. Microsoft HoloLens. [Електронний ресурс]. Режим доступу: https://www.microsoft.com/microsoft-hololens/en-us (30 березня 2016)
5. Open Source Computer Vision Library. Режим доступу: http://opencv.org/
6. OpenCV Wiki. [Електронний ресурс]. Режим доступу: https://github.com/Itseez/opencv/wiki (30 березня 2016)
7. Qt-creator. [Електронний ресурс]. Режим доступу: https://www.qt.io/ide/ (31 березня 2016)
8. Xbox.com: Kinect / Microsoft. [Електронний ресурс]. Режим доступу: (Архівна копія від 22 вересня 2010) https://web.archive.org/web/20100922111751/http://www.xbox.com/en-US/kinect
9. Декоративно-прикладне мистецтво / Антонович Є. А., Захарчук-Чугай Р. В., Станкевич М.Є. – Львів.: Видавництво «Світ», 1992. — С.258.
10. Документація QtCore. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-5/qtcore-index.html (31 березня 2016)
11. Документація QApplication. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-4.8/qapplication.html (31 березня 2016)
12. Документація QImage. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-4.8/qimage.html (31 березня 2016)
13. Документація QMainWindow. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-4.8/QMainWindow.html (31 березня 2016)

14. Документація QtGui. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-4.8/qtgui-module.html (31 березня 2016)
15. Документація QWidget. [Електронний ресурс]. Режим доступу: http://doc.qt.io/qt-4.8/qwidget.html (31 березня 2016)
16. Електронний посібник з дисципліни: Комп'ютерна графіка / к.т.н., доцент Федік Л. Ю., – Луцьк 2012. Режим доступу: http://elib.lutsk-ntu.com.ua/book/knit/auvp/2012/12-14/ (29 березня 2016)
17. Задача распознавания образов с точки зрения математической статистики. В кн.: Читающие автоматы и распознавание образов. / Ковалевский В. А. – К., 1965;
18. Комп'ютерний зір – Computer Vision. / Л.Шапіро, Дж. Стокман — М.: Біном. Лабораторія знань, 2006. – С.752.
19. Компьютерное зрение / Л. Шапиро, Дж. Стокман; Пер. с англ. – М.:БИНОМ. Лаборатория знаний, 2006. – 752 с., 8 с. ил.: ил.
20. Компьютерное зрение. [Електронний ресурс]. Режим доступу: https://events.yandex.ru/lib/talks/701/ (29 березня 2016)
21. Компьютерное зрение. Современный поход / Дэвид Форсайт, Жан Понс.;Пер. с англ. – М.: Издательский дом «Вильямс», 2004 – 928с.: ил. – Парал. тит. англ.
22. Методы распознавания / А. Л. Горелик, В. А. Скрипкин М.: Высшая школа, 1989.
23. Принципы распознавания образов / Ту Дж., Гонсалес Р. – М. 1978
24. Теория распознавания образов / В. Н. Вапник, А. Я. Червоненкис М.: Наука, 1974.

ДОДАТКИ

Додаток А. Файл головної програмної функції main.

```
//main.cpp
#include "window.h"
#include <QApplication>

int main()
{
    QApplication a(argc, argv);
    window w;
    w.show();

    return 0;
}
```

Додаток В. Модуль відповідальний за вікно програми.

```
//window.cpp
#include "window.h"
#include "ui_window.h"
#include "logic.h"

int delayMS = 10;
template <typename T>
void delay(T ui)
{
    if (getShow()) {
        delayMS = ui->delayBox->value();
        QTime dieTime= QTime::currentTime().addMSecs(delayMS);
        while (QTime::currentTime() < dieTime)
            QCoreApplication::processEvents(QEventLoop::AllEvents,
delayMS/10);
    }
}

window::window(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::window)
{
    ui->setupUi(this);
    ui->label->setText("");
    ui->showButton->setEnabled(false);
    ui->fixImageButton->setEnabled(false);
    ui->findSilhButton->setEnabled(false);
}

window::~window()
{
    delete ui;
}

void window::on_openFileButton_clicked()
{
    //getting file name & path
    QString userFilename = QFileDialog::getOpenFileName(
                this,
                tr("Open file..."),
                "./",
                "All supported files... (*.jpg; *.gif; *.bmp)"
                );
    //saving user image
    QImage tempImage(userFilename);

    if (userFilename.size() != 0) {
        saveImage(tempImage);
```

```
        ui->titleLabel->setText("Image downloaded.");
        //enabling buttons
        ui->showButton->setEnabled(true);
        ui->fixImageButton->setEnabled(true);
    } else {
        ui->titleLabel->setText("Please choose the file!");
        //enabling buttons
        ui->showButton->setEnabled(false);
        ui->fixImageButton->setEnabled(false);
    }
}

void window::on_showButton_clicked()
{
    showImage(ui, getImage());
}

void window::on_fixImageButton_clicked()
{
    saveImage(fixImage(ui, getImage()));
}

void window::on_findSilhButton_clicked()
{
    findSilhouettes(ui, getImage());
}

void window::on_showRadioButton_toggled(bool checked)
{
    if (checked) {
        setShow(true);
    } else {
        setShow(false);
    }
}
```

Додаток С. Заголовковий файл вікна програми.

```
//window.h
#ifndef WINDOW_H
#define WINDOW_H
#include <QMainWindow>
namespace Ui {
class window;
}
class window : public QMainWindow
{
    Q_OBJECT
public:
    explicit window(QWidget *parent = 0);
    ~window();
private slots:
    void on_openFileButton_clicked();
    void on_showButton_clicked();
    void on_fixImageButton_clicked();
    void on_findSilhButton_clicked();
    void on_showRadioButton_toggled(bool checked);
private:
    Ui::window *ui;
};
#endif // WINDOW_H
```

Додаток D. Модуль логіка програми.

```
//logic.h
#ifndef LOGIC_H
#define LOGIC_H
#include <QFileDialog>
#include <QMessageBox>
#include <iostream>
#include <QTime>

/* The global image variable for work with an image from the whole
scope. */
QImage GLOBAL_IMAGE;

/* The magic number: lightness filter for clearing image process. */
int LIGHTNESS = 90;

/* The magic number: the mask -
 *  interest characteristics of the objects
 *  that will be dropped by the counter. */
double MINIMUM_SILHOUETTE_SIZE = 0.005;

/* The global SHOW-switcher to showing image. */
bool SHOW = true;

/* The method that save input image into global image variable. */
void saveImage(QImage image) {
    GLOBAL_IMAGE = image;
}

/* The method that returns global image variable. */
QImage getImage() {
    return GLOBAL_IMAGE;
}

/* The method that set the SHOW-switcher in accordance with a
showRadioButton. */
void setShow(bool switcher) {
    SHOW = switcher;
}

/* The method that returns the SHOW-switcher. */
bool getShow() {
    return SHOW;
}

/* The method that shows the image on the program screen (ui.label)
 * @param ui - user interfase inherited from window;
 * @param image - image that need to show on the program screen. */
template <typename T>
void showImage(T ui, QImage image) {
```

```
    if (SHOW) {
        image = image.scaledToWidth(ui->scrollArea->width()-50);
        ui->label->setPixmap(QPixmap::fromImage(image));
    }
}

/* The method that fixing the image.
 * @param ui - user interfase inherited from window;
 * @param image - image that need to fix.
 * @return - fixed image. */
template <typename T>
QImage fixImage(T ui, QImage image) {
    ui->titleLabel->setText("Initializating...");

    int imgHeight = image.height();
    int imgWidth = image.width();

    ui->titleLabel->setText("Clearing image...");

    for (int x = 0; x < imgWidth; x++) {
        if (SHOW) {
            std::cout << (  (x*100)/imgWidth  ) << std::endl;
        }
        for (int y = 0; y < imgHeight; y++) {

            QImage tempImage = image;

            //set yellow cursor
            if (SHOW) {
                tempImage.setPixel(x, y, qRgb(255, 255, 0));
                showImage(ui, tempImage);
            }

            //small pause
            delay(ui);

            //clearing current pixel
            QColor curPixCol = image.pixel(x, y);
            if (curPixCol.lightness() >= LIGHTNESS) {
                image.setPixel(x, y, qRgb(255, 255, 255));
            } else {
                image.setPixel(x, y, qRgb(0, 0, 0));
            }

            if (SHOW) {
                //repaint image
                showImage(ui, tempImage);
            }

            // if (last pixel) { save image }
            if ((x == imgWidth-1) && (y == imgHeight-1)) {
```

```
                GLOBAL_IMAGE = tempImage;
            }
        }
    }
    ui->titleLabel->setText("Image cleared.");
    ui->findSilhButton->setEnabled(true);
    return GLOBAL_IMAGE;
}

/* The method which finds the number of silhouettes of the objects.
 *  Calls the getCountOfObj() method which doing the dirty job.
 * @param ui - user interfase inherited from window;
 * @param image - image that need to analyze. */
template <typename T>
void findSilhouettes(T ui, QImage image) {
    ui->titleLabel->setText("Searching for silhouettes...");
    int counter = getCountOfObj(ui, image);
    ui->titleLabel->setText(" ");

    QString finalStr = "Here I found " + QString::number(counter) + "
objects.";

    QMessageBox msgBox;
    msgBox.setText("Done!");
    msgBox.setInformativeText(finalStr);
    msgBox.exec();
}

/* The main method of finding the number of silhouettes of the
objects.
 *  Calling recursive method fillArea() to mark all the pixels of
image.
 * @param ui - user interfase inherited from window;
 * @param image - image that need to analyze.
 * @return counter - count of the objects. */
template <typename T>
int getCountOfObj(T ui, QImage image) {
    int counter = 0;

    int imgHeight = image.height();
    int imgWidth = image.width();

    /* Calculating the MINIMUM_SILHOUETTE_SIZE given the image size */
    MINIMUM_SILHOUETTE_SIZE = (imgHeight*imgWidth) *
MINIMUM_SILHOUETTE_SIZE;

    for (int y = 0; y < imgHeight; y++) {
        for (int x = 0; x < imgWidth; x++) {
            QColor curPixCol = image.pixel(x, y);
            //set yellow cursor
            if (SHOW) {
```

```
                QImage tempImage = image;
                tempImage.setPixel(x, y, qRgb(255, 0, 255));
                showImage(ui, tempImage);
                delay(ui);
            }
            if (curPixCol.black() > 230) {
                int silhouetteSize = 0;
                fillArea(ui, &image, x, y, counter+1, silhouetteSize);
                if (silhouetteSize >= MINIMUM_SILHOUETTE_SIZE) {
                    counter++;
                    //ui->titleLabel-
>setText(QString::number(counter));
                    GLOBAL_IMAGE = image;
                }
            }
        }
    }
    return counter;
}

/* The recursive method which marks all the pixels of image.
 *  Analyze the pixel under "cursor" with (x, y) coords.
 * @param ui - user interfase inherited from window;
 * @param image - image that need to analyze.
 * @param x - X-coord of "cursor".
 * @param y - Y-coord of "cursor".
 * @param &silhouetteSize - address to the silhouetteSize variable
which contains the number of pixels for the minimum size of a
silhouette. */
template <typename T>
void fillArea(T ui, QImage* image, int x, int y, int marker, int
&silhouetteSize) {

    QColor curPixCol = image->pixel(x, y);
    int imgHeight = image->height();
    int imgWidth = image->width();

    //set yellow cursor
    if (SHOW) {
        QImage tempImage = *image;
        tempImage.setPixel(x, y, qRgb(255, 255, 0));
        showImage(ui, tempImage);
        delay(ui);
    }

    if ( (x > 0) &&
         (x < imgWidth-1) &&
         (y > 0) &&
         (y < imgHeight-1) &&
         (curPixCol.black() > 240) )
    {
```

```
        if (curPixCol.black() > 240) {
            image->setPixel(x, y, qRgb(220, 0, 0));

            showImage(ui, *image);
            delay(ui);
            silhouetteSize++;

            fillArea(ui, image, x+1, y, marker, silhouetteSize); //
right
            fillArea(ui, image, x-1, y, marker, silhouetteSize); //
left
            fillArea(ui, image, x, y+1, marker, silhouetteSize); //
down
            fillArea(ui, image, x, y-1, marker, silhouetteSize); // up
        } else {
            return;
        }
    }
}
#endif // LOGIC
```

Додаток Е. Модуль розмітки віконного додатку.

```
//window.ui
// <?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>window</class>
 <widget class="QMainWindow" name="window">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>593</width>
    <height>513</height>
   </rect>
  </property>
  <property name="maximumSize">
   <size>
    <width>1280</width>
    <height>720</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Silhouettes Counter v1.0</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0" colspan="3">
     <widget class="QLabel" name="titleLabel">
      <property name="text">
       <string>Please choose the file.</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignCenter</set>
      </property>
     </widget>
    </item>
    <item row="3" column="2" rowspan="2">
     <widget class="QPushButton" name="findSilhButton">
      <property name="sizePolicy">
       <sizepolicy hsizetype="Minimum" vsizetype="Fixed">
        <horstretch>0</horstretch>
        <verstretch>0</verstretch>
       </sizepolicy>
      </property>
      <property name="minimumSize">
       <size>
        <width>0</width>
        <height>48</height>
       </size>
      </property>
      <property name="text">
```

```
   <string>Find silhouettes</string>
  </property>
  <property name="checkable">
   <bool>false</bool>
  </property>
  <property name="flat">
   <bool>false</bool>
  </property>
 </widget>
</item>
<item row="3" column="1">
 <widget class="QPushButton" name="showButton">
  <property name="text">
   <string>Show</string>
  </property>
 </widget>
</item>
<item row="3" column="0">
 <widget class="QRadioButton" name="showRadioButton">
  <property name="text">
   <string>Show image</string>
  </property>
  <property name="checked">
   <bool>true</bool>
  </property>
  <property name="autoRepeat">
   <bool>true</bool>
  </property>
 </widget>
</item>
<item row="1" column="0" colspan="3">
 <widget class="QScrollArea" name="scrollArea">
  <property name="frameShape">
   <enum>QFrame::Box</enum>
  </property>
  <property name="widgetResizable">
   <bool>true</bool>
  </property>
  <property name="alignment">
   <set>Qt::AlignCenter</set>
  </property>
  <widget class="QWidget" name="scrollAreaWidgetContents">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>571</width>
     <height>347</height>
    </rect>
   </property>
   <layout class="QFormLayout" name="formLayout_2">
```

```
      <item row="0" column="0">
       <widget class="QLabel" name="label">
        <property name="text">
         <string/>
        </property>
       </widget>
      </item>
     </layout>
    </widget>
   </widget>
  </item>
  <item row="2" column="0">
   <widget class="QLabel" name="delayLabel">
    <property name="maximumSize">
     <size>
      <width>70</width>
      <height>16777215</height>
     </size>
    </property>
    <property name="text">
     <string>Delay (ms)</string>
    </property>
   </widget>
  </item>
  <item row="2" column="2">
   <widget class="QPushButton" name="openFileButton">
    <property name="text">
     <string>Open File...</string>
    </property>
   </widget>
  </item>
  <item row="4" column="0" colspan="2">
   <widget class="QPushButton" name="fixImageButton">
    <property name="text">
     <string>Fix image</string>
    </property>
   </widget>
  </item>
  <item row="2" column="1">
   <widget class="QSpinBox" name="delayBox">
    <property name="minimum">
     <number>1</number>
    </property>
    <property name="maximum">
     <number>1000</number>
    </property>
   </widget>
  </item>
 </layout>
</widget>
<widget class="QMenuBar" name="menuBar">
```

```
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>593</width>
     <height>20</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```