МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

“ІНТЕЛЕКТУАЛЬНА СИСТЕМА ФОРМУВАННЯ ЗОБРАЖЕНЬ”

Рівень “бакалавр”

Галузь знань 12 “Інформаційні технології”

Спеціальність 121 “Інженерія програмного забезпечення”

Виконав здобувач 4 курсу

Вогністий Олег Валентинович

Керівник: к.пед.н., доцент

Логінова Наталія Іванівна

Рецензент: к.т.н., доцент

Щербина Юрій Володимирович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ”
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань 12 Інформаційні технології
Спеціальність 121 “Інженерія програмного забезпечення”
Назва освітньої програми “ Інженерія програмного забезпечення ”

ЗАТВЕРДЖУЮ

Завідувач кафедри інформаційних технологій
доцент Наталія Логінова _____
“____”_____ 20___ року

ЗАВДАННЯ
на кваліфікаційну (бакалаврську) роботу
здобувачу Вогністому Олегу Валентиновичу

1. Тема роботи: “Інтелектуальна система формування зображень”
   Керівник роботи: к.пед.н., доцент Логінова Наталія Іванівна
   затверджені наказом НУ “ОЮА” № 809-06 від “02” квітня 2024 року
2. Дата видачі завдання “15”вересня 2023 року
3. Строк подання здобувачем роботи “20” травня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми та складання плану роботи	20.10.2023	
2	Підготовка першого розділу роботи	30.11.2023	
3	Підготовка другого розділу роботи	25.12.2023	
4	Підготовка третього розділу роботи	12.02.2024	
5	Підготовка вступу та висновків	19.03.2024	
6	Доопрацювання роботи з урахуванням зауважень	20.04.2024	
7	Подача електронного варіанту роботи для перевірки на плагіат	19.05.2024	
8	Допуск роботи до захисту завідуючим кафедри		
9	Захист роботи		

Здобувач _____
(підпис) (ім’я та прізвище)

Керівник роботи _____
(підпис) (ім’я та прізвище)

АНОТАЦІЯ

Вогністий О. В. «Інтелектуальна система формування зображень». – Кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення», освітньою програмою «Інженерія програмного забезпечення».

Кваліфікаційна робота викладена на 46 сторінках основного тексту, містить 3 розділи, 28 рисунків та 10 використаних джерел.

Метою роботи є розроблення інтелектуальної системи формування зображення, яка .

Об’єкт дослідження – інтелектуальна система формування зображень.

Предмет дослідження – засоби розробки інтелектуальної системи оброки зображень.

У кваліфікаційної роботі розроблена інтелектуальна система формування зображень, засобами мови програмування Python та бібліотек PyQt5 і OpenCV.

Перший розділ роботи присвячений аналізу предметної області дослідження та обрано засоби розробки інтелектуальної системи формування зображень. Визначені функціональні та нефункціональні вимоги. В другому розділі виконано проєктування інтелектуальної системи формування зображень. Розглянута архітектура інформаційної системи. Побудовано алгоритм розпізнавання контурів об’єктів на зображенні. Проаналізовані існуючі способи виділення контурів на зображенні. Третій розділ присвячений процесу розробки та тестування програмного продукту, який був створений для реалізації системи формування зображень. Створена система надає можливість завантажувати зображення, автоматично визначати контури предметів на зображенні і створювати викрійку обраного об'єкта за введеними параметрами.

Ключові слова: зображення, контур, інтелектуальна система, розпізнавання образів, Python, PyQt5, OpenCV

ABSTRACT

Vognistii O. V. "Intellectual image formation system". – Bachelor's qualification work on specialty 121 "Software engineering", educational program "Software engineering".

The qualification work is laid out on 46 pages of the main text, contains 3 chapters, 28 figures and 10 used sources.

The purpose of the work is to develop an intelligent system of image formation, which

The object of research is an intelligent system of image formation.

The subject of the study is the means of developing an intellectual system of image rendering.

In the qualification work, an intelligent image formation system was developed using the Python programming language and the PyQt5 and OpenCV libraries.

The first section of the work is devoted to the analysis of the subject area of research and the means of developing an intelligent system of image formation are selected. Functional and non-functional requirements are defined. In the second section, the design of an intelligent system of image formation is performed. The considered architecture of the information system. An algorithm for recognizing contours of objects in the image was built. The existing methods of selecting contours on the image are analyzed. The third section is devoted to the process of development and testing of the software product, which was created to implement the imaging system. The created system provides an opportunity to download images, automatically determine the contours of objects in the image and create a pattern of the selected object according to the entered parameters.

Keywords: image, contour, intelligent system, pattern recognition, Python, PyQt5, OpenCV

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

LBP – Local Binary Patterns

GUI – Graphical User Interface

PIL – Python Imaging Library

ВСТУП

У сучасному світі неможливо уявити будь-яку сферу життєдіяльності людини без цифрових зображень. Цифрові зображення використовуються в маркетингу, рекламі, дизайні, мистецтві, освіті та інших галузях. З розвитком технологій обробки зображень та штучного інтелекту, цифрові зображення стали джерелами передачі інформації та виразу ідей. Для того щоб цифрові зображення відповідали певним завданням, їх потрібно розпізнавати та оброблювати.

Розпізнавання зображення полягає у присвоєнні йому певного класу на основі виділених ознак. Для ефективного вирішення цієї задачі використовується цифрова обробка зображень, така як покращення якості зображення, визначення параметрів та розпізнавання самого зображення. Значною складовою успішного вирішення цих завдань є використання різних графічних бібліотек та мов програмування, які за допомогою спеціальних алгоритмів та програм виявляють об'єкти на вхідних даних, ідентифікують їх та класифікують для подальшого розпізнавання.

Для якісного розпізнавання зображень потрібна цифрова обробки зображень, як-то: поліпшення якості зображення, визначення параметрів зображення, змінення розмірів зображення та ін. Тому тема кваліфікаційної роботи є актуальною.

Метою роботи є розроблення інтелектуальної системи формування зображення.

Відповідно до поставленої мети **завданнями** кваліфікаційної роботи є:

1. Аналіз предметної області.
2. Формування вимог та вибір засобів розробки системи.
3. Проєктування інтелектуальної системи.
4. Розробка інтелектуальної системи обробки зображень.
5. Тестування та оцінка якості створеної системи.

Об’єктом дослідження є інтелектуальна система формування зображень.

Предметом дослідження є засоби розробки інтелектуальної системи оброки зображень.

В кваліфікаційній роботі використовувалися загальнонаукові та спеціальні методи дослідження, а саме: аналіз, моделювання, порівняння, класифікації тощо.

Практична значущість результатів дослідження полягає в можливості спростити процес обробки та редагування зображень, а саме система надає можливість завантажувати зображення, автоматично визначати контури предметів на зображенні і створювати викрійку обраного об'єкта за введеними параметрами.

Кваліфікаційна робота викладена на 46 сторінках основного тексту, містить 3 розділи, 28 рисунків та 10 використаних джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Аналіз особливостей розпізнавання цифрових зображень

Цифрові зображення створюються багатьма пристроями та використовуються для різних цілій, включаючи розважальні, медичні, військові, наукові тощо.

При роботі з цифровими зображеннями існує три напряму: аналіз, синтез та обробка.

Аналіз зображень включає в себе виявлення певних характеристик зображення, таких як контури, текстури, кольори тощо.

Синтез зображень передбачає створення нових зображень на основі вже наявних даних.

Обробка зображень включає в себе різноманітні операції, такі як зменшення шуму, підвищення роздільної здатності, зміна кольорів тощо.

Аналіз зображень відноситься до питань розпізнавання образів.

Розпізнавання зображень – інформаційна технологія, що створена для отримання та розуміння зображень, їх перетворення на цифрову форму для подальшої обробки та аналізу. У цю область залучено інтелектуальний аналіз даних, розпізнавання образів тощо. Наукові досягнення в розпізнаванні графічних зображень призвели до того, що комп'ютери стали здатні імітувати людський зір.

Сьогодні розпізнавання зображень – одне з основних і широко використовуваних завдань комп'ютерного зору. Розпізнавання образів на зображеннях і вилучення ознак також є важливою частиною інших, більш складних методів комп'ютерного зору, таких як виявлення об'єктів і сегментація зображення. Досить велика та універсальна функціональність розпізнавання може забезпечити цілу низку корисних функцій, таких як: модерація контенту, покращений візуальний пошук та інтерактивний маркетинг.

Завданнями розпізнавання образів є класифікація вихідних даних за певними категоріями, шляхом виявлення ключових ознак, що визначають характеристики цих даних, серед обсягу неважливої інформації.

З практичної точки зору вирішення проблеми розпізнавання відкриває можливості автоматизації процесів, які дотепер вважалися виключно людськими. Зважаючи на те, що зорове сприйняття є одним з ключових джерел цифрових даних, серед основних задач, які розв'язуються у сучасних інформаційних системах, можна виділити розпізнавання та ідентифікацію об'єктів у системах технічного зору, розпізнавання букв, цифр, слів або фраз, ідентифікацію за фотографією, виявлення цілей тощо [1].

Специфіка алгоритмів автоматичного аналізу цифрових зображень визначаються низкою характеристик. По-перше, в інформаційних системах потрібно вирішувати конкретні та вузькі задачі проблемно-орієнтованої інтерпретації зображення, такі як сегментація областей з подальшим аналізом їх розташування та властивостей, або виявлення та ідентифікація семантичних об'єктів на зображенні. У цьому контексті рішення задачі аналізу зображення базується на яскравій та геометричній моделі об'єктів, що потребують виділення. Другим аспектом, який потрібно враховувати при розробці алгоритмів обробки зображень в системах інформаційного забезпечення, є спеціальні вимоги, пов'язані з унікальними характеристиками системи управління, такими як швидкодія, вірогідність виявлення та точність вимірювання різних характеристик об'єктів.

Отже, основою обробки і аналізу цифрових зображень є необхідність формалізації моделей зображень (об'єктів) та розроблення методів аналізу зображень на основі цих формалізованих моделей. У сфері аналізу зображень зусилля науковців та практиків спрямовані на створення універсальних та предметно адекватних моделей для різних застосувань, що призвело до розвитку різноманітних методів сучасного комп'ютерного зору.

Класичні методи розпізнавання образів [2]:

– статистичний;

– синтаксичний;

– відповідність шаблону.

Статистичний метод розпізнавання образів використовує визначення подібності об'єктів з урахуванням їх характеристик і статистичних даних. Він включає збір статистичних даних про об'єкти, аналіз інформації та застосування математичних алгоритмів для визначення відповідності об'єктів певним критеріям. Цей метод дозволяє автоматизувати процес класифікації та розпізнавання об'єктів на основі їх ознак.

Поширеним статистичним методом є кореляційний аналіз цифрових зображень. На основі кореляційного підходу більш успішним є метод матриць взаємозв’язку. Дані матриці показують частоту пар різних градацій сірого кольору, що присутні в зображені, і визначаються шляхом кореляційного аналізу пікселів зображення. Якщо піксель відповідає вибраній градації, то він враховується як одиничне значення, якщо не відповідає, то враховується як нульове значення. Якщо зображення кольорове, то даний підхід застосовують до аналізу кожного базового кольору. Іншим статистичним методом є аналогічний аналіз в спектральній області із застосуванням довимірного дискретного перетворення Фур’є, дискретних експоненціальних функцій, косинусних, Уолша-Адамара, Харра тощо [3].

Синтаксичний метод розпізнавання образів – це метод, який використовує комп'ютерні алгоритми для аналізу та розуміння структури природної мови чи інших форм цифрової інформації. Для роботи даного методу комп'ютер використовує спеціальні алгоритми синтаксичного аналізу, які дозволяють йому побудувати семантичну структуру. Цей метод широко застосовується у обробці природної мови, автоматичному перекладі текстів, соціальній та інших галузях, де необхідно розуміння та аналіз структури інформації.

Класичні методи розпізнавання образів є наборами алгоритмів, які використовують для аналізу та класифікації цифрових зображень [4]. До них відносяться:

Фільтр Собеля – алгоритм використовується для виділення меж на зображенні. Працює такої алгоритм, шляхом застосування матриці згортки до кожного пікселя зображення та дозволяє обчислювати градієнт яскравості. Так визначається, де на зображенні знаходяться переходи від світлих до темних областей і навпаки.

Image Moments – це алгоритм пов'язаний із статистичними характеристиками зображення, такими як центр маси, площа, орієнтація тощо. Алгоритм застосовує для аналізу форми об'єктів на зображенні.

Ці методи широко застосовуються, наприклад, для виділення ключових рис особи у системах безпеки, ідентифікації та авторизації, а також у фото- та відеоредакторах.

Таки методи для розпізнавання образів використовує бібліотека OpenCV, яка застосовується для комп’ютерного зору та аналізу зображень.

Метод зіставлення шаблонів у розпізнаванні образів використовується для зіставлення шаблонів, попередньо створених зображень або зразків, із новими зображеннями або даними. Цей метод заснований на порівнянні характеристик шаблону з характеристиками інших зображень та визначення ступеня їхньої подібності. При цьому відбувається аналіз форми, текстури, кольору та інших параметрів зображення.

До таких методів відносяться *локальні бінарні шаблони* (Local Binary Patterns, LBP) – це метод, який використовується для текстурного аналізу зображень. В даному методі для кожного пікселя зображення визначається його локальне оточення, і на основі значень яскравості в цьому просторі створюється бінарний шаблон. Потім бінарні шаблони використовують для опису текстурних характеристик на зображенні. Метод надає можливість описати текстури, не залежно від яскравості зображення та може бути ефективно використаний для аналізу текстур у різних умовах освітлення.

LBP названо так через спосіб, яким цей метод описує текстурні особливості на зображеннях. Кожен піксель на зображенні розглядається як локальний центр

для навколишньої області, і для кожного пікселя обчислюється бінарне значення на основі його порівняння з навколишніми пікселями.

Прикладами інформаційних продуктів, створених на основі локальних бінарних шаблонів є платформа для розпізнавання та аналізу осіб Face++ компанії Megvii, яка дозволяє виявляти та порівнювати унікальні текстурні шаблони на обличчі та використовується в системах ідентифікації та верифікації осіб.

Каскади Хаара є набором класифікаторів, які навчаються на великій кількості позитивних і негативних прикладів вихідного об'єкта. Потім вони застосовуються до зображення у вигляді "каскаду", де простіші класифікатори розглядаються першими для швидкого відсіювання великої кількості негативних областей, а більш складні класифікатори використовуються пізніше для більш точної перевірки. На етапі навчання алгоритм тренується на великому наборі тестових зображень, які містять як позитивні (зображення з об'єктами, які необхідно виявити), так і негативні (зображення без об'єктів інтересу) приклади. До кожного об'єкта на позитивних зображеннях витягуються характерні ознаки Хаара. Вони є яскравими відмінностями в різних областях зображення. Після навчання створюється каскад класифікаторів. Каскад складається з кількох етапів (класифікаторів), кожен із яких є бінарним класифікатором (так/ні). Етапи каскаду характеризуються пороговим значенням, у якому об'єкт вважається виявленим чи відхиленим. Процес виявлення починається із застосування першого етапу каскаду до зображення. Якщо етап не пройшов, зображення відхиляється. Якщо пройшов, то зображення передається на наступний етап, і так далі. Коли всі етапи каскаду успішно пройдено, об'єкт вважається виявленим.

Каскади Хаара надають високу швидкість виявлення та можуть використовуватись для різних типів об'єктів: осіб, очей, автомобілів та інших. Цей метод став популярним у галузі комп'ютерного зору завдяки своїй ефективності та здатності працювати в реальному часі.

Бібліотека OpenCV, наприклад, надає реалізації каскадів Хаара для виявлення таких об'єктів, як посмішки, автомобілі, велосипеди на зображеннях та відеопотоці. Це одна з найпопулярніших бібліотек для реалізації систем

комп'ютерного зору. Інший аналогічний проект – Dlib. Він включає реалізації такого типу класифікаторів для завдань розпізнавання осіб.

Для класичних методів розпізнавання образів є певні переваги та недоліки.

Перевага класичних алгоритмів розпізнавання образів у тому, що вони не потрібують даних для навчання, обробляють зображення на високої швидкості й можуть працювати у режимі реального часу. Каскади Хаара та алгоритми з урахуванням характеристичних ознак ефективні без навчання на великих наборах даних, що робить їх корисними в ситуаціях, де навчання неможливе або скрутне. Крім того, їх легко інтерпретувати, тому що вони ґрунтуються на чітких математичних принципах.

До недоліків можна віднести те, що налаштування параметрів класичних методів може бути трудомістким завданням та вимагає експертних знань для досягнення потрібного результатів. Класичні методи чутливі до змін за умов зйомки. При складних умовах їх продуктивність може знижуватися.

Для визначення методів розпізнавання образів, які будуть застосовуватися для інтелектуальної системи формування зображень, потрібно визначити вимоги.

1.2 Аналіз вимог до інформаційної системи

Для створення якісної інформаційної системи потрібно визначити вимоги, які є основою для розробки, тестування та подальшого використання інформаційного продукту. Існують функціональні та нефункціональні вимоги [6].

Функціональні вимоги – це визначення того, що система повинна робити для задоволення потреб або очікувань користувача. Вони відрізняються від нефункціональних вимог, які визначають, як система повинна працювати.

Система, яка створюється, повинна спростити процес обробки та редагування зображень. Вона повинна мати можливість завантажувати зображення, автоматично визначати контури предметів на зображенні та створювати викрійку обраного об'єкта за введеними параметрами.

Для розробки інформаційної системи обробки зображень потрібно врахувати декілька основних аспектів:

1. Потрібно визначити, які методи і алгоритми будуть використовуватися для обробки зображень.

2. Інформаційна система може використовувати навчальні моделі для автоматичного визначення контурів на зображеннях.

3. Інтеграція з іншими системами – інформаційна система для виділення контурів цифрових зображень може бути інтегрована з іншими системами, наприклад, системами аналізу зображень або системами візуалізації даних, що допоможе в удосконаленні процесу обробки зображень і забезпечить більш точні результати.

4. Інформаційна система повинна працювати швидко і ефективно, особливо у випадку обробки великої кількості зображень. Тому важливо розробити оптимізовані алгоритми та використовувати потужне обладнання для забезпечення продуктивності системи.

Загалом, розробка інформаційної системи для виділення контурів цифрових зображень вимагає комплексного підходу для досягнення оптимальних результатів.

Інтелектуальна система формування зображень повинна бути стабільною та надійною у використанні, мати інтуїтивно-зрозумілий та зручний інтерфейс для забезпечення взаємодії користувачів з системою.

Функціональні вимоги визначають можливість системи виконувати поставлені завдання:

1. У користувача повинна бути можливість завантажувати зображення з локального пристрою у таких основних форматах – .png, .jpg, .jpeg.

2. Повинні визначатись контури об’єкта зображеного на фото лише по краю всього предмету не чіпляючи середину самого предмета.

3. Користувач повинен ввести параметр, тобто число яке переводиться потім у сантиметри, для того щоб збільшити цей контур на задану кількість сантиметрів з усіх сторін, залишаючи при цьому завантажене зображення.

4. При введені параметру, користувач повинен побачити результат обробки фото з доданим контуром, який можна масштабувати за необхідними параметрами.

5. Останнім кроком є збереження готового результату.

Графічно функціональні вимоги подано у вигляді діаграми варіантів використання (рис. 1.1), яка показує стосунки між користувачем і прецедентами.

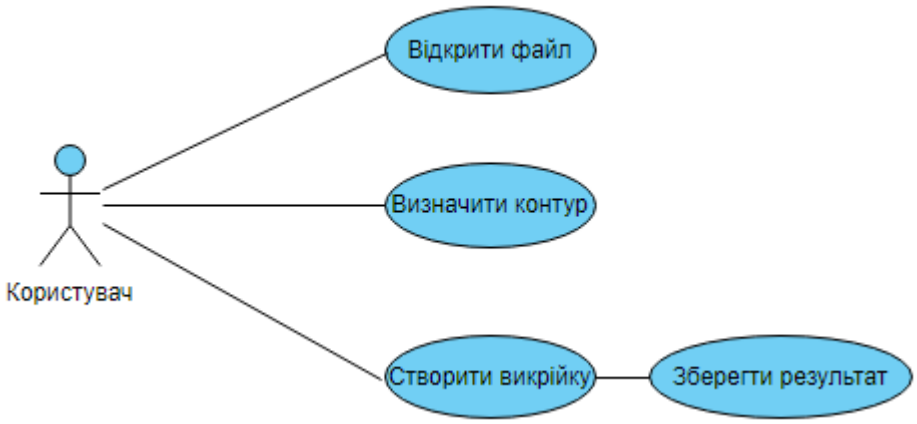


Рисунок 1.1 – Діаграма варіантів використання

Для системи характерний один користувач, який взаємодії з варіантами використання.

Нефункціональні вимоги системи:

– обробка зображень повинна виконуватися на будь-якої операційної системи;

– мати зручний та простий інтерфейс;

– відповідати вимогам продуктивності роботи.

1.3 Вибір засобів розробки

Для розробки інтелектуальної системи формування зображень було обрано мову програмування Python, яка використовується для програмування графічного інтерфейсу (Graphical User Interface, GUI). Це інтерактивне середовище, яке бачить користувач і взаємодіє з ним після відкриття програми. GUI відображає об’єкти, які передають інформацію, і представляє дії, які може виконати користувач.

Графічний інтерфейс має вирішальне значення для підвищення репутації вашої платформи та кількості користувачів, а поєднання всіх цих елементів відіграє важливу роль у взаємодії з вашим додатком або веб-сайтом.

Для створення GUI багато розробників звертаються до Python, який має багато різних фреймворків.

Python – це інтерактивна мова програмування, яка дозволяє створити GUI. Python має широкий спектр опцій для фреймворків графічного інтерфейсу користувача, включаючи міжплатформні фреймворки та фреймворки для певних платформ.

До Python легко підключити бібліотеки, які розширюють можливі рішення застосунків.

Для розробки графічного інтерфейсу будемо використовувати бібліотеку PyQt5, оскільки вона містить необхідні інструменти для створення десктопних додатків, та дозволяє створювати інтуїтивно зрозумілі та функціональні інтерфейси користувача.

PyQt5, розроблений Riverbank Computing, є одним із найпопулярніших фреймворків Python для GUI. Пакет PyQt побудовано навколо фреймворку Qt, який є міжплатформенним фреймворком, який використовується для створення різних програм на різних платформах.

PyQt5 є повністю кросплатформним, тобто розробники можуть використовувати його для створення програм на різноманітних платформах, таких як Mac, Windows, Linux, iOS та Android. Він пропонує модулі QtGUI і QtDesigner, які надають візуальні елементи. Також можна вибрати створення елемента за допомогою коду, що дає змогу легко розробляти різномасштабні програми.

Основними перевагами PyQt5 є:

– універсальність кодування;

– наявність різних компонент інтерфейсу користувача;

– наявність навчальних ресурсів;

– різноманітність API рідної платформи для мереж, керування базами даних тощо [7].

Для аналізу та обробки зображень ефективна бібліотека Python Imaging Library (PIL) та бібліотека OpenCV.

OpenCV – це бібліотека з відкритим кодом для комп'ютерного зору, машинного навчання та обробки зображень [8]. Саме завдяки їй є можливість визначати контури об'єктів на цифрових зображеннях.

Бібліотека OpenCV має модульну структуру (рис. 1.2) [9].

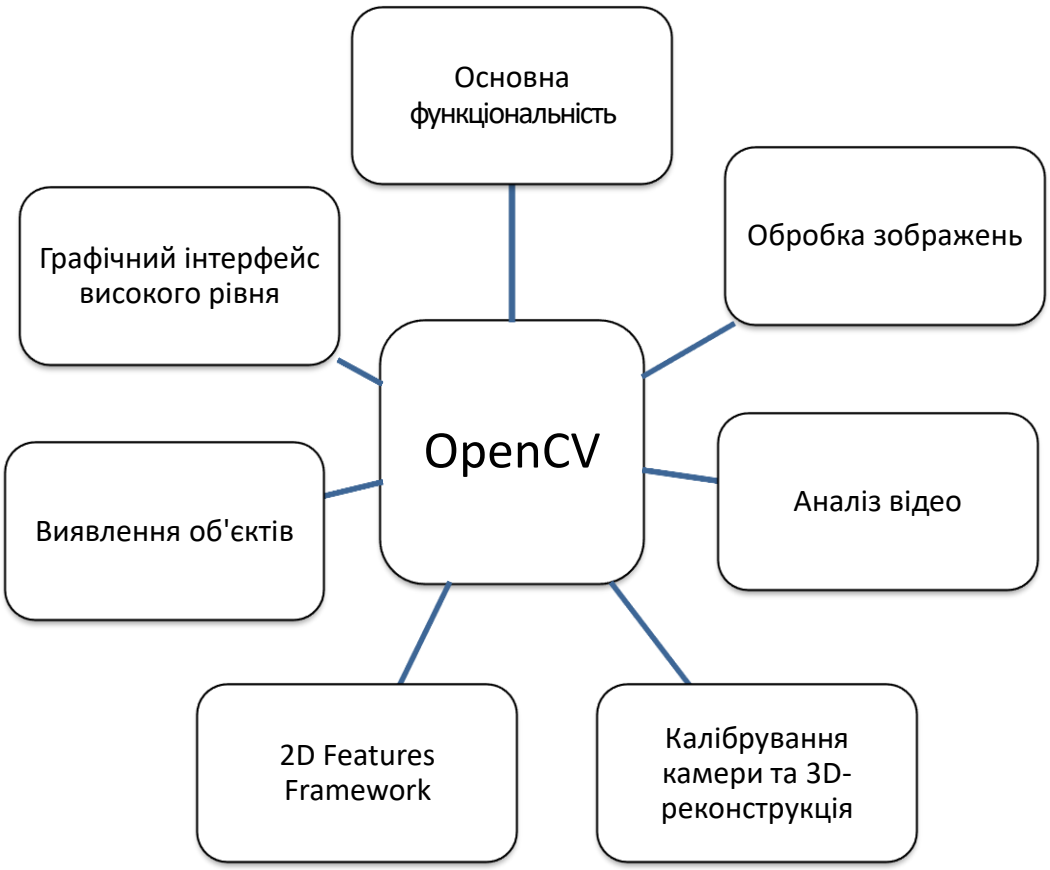


Рисунок 1.2 – Модулі бібліотеки OpenCV

Основна функціональність – модуль, що визначає базові структури даних, включаючи щільний багатовимірний масив Mat і базові функції, які використовуються всіма іншими модулями.

Обробка зображень– модуль обробки зображень, який включає лінійну та нелінійну фільтрацію зображень, геометричні перетворення зображення, перетворення простору кольорів, гістограми тощо.

Аналіз відео – аналіз відео (включає алгоритми оцінки руху, віднімання фону та відстеження об'єктів).

Калібрування камери та 3D-реконструкція – модуль з базовими алгоритмами геометрії кількох ракурсів, калібрування однієї та стереокамери, оцінка пози об’єкта, алгоритми стереовідповідності та елементи 3D-реконструкції.

2D Features Framework – детектори, дескриптори та відповідники дескрипторів.

Виявлення об'єктів – модуль виявлення об'єктів і примірників попередньо визначених класів (наприклад, обличчя, очі, люди, машини і так далі).

Графічний інтерфейс високого рівня – модуль для інтерфейсу користувача.

Video I/O – інтерфейс для захоплення відео та відеокодеків.

Бібліотеку OpenCV використовують для мови програмування Python через його легкий синтаксис та великий набір вбудованих модулів. Також бібліотека OpenCV легко інтегрується з іншими модулями Python, такими як: NumPy, matplotlib, tensorflow та ін.

1.4 Висновки до першого розділу

В першому розділі кваліфікаційної роботи проаналізована предметна області дослідження та обрано засоби розробки інтелектуальної системи формування зображень.

Для інтелектуальної системи формування зображень визначено функціональні та нефункціональні вимоги. Побудована діаграма варіантів використання.

Для розробки інтелектуальної системи формування зображень було обрано мову програмування Python, бібліотекі PyQt5 та OpenCV.

2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ФОРМУВАННЯ ЗОБРАЖЕНЬ

2.1 Архітектура інформаційної системи

Для реалізації інтелектуальної системи обробки зображень створено алгоритм розпізнавання контурів об’єктів на зображені (рис. 2.1).

Рисунок 2.1 – Алгоритм розпізнавання контурів об’єктів на зображені

Завантаження зображення з графічного файлу у форматі png або jpg.

Обробка зображення: перевіряємо, що зображення завантажилось правильно та не виходить за рамки робочої області.

Розпізнавання контурів та створення викрійки здійснюється за допомогою мови програмування Python і бібліотеки OpenCV.

2.2 Обробка зображення та розпізнавання контурів

Після завантаження фото виконується його перетворення та визначення контурів об’єктів за допомогою бібліотеки OpenCV.

Для того щоб утворити потім викрійку потрібно зберегти контур та накласти його поверх зображення враховуючи ті параметри які ввів користувач. Але найважливішим є саме правильне визначення контуру, бо не всі предмети на фото мають рівні чи округлі контури, на складних структурах повинно чітко бути визначити контур по краю для того щоб за створеною викрійкою можна потім було правильно зшити подушку.

Існує декілька способів виділення контурів.

Для знаходження контурів в OpenCV є функція `findContours()`, яка використовується для пошуку контуру у бінарному зображенні. Її синтаксис:

`cv2. findContours(thes, cv2.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)`

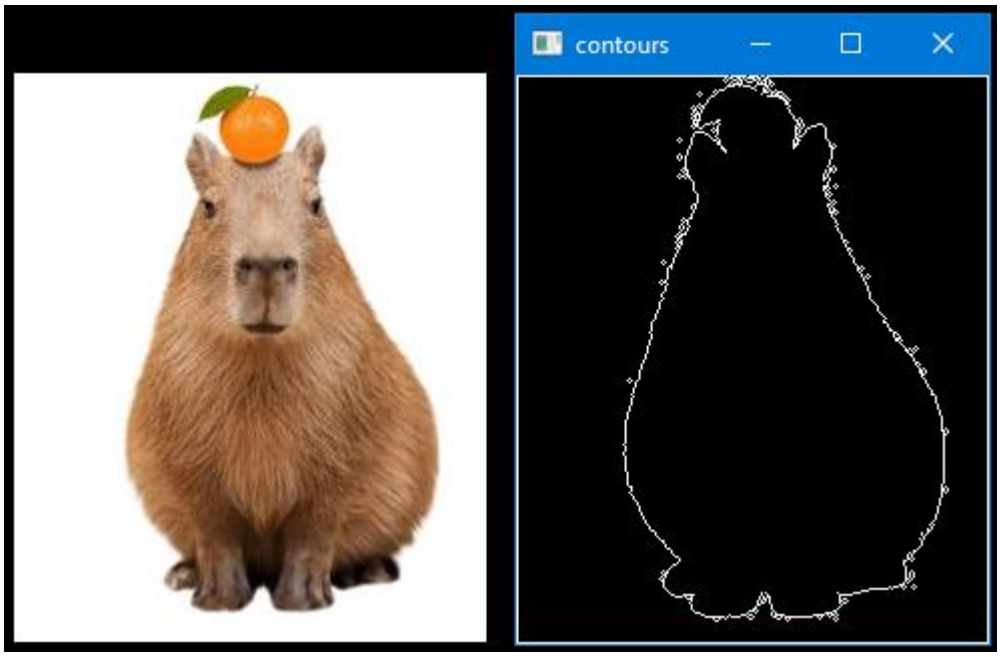
Функція `findContours()` приймає три аргументи: перший аргумент – це вихідне зображення, другий – режим пошуку контурів, а третій – апроксимація контурів.

На рис. 2.2 наведено фрагмент коду, який виділяє контур.

```
import cv2
import numpy as np

my_photo = cv2.imread('ris.jpg')
img_grey = cv2.cvtColor(my_photo,cv2.COLOR_BGR2GRAY)

#задаємо поріг
thresh = 250

#отримуємо картинку
ret,thresh_img = cv2.threshold(img_grey, thresh,  maxval: 255, cv2.THRESH_BINARY)

#надаємо контури
contours, hierarchy = cv2.findContours(thresh_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#сствоюємо порожню картинку
img_contours = np.zeros(my_photo.shape)

#відддзеркалюємо контури
cv2.drawContours(img_contours, contours, -1,  color: (255,255,255),  thickness: 1)

cv2.imshow( winname: 'contours', img_contours) # виводимо зображення у вікно

cv2.waitKey()
cv2.destroyAllWindows()
```

Рисунок 2.2 – Виділення контуру

Результат виконання коду (рис. 2.3).



а) *б)*

Рисунок 2.3 – Виділення фрагмента на зображенні:
а – оригінал; б – контур

Але більш зручно використовувати алгоритм Кенні для виділення контурів (рис. 2.4).

```python
import cv2

# Відкрити зображення
image = cv2.imread('ris.jpg')

# Перетворити в градацію сірого
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Застосування алгоритму Канні
edges = cv2.Canny(gray_image, 160, 320)


# Результат
cv2.imshow( winname: 'Original', image)
cv2.imshow( winname: 'Edges', edges)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Рисунок 2.4 – Виділення контурів за допомогою алгоритму Кенні

Результат роботи алгоритму (рис. 2.5).

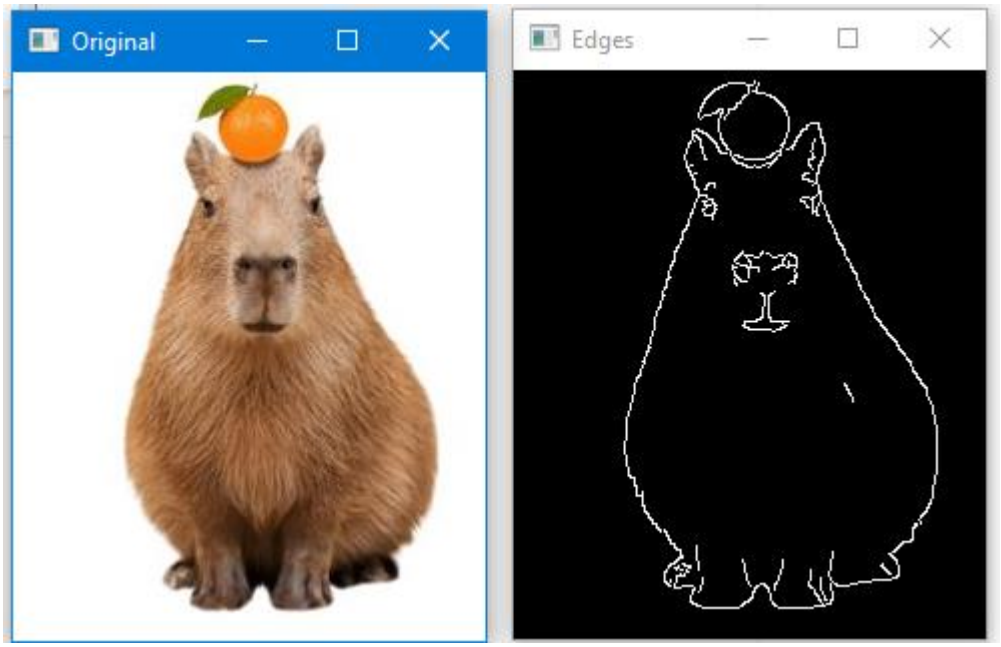


Рисунок 2.5 – Виділення фрагмента на зображенні за допомогою алгоритму Кенні (оригінал і контур)

Кенні вивчив математичну проблему отримання фільтра, оптимального за критеріями виділення, локалізації та мінімізації кількох відгуків одного краю. Це означає, що детектор повинен реагувати на межі, але при цьому ігнорувати помилкові, точно визначати лінію кордону (без її фрагментування) і реагувати на кожну межу один раз, що дозволяє уникнути сприйняття широких смуг зміни яскравості як сукупності меж. Канні ввів поняття Non-Maximum Suppression (пригнічення не-максимумів), яке означає, що пікселями межі оголошуються точки, в яких досягається локальний максимум градієнта у напрямку вектора градієнта.

Алгоритм складається з окремих кроків:

Згладжування – розмиття зображення для видалення шуму.

Пошук градієнтів – межі відзначаються там, де градієнт зображення набуває максимального значення.

Пригнічення не-максимумів – лише локальні максимуми відзначаються як межі.

Подвійна гранична фільтрація – потенційні межі визначаються порогами.

Трасування області неоднозначності – підсумкові межі визначаються шляхом подавлення всіх країв, не пов'язаних із певними (сильними) межами.

Перед застосуванням детектора, потрібно перетворити зображення у відтінки сірого, щоб зменшити обчислювальні витрати.

Алгоритм Кенні містить низку регульованих параметрів, які можуть впливати на тривалість обчислення та ефективність алгоритму.

Розмір гаусівського фільтра, який застосовують на першому етапі як фільтр згладжування, має прямий вплив на результати алгоритму Кенні. Користуючись меншими фільтрами, можна досягти меншого рівня розмиття та легше виявляти дрібні чіткі лінії. Використання більшого фільтра супроводжується вищим рівнем розмиття, оскільки значення певного пікселя розповсюджується по більшій площі зображення. Великі радіуси розмиття корисніші для виявлення великих, більш згладжених контурів.

Параметри алгоритму Кенні дозволяють налаштовувати його для впізнавання контурів з різними характеристиками, залежно від конкретних вимог [10].

2.3 Висновки до другого розділу

В другому розділі кваліфікаційної роботи виконано проєктування інтелектуальної системи формування зображень. Розглянута архітектура інформаційної системи. Побудовано алгоритм розпізнавання контурів об’єктів на зображенні. Проаналізовані існуючі способи виділення контурів на зображенні – за допомогою бібліотеки OpenCV функції `findContours()` та алгоритму Кенні.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТА

Система формування зображень була спроектована із використання модульного підходу, що дозволяє легко підтримувати та розширювати функціонал. На початкової стадії розробки вона містить такі компоненти:

– головне вікно програми, що об’єднує всі елементи інтерфейсу;

– панель інструментів для взаємодії з користувачем, що надає кнопки для відкриття зображень і створення викрійок;

– робочу область для відображення завантажених фото та результатів обробки;

– діалогове вікно куди користувач вводить дані для того щоб була намальована викрійка.

Основний інтерфейс який потрібен користувачу досить невеликий та компактний, та згодом його можна буде удосконалювати (рис. 3.1).

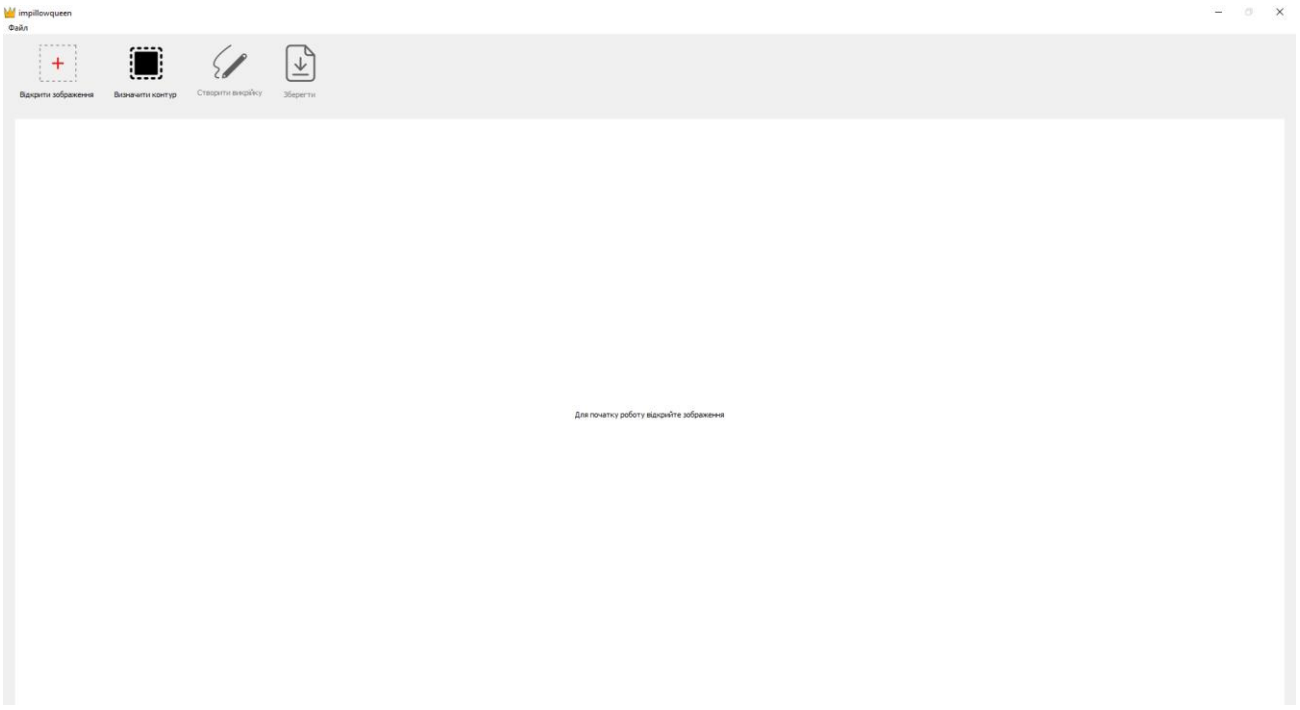


Рисунок 3.1 – Інтерфейс програми

Для того щоб завантажити фото, додано 2 кнопки, одна на панелі інструментів, інша у випадному списку під назвою, можливо для когось так більш зручніше. Також можна буде закрити фото, якщо помилково відкрито було не те

що потрібно, та якщо просто не знадобилось редагувати фото. Там же на панелі інструментів міститись кнопка для визначення контурів та створення викрійки.

Нижче фіксована під панеллю робоча область для відображення завантаженого фото, та результату обробки.

Процес розробки складався з поєднання між собою окремих файлів, що містять усе необхідне для коректної роботи програми та це також забезпечить можливістю з легкістю додавати нові функції та редагувати або створювати новий функціонал. Розглянемо детально файли проекту.

3.1 Файл запуску програми main.py

Файл *main.py* є основним для входу у програму. Він ініціалізує та створює головне вікно (рис. 3.2).

```
1    import sys
2    from PyQt5.QtWidgets import QApplication
3    from main_window import MainWindow
4
5    if __name__ == '__main__':
6        app = QApplication(sys.argv)
7        window = MainWindow()
8        window.show()
9        sys.exit(app.exec_())
```

Рисунок 3.2 – Код файлу main.py

Імпорт модулів: `import sys` – використовуємо для керування середою виконання Python, в даному випадку з цим модулем можемо отримати список аргументів командного рядка та завершити роботу консолі Python, а також виходу з програми у разі виключення. Також імпортуємо модулі з бібліотеки PyQt5 та клас `MainWindow` із файлу `main_window.py`.

Для створення цього головного вікна – створюється екземпляр головного вікна `MainWindow` та викликається метод `show()` для відображення вікна. І вже цикл подій `sys.exit(app.exec_())` запускає програму та забезпечує коректне завершення програми.

3.2 Головний файл програми main_window.py

Файл *main_window.py* містить клас *MainWindow*, в якому визначаються основні компоненти інтерфейсу та функції для взаємодії з користувачем.

Як і у минулому файлі імпортуємо необхідні модулі з бібліотеки PyQt5, OpenCV та NumPy.

У методі *__init__* визначаються основні елементи інтерфейсу, такі як панель інструментів, робоча область та меню. Також тут задаються параметри розміру вікна та його назва (рис. 3.3).

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('impillowqueen')
        self.setWindowIcon(QIcon('icons/iconProject.png'))

        desktop = QApplication.desktop()
        rect = desktop.availableGeometry(desktop.primaryScreen())
        self.setMinimumSize(rect.width(), rect.height())
        self.setMaximumSize(rect.width(), rect.height())

        self.toolbar = Toolbar()
        self.workspace = Workspace()

        self.toolbar.image_selected.connect(self.workspace.set_image)
        self.toolbar.select_button.clicked.connect(self.select_item)
        self.toolbar.create_pattern.connect(self.show_pattern_dialog)
        self.toolbar.save_result.connect(self.save_result)  # Подключаем сиг

        layout = QVBoxLayout()
        layout.addWidget(self.toolbar)
        layout.addWidget(self.workspace)

        central_widget = QWidget()
        central_widget.setLayout(layout)
        self.setCentralWidget(central_widget)

        self.showMaximized()

        self.create_menu()

        self.contours = None  # Добавляем переменную для хранения контуров
```

Рисунок 3.3 – Реалізація функції __init__

Одразу під назвою є невелике меню для відкриття або закриття файлу. Реалізовано цей функціонал в окремому методі *create_menu*. Кнопка закрити

недоступна поки не буде відкрито зображення, для того щоб уникнути помилок у роботі програми (рис. 3.4):

```
def create_menu(self):
    menu_bar = self.menuBar()

    file_menu = menu_bar.addMenu('Файл')

    open_action = QAction('Відкрити', self)
    open_action.triggered.connect(self.select_image)
    file_menu.addAction(open_action)

    self.close_action = QAction('Закрити', self)
    self.close_action.setEnabled(False)
    self.close_action.triggered.connect(self.close_image)
    file_menu.addAction(self.close_action)
```

Рисунок 3.4 – Реалізація методу create_menu

Наступний метод `select_image` – відкриває діалогове вікно для вибору зображення та відображає потім його в робочій області (рис. 3.5):

```
def select_image(self):
    file_dialog = QFileDialog(self)
    if file_dialog.exec_():
        selected_files = file_dialog.selectedFiles()
        if selected_files:
            image_path = selected_files[0]
            self.workspace.set_image(image_path)
            self.close_action.setEnabled(True)
```

Рисунок 3.5 – Реалізація методу select_image

Для того щоб закрити зображення потрібно також було створити метод для цього, після натискання кнопки закрити робоча область очищується від завантаженого зображення (рис. 3.6):

```
1 usage
def close_image(self):
    self.workspace.clear_image()
    self.close_action.setEnabled(False)
```

Рисунок 3.6 – Метод close_image

Далі створимо один з ключових методів даної програми – це `select_item` (рис. 3.7):

```
def select_item(self):
    try:
        pixmap = self.workspace.image_label.pixmap()
        image = pixmap.toImage()

        image = image.convertToFormat(QtGui.QImage.Format_ARGB32)
        width = image.width()
        height = image.height()
        ptr = image.bits()
        ptr.setsize(image.byteCount())
        arr = np.array(ptr).reshape(height, width, 4)

        gray = cv2.cvtColor(arr, cv2.COLOR_BGRA2GRAY)
        edges = cv2.Canny(gray, 100, 200)

        contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        self.contours = contours  # Сохраняем контуры для дальнейшей обработки

        contour_mask = np.zeros( shape: (height, width, 4), dtype=np.uint8)
        cv2.drawContours(contour_mask, contours, -1,  color: (255, 255, 255, 255),  thickness: 2)

        result_image = np.copy(arr)
        result_image[contour_mask[:, :, 0] != 0] = contour_mask[contour_mask[:, :, 0] != 0]

        qImg = QtGui.QImage(result_image.data, width, height,
                            result_image.strides[0],
                            QtGui.QImage.Format_ARGB32)

        self.workspace.image_label.setPixmap(QtGui.QPixmap.fromImage(qImg))
        self.toolbar.enable_pattern_button()  # Сделать кнопку доступной
        self.toolbar.enable_save_button()  # Сделать кнопку сохранения доступной
    except Exception as e:
        print("An error occurred:", e)
```

Рисунок 3.7 – Метод select_item

Цей метод відповідає за вибір об'єкта на зображенні та визначення його контуру. Після визначення контурів, вони зберігаються для подальшого використання в створенні викрійки.

Розберемо покроково, що саме відбувається у цій частині програмного коду:

1. Метод починається з отримання зображення, яке було завантажене в *QLabel (self.workspace.image_label.pixmap());*

2. Зображення конвертується у формат ARGB32, який є зручним для обробки з використанням OpenCV *(image.convertToFormat(QtGui.QImage.Format_ARGB32));*

3. Зображення перетворюється в масив NumPy для подальшої обробки *(np.array(ptr).reshape(height, width, 4));*

4. Масив перетворюється у відтінки сірого для спрощення визначення контурів (*cv2.cvtColor(arr, cv2.COLOR_BGRA2GRAY)).*

5. Використовується алгоритм Canny для визначення країв об'єктів на зображенні (*cv2.Canny(gray, 100, 200)).*

6. Контури знаходяться за допомогою функції *cv2.findContours* та зберігаються у змінній *self.contours* для подальшого використання *(cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)).*

7. Створюється порожня маска та накладаються контури на цю маску *(cv2.drawContours(contour_mask, contours, -1, (0, 0, 0, 255), 2)).*

8. Контури накладаються на оригінальне зображення *(result_image[contour_mask[:, :, 3] != 0] = contour_mask[contour_mask[:, :, 3] != 0]).*

9. Результуюче зображення перетворюється назад в формат QImage *(QtGui.QImage(result_image.data, width, height, result_image.strides[0], QtGui.QImage.Format_ARGB32)).*

10. Нове зображення з контурами встановлюється в *QLabel*.

11. Кнопка для створення викройки та збереження результату стає доступною (*self.toolbar.enable_pattern_button()).*

Після виконання всіх цих дій ми зможемо побачити знайдений контур предмета прямо на фото. І оскільки нам стала доступна тепер кнопка створення викрійки то при натисканні ми побачимо діалогове вікно, у якому треба ввести число на яке потрібно збільшити контур щоб отримати необхідний результат у вигляді готової викрійки.

Відкриття діалогового вікна виконано в окремому методі *show_pattern_dialog* (рис. 3.8):

```
def show_pattern_dialog(self):
    dialog = PatternDialog()
    if dialog.exec_():
        size = dialog.get_size()
        try:
            size_cm = float(size)
            if self.contours is not None and size_cm > 0:
                self.create_pattern(size_cm)  # створюємо викрійку з указаним праметром
            else:
                print("Invalid input. Please enter a valid size (in cm).")
        except ValueError:
            print("Invalid input. Please enter a valid number for the size (in cm).")
```

Рисунок 3.8 – метод show_pattern_dialog

Метод створює об'єкт *PatternDialo*g, який відповідає за відображення діалогового вікна для введення розміру викрійки. Далі метод викликає діалогове вікно для введення значення і очікує підтвердження від користувача, після підтвердження, отримує введене значення розміру. Використовуючи *try:* перетворює введене значення у число з плаваючою комою (см), і перевіряємо чи було створено контур та чи введене значення більше нуля.

І якщо все вірно, то викликається наступний метод – *create_pattern*, а якщо введене значення некоректне, то виводиться повідомлення про помилку.

Перейдемо до останнього методу цього файлу – *create_pattern*, нижче представлено код даного методу (рис. 3.9 та рис. 3.10):

```python
def create_pattern(self, size_cm):
    try:
        if self.workspace.image_label.pixmap() is None or self.contours is None:
            return

        pixmap = self.workspace.image_label.pixmap()
        image = pixmap.toImage()
        image = image.convertToFormat(QtGui.QImage.Format_ARGB32)
        width = image.width()
        height = image.height()
        ptr = image.bits()
        ptr.setsize(image.byteCount())
        arr = np.array(ptr).reshape(height, width, 4)

        # Перетворюємо зображення в пікселі (приблизно 96 DPI)
        dpi = 96
        size_pixels = int(size_cm * dpi)

        # Збільшуємо розмір контура на вказану к-сть пікселів
        enlarged_contours = []
        for contour in self.contours:
            # Отримуємо прямокутник, що обмежує, навколо контуру
            x, y, w, h = cv2.boundingRect(contour)
            # Збільшуємо розмір прямокутника
            enlarged_w = w + size_pixels
            enlarged_h = h + size_pixels
            # Додаємо пікселі до координат контуру
            enlarged_contour = contour.copy() + size_pixels
            enlarged_contours.append(enlarged_contour)
```

Рисунок 3.9 – Метод create_pattern

```python
    # Створюємо маску зі збільшеним контуром
    contour_mask = np.zeros( shape: (height, width, 4), dtype=np.uint8)
    cv2.drawContours(contour_mask, enlarged_contours, -1,  color: (0, 0, 0, 255),  thickness: 2)  # черный цвет

    # Видаляємо старий контур із зображення
    original_mask = np.zeros( shape: (height, width, 4), dtype=np.uint8)
    cv2.drawContours(original_mask, self.contours, -1,  color: (0, 0, 0, 0), -1)

    result_image = np.copy(arr)
    result_image[original_mask[:, :, 3] != 0] = (0, 0, 0, 0)  # Забираємо старий контур

    # Додаємо новий контур до зображення
    result_image[contour_mask[:, :, 3] != 0] = contour_mask[contour_mask[:, :, 3] != 0]

    qImg = QtGui.QImage(result_image.data, width, height, result_image.strides[0], QtGui.QImage.Format_ARGB32)

    self.workspace.image_label.setPixmap(QtGui.QPixmap.fromImage(qImg))
except Exception as e:
    print("An error occurred while creating pattern:", e)
```

Рисунок 3.10 – 2 частина методу create_pattern

Метод починається з перевірки, чи існує зображення та чи були знайдені контури, він отримує зображення, яке було завантажене в QLabel. Вказаний розмір у сантиметрах конвертується в пікселі. Використовується приблизне значення DPI екрану (96) для перетворення. Далі збільшується кожна сторона контура на вказану к-сть пікселів. Створюється порожня маска та накладаються збільшені контури на цю маску. І щоб зберегти зображення без накладеного поверх контуру, той контур, який визначали, прибираємо, а нові контури додаються до оригінального зображення.

Результуюче зображення перетворюється назад в формат QImage. Нове зображення з викройкою встановлюється в QLabel.

Якщо виникає помилка під час створення викройки, метод виводить повідомлення про помилку.

Розглянемо, як працює панель інструментів. Код програми показано на рис. 3.11 та рис. 3.12:

```python
import ...

2 usages
class Toolbar(QToolBar):
    image_selected = pyqtSignal(str)
    create_pattern = pyqtSignal()
    save_result = pyqtSignal()  # Додаємо новий сигнал

    def __init__(self):
        super().__init__()

        self.setMinimumHeight(50)

        icon_size = QSize(55, 55)
        self.setIconSize(icon_size)

        self.open_button = QToolButton()
        self.open_button.setIcon(QIcon('icons/add.png'))
        self.open_button.setText('Відкрити зображення')
        self.open_button.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.open_button.clicked.connect(self.select_image)
        self.addWidget(self.open_button)

        self.select_button = QToolButton()
        self.select_button.setIcon(QIcon('icons/select.png'))
        self.select_button.setText('Визначити контур')
        self.select_button.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.addWidget(self.select_button)

        self.pattern_button = QToolButton()
        self.pattern_button.setIcon(QIcon('icons/pattern.png'))
        self.pattern_button.setText('Створити викрійку')
        self.pattern_button.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.pattern_button.clicked.connect(self.create_pattern.emit)
        self.pattern_button.setEnabled(False)  # Сделать кнопку недоступной по умолчанию
        self.addWidget(self.pattern_button)

        # Додаємо нову кнопку для збереження результату
        self.save_button = QToolButton()
        self.save_button.setIcon(QIcon('icons/save.png'))
        self.save_button.setText('Зберегти')
```

Рисунок 3.11 – Перша частина реалізації файлу toolbar.py

```
        self.save_button.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
        self.save_button.clicked.connect(self.save_result.emit)
        self.save_button.setEnabled(False)  # Зробити кнопку недоступною за замовчуванням
        self.addWidget(self.save_button)

        self.setStyleSheet("QToolButton { padding: 10px; }")

    1 usage
    def select_image(self):
        file_dialog = QFileDialog(self)
        if file_dialog.exec_():
            selected_files = file_dialog.selectedFiles()
            if selected_files:
                image_path = selected_files[0]
                self.image_selected.emit(image_path)

    1 usage
    def enable_pattern_button(self):
        self.pattern_button.setEnabled(True)

    1 usage
    def enable_save_button(self):
        self.save_button.setEnabled(True)
```

Рисунок 3.12 – Друга частина реалізації toolbar.py

Клас `Toolbar` успадковується від `QToolBar`. Визначаються два сигнали:

`image_selected` – сигнал передає шлях до обраного зображення у вигляді рядка та `create_pattern` – сигнал для створення викройки.

Конструктор `__init__` викликає конструктор батьківського класу `QToolBar` за допомогою `super().__init__()`. Встановлюється мінімальна висота панелі інструментів до 50 пікселів і розмір іконок на панелі 55x55 пікселів.

Далі робимо такі кроки для створення кнопки Відкрити:

– Створюється кнопка `QToolButton` для відкриття зображень.

– Встановлюється іконка для кнопки.

– Додаємо текст "Відкрити" під іконкою.

– Зв'язується натискання кнопки з методом `select_image` за допомогою `self.open_button.clicked.connect(self.select_image)`.

– Додається кнопка до панелі інструментів.

Дещо схоже робимо для кнопки *Визначити контур*:

– Створюється кнопка для визначення контурів.

– Встановлюється іконка та текст для кнопки.

– Додається кнопка до панелі інструментів.

Кнопка для створення викрійки:

– Створюється кнопка для створення викройки.

– Встановлюється іконка та текст для кнопки.

– Зв'язується натискання кнопки з сигналом *create_pattern* за допомогою *self.pattern_button.clicked.connectself.create_pattern.emit).*

– Кнопка за замовчуванням вимкнена (*setEnabled(False)),* доки не буде визначено контур.

– Додається кнопка до панелі інструментів.

Остання кнопка для збереження зображення:

– Створюється кнопка для створення викройки.

– Встановлюється іконка та текст для кнопки.

– Зв'язується натискання кнопки з сигналом *create_pattern* за допомогою *self.pattern_button.clicked.connect(self.create_pattern.emit).*

– Кнопка за замовчуванням вимкнена *(setEnabled(False)),* доки не буде визначено контур.

– Додається кнопка до панелі інструментів.

Додамо відступ до кожної кнопки по 10 пікселів.

Нижче є метод для додавання зображення – метод *select_image* відкриває діалогове вікно для вибору файлу. Якщо користувач вибирає файл, шлях до файлу передається через сигнал *image_selected*.

Останній метод *enable_pattern_button* активує кнопку "Створити викрійку", дозволяючи її натискання після визначення контуру.

3.4 Робоча область workspace.py

Розберемо тепер файл у якому було створено робочу область для відкритих зображень (рис. 3.13).

```
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QLabel
from PyQt5.QtCore import Qt

2 usages
class Workspace(QWidget):
    def __init__(self):
        super().__init__()

        self.image_label = QLabel('Для початку роботу відкрийте зображення')
        self.image_label.setAlignment(Qt.AlignCenter)
        self.image_label.setStyleSheet("background-color: white;")  # Установить белый фон

        layout = QVBoxLayout()
        layout.addWidget(self.image_label)
        self.setLayout(layout)

        self.setMinimumWidth(800)
        self.setMinimumHeight(600)

    2 usages
    def set_image(self, image_path):
        pixmap = QPixmap(image_path)
        scaled_pixmap = pixmap.scaled(self.image_label.size(), Qt.KeepAspectRatio)
        self.image_label.setPixmap(scaled_pixmap)

    1 usage
    def clear_image(self):
        self.image_label.clear()
```

Рисунок 3.13 – файл workspace.py

Клас `Workspace` успадковується від `QWidget`. В конструкторі `__init__` викликається конструктор батьківського класу `QWidget` за допомогою `super().__init__()`. Створюємо мітку `QLabel` для відображення зображення `(self.image_label)`. Встановлюємо вирівнювання мітки по центру за допомогою `self.image_label.setAlignment(Qt.AlignCenter)`. Далі створюється вертикальний макет `QVBoxLayout`, куди додається мітка

self.image_label та встановлюється макет для віджета за допомогою *self.setLayout(layout)*.

Метод *set_image* приймає шлях до зображення *(image_path)* і завантажує зображення за допомогою *QPixmap* і встановлює його на мітку *self.image_label* за допомогою *self.image_label.setPixmap(pixmap)*.

І останній метод *clear_image* очищає мітку, видаляючи зображення за допомогою *self.image_label.clear()*.

3.5 Створення діалогового вікна patterm_dialog.py

Імпортуються необхідні класи з модуля PyQt5 для створення діалогового вікна: *QDialog*, *QVBoxLayout*, *QLabel*, *QLineEdit*, *QPushButton* (рис. 3.14).

```
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QLabel, QLineEdit, QPushButton

2 usages
class PatternDialog(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("Створити викрійку")

        layout = QVBoxLayout()

        self.size_line_edit = QLineEdit()
        self.size_line_edit.setPlaceholderText("Розмір (см)")

        self.create_button = QPushButton("Створити")
        self.create_button.clicked.connect(self.accept)

        layout.addWidget(QLabel("Введіть розмір для створення викрійки:"))
        layout.addWidget(self.size_line_edit)
        layout.addWidget(self.create_button)

        self.setLayout(layout)

    def get_size(self):
        return self.size_line_edit.text()
```

Рисунок 3.14 – Файл pattern_dialog.py

Клас *PatternDialog* успадковується від *QDialog*, що дозволяє створювати діалогові вікна. Метод *__init__* є конструктором класу. Він викликає конструктор батьківського класу *QDialog* за допомогою *super().__init__()*.

Додаємо заголовок вікна діалогу за допомогою *self.setWindowTitle("Створити викрійку")*. Далі створюємо вертикальний макет *(QVBoxLayout*), в який будуть додаватися всі елементи управління (віджети).

Створюється поле для введення (*QLineEdit)*, де користувач вводить розмір у сантиметрах.

Встановлюється підказка в полі введення за допомогою *self.size_line_edit.setPlaceholderText("Розмір (см)")*.

Створюється кнопка (*QPushButton*) з написом "*Створити*".

Кнопка пов'язується з методом *accept*, який вбудований у *QDialog* і відповідає за підтвердження введених даних та закриття діалогового вікна. Це робиться за допомогою *self.create_button.clicked.connect self.accept)*.

До вертикального макета (*QVBoxLayout)* додаються:

– Текстовий напис *(QLabel)* з інструкцією для користувача.

– Поле введення (*QLineEdit)*.

– Кнопка (*QPushButton*).

Встановлюється макет для діалогового вікна за допомогою *self.setLayout(layout)*.

Метод *get_size* повертає текст, введений у полі *QLineEdit*.

Тепер коли ми розібрали основні компоненти перейдемо до тестування.

3.6 Тестування програми

Тестування програми здійснювалося крок за кроком, як з нею буде працювати майбутній користувач.

Запускаємо програму та щоб почати роботу з зображенням його треба відкрити, спробуємо спочатку це зробити через випадаючий список (рис 3.15):

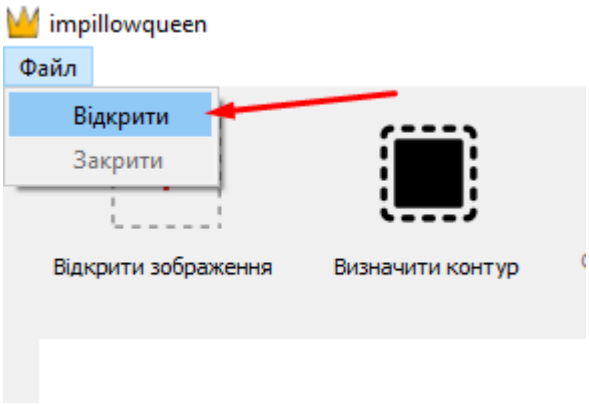


Рисунок 3.15 – Кнопка для додання зображення

З’являється стандартне вікно для вибору зображення на комп’ютері. Обрати необхідний файл з зображенням для редагування та натиснути кнопку *Відкрити*. Зображення відкривається у робочому просторі програми (рис. 3.16):

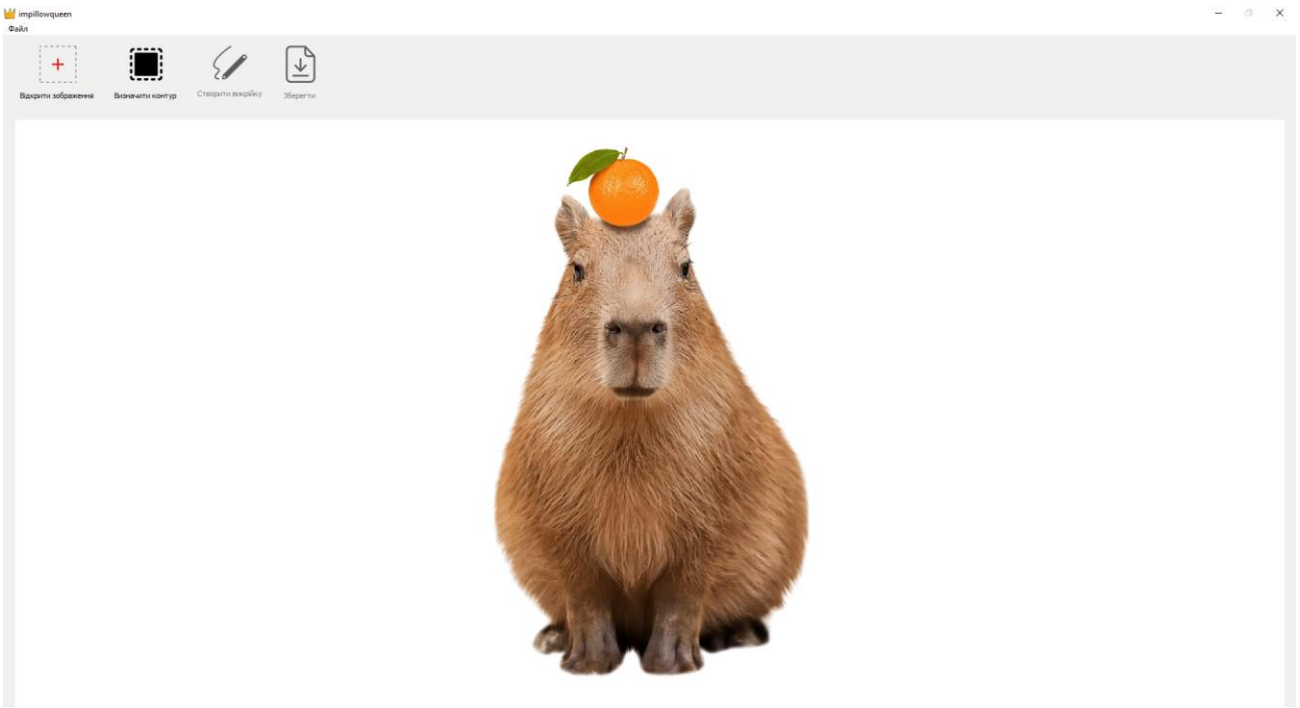


Рисунок 3.16 – Завантажене зображення у робочій області

Зображення відкрилося та знаходиться у межах робочого простору. Далі можна було б спробувати одразу створити викрійку не знайшовши контур. Але кнопка для цього не активна, доки не знайшли контур на зображенні, так само як і кнопка збереження файлу.

Перевіримо, як саме буде відмальовано контур для такого зображення (рис. 3.17):

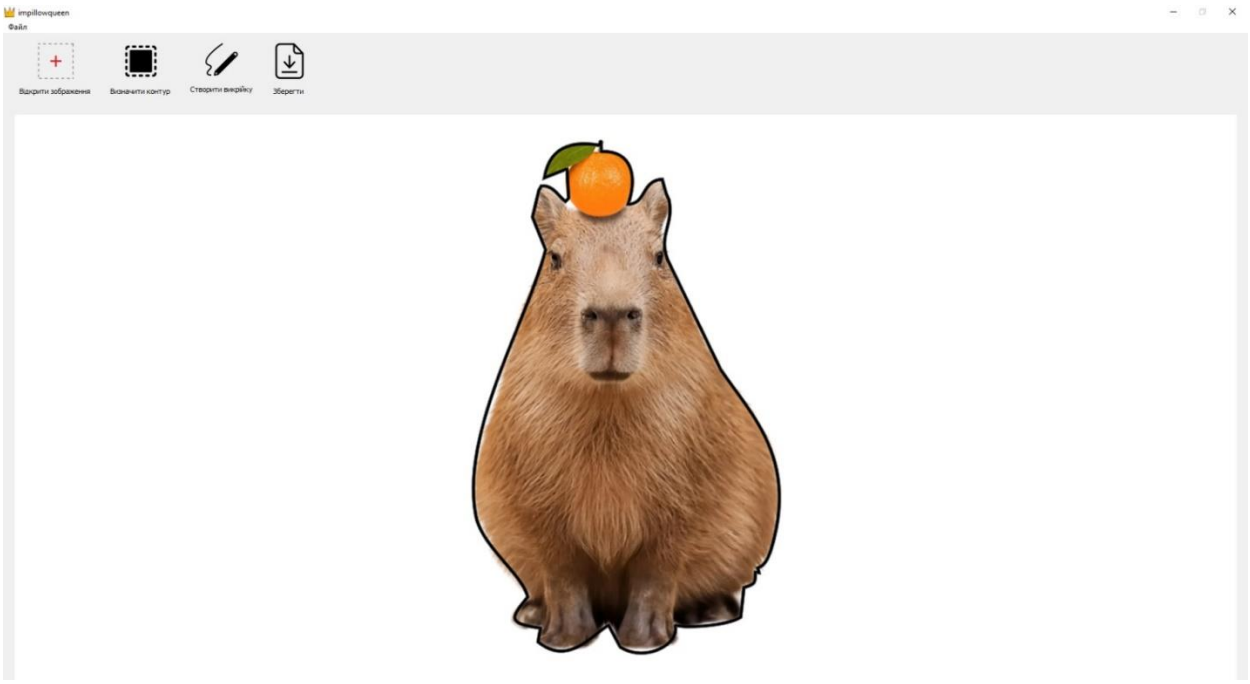


Рисунок 3.17 – Знайдений контур на фото

Як бачимо, відтворився контур предмета, який є на фото, він трохи деформований у деяких місцях, але нам і не треба щоб він був повністю на 100% за формою предмета на фото, оскільки коли буде шитись подушка її форма буде відступати трохи від самої викрійки.

Після цього, стали активні кнопки *Створити викрійку* та *Зберегти файл*. Спробуємо тоді його намалювати. Натискаємо кнопку створити викрійку та бачимо діалогове вікно у якому треба вказати на скільки см треба збільшити цей контур відносно зображення. Це фото було завантажене розміром 45 см, я знаю, що для подушки з таким розміром меня потрібно щоб відступ від предмета на фото був 3 см, тому в діалоговому вікні я вказую саме 3 см (рис. 3.18):

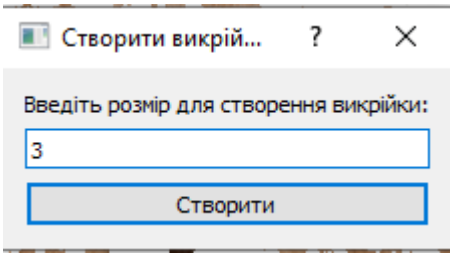


Рисунок 3.18 – Діалогове вікно для введення числа для утворення викрійки

Після введення числа, тиснемо створити і подивимось на результат, бачимо що контур збільшився на необхідну кількість см розширивши при цьому гострі кути на зображенні (рис. 3.19):

Рисунок 3.19 – Утворена викрійка

Результат нас цілком влаштовує і коли ми переконались що все підходить зберігаємо зображення.

До речі, поки що можна знаходити контур та робити викрійки тільки на підготовлених заздалегідь зображеннях, тобто на білому фоні, щоб було легше віднайти контур. Це не проблема, оскільки потрібні саме такі фото, для друку на папері фон повинен бути білий.

Спробуємо ще якесь зображення для перевірки робото спроможності (рис. 3.20 та рис. 3.21):

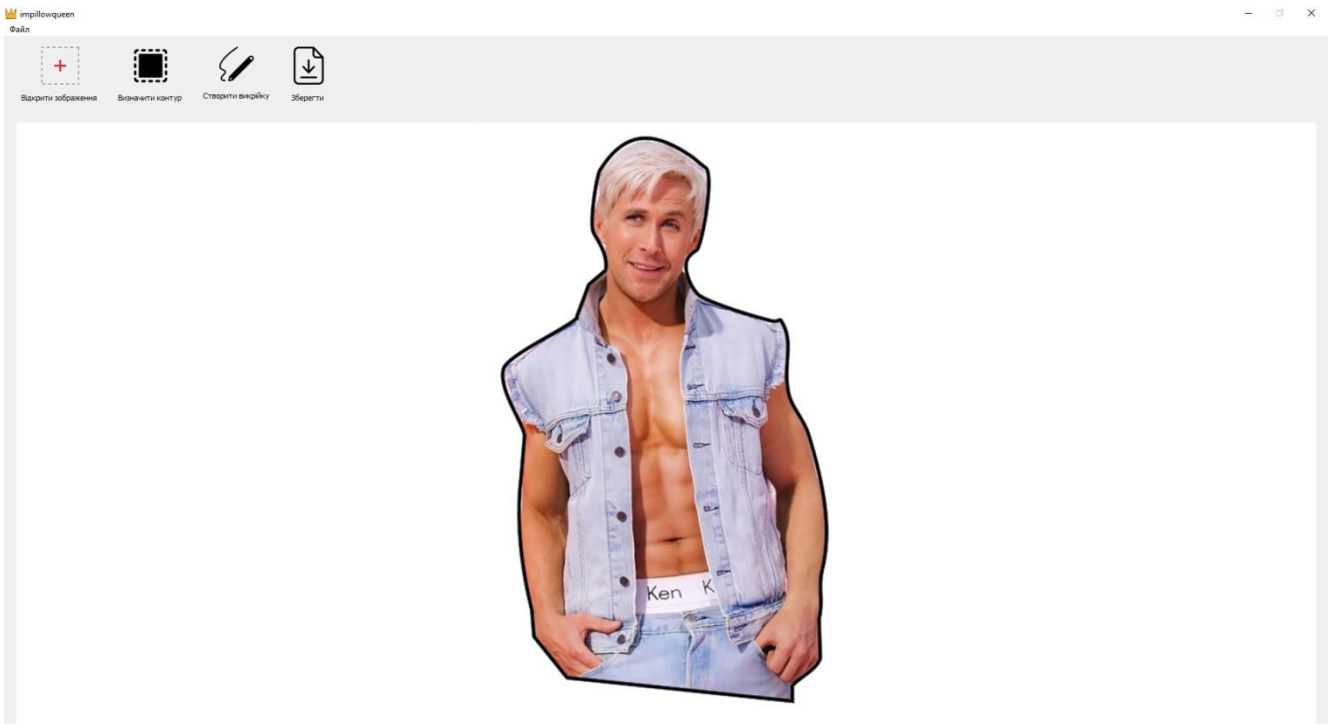


Рисунок 3.20 – Ще один контур на іншому фото

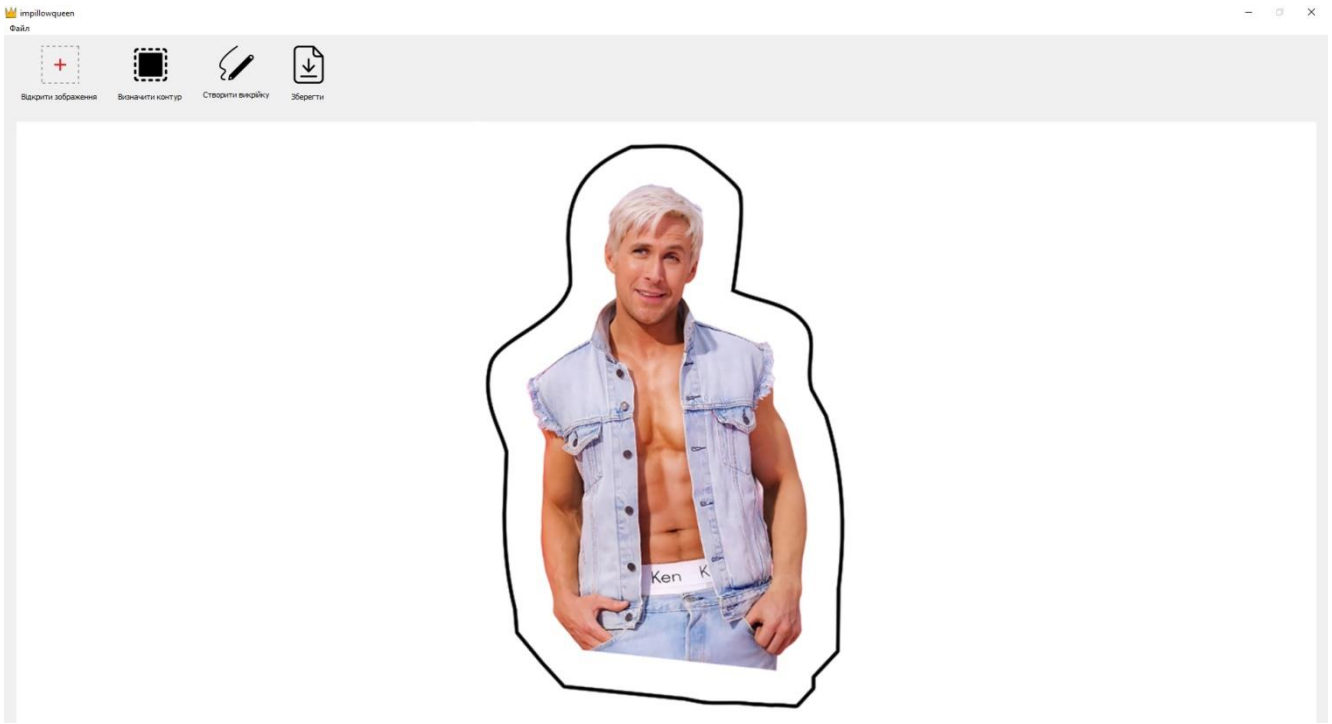


Рисунок 3.21 – Утворена викрійка із контуру

Результат знову бачимо, такий як треба і використовуючи такі створені викрійки тепер можна у декілька разів пришвидшити створення подушок,

оскільки тепер не потрібно витрачати час на те, щоб вручну малювати форму подушки на тканині.

3.7 Висновки до третього розділу

Третій розділ кваліфікаційної роботи присвячений процесу розробки програмного продукту, який був створений для реалізації системи формування зображень. Описані основні файли програми та програмні методи, які були в них реалізовані.

Наведено результати тестування програми на різних об’єктах в цифрових зображеннях.

ВИСНОВКИ

У кваліфікаційній роботі розроблена інтелектуальна система формування зображень засобами мови програмування Python та бібліотек PyQt5 і OpenCV.

Аналіз предметної області показав, що наразі не існує програм, які б дозволяли виконувати автоматично обробку та редагування цифрових зображень.

При роботі з цифровими зображеннями існує три напряму: аналіз, синтез та обробка.

Здійснено аналіз особливостей розпізнавання цифрових зображень які дозволяють визначати контури об’єктів.

Проаналізовано існуючі методи розпізнавання образів, які є наборами алгоритмів, які використовують для аналізу та класифікації цифрових зображень. Визначено їх переваги та недоліки.

Для інтелектуальної системи формування зображень визначено функціональні та нефункціональні вимоги. Графічно функціональні вимоги подано у вигляді діаграми варіантів використання.

При проєктування інтелектуальної системи формування зображень розглянута архітектура інформаційної системи, створено алгоритм розпізнавання контурів об’єктів на зображені. Проаналізовані існуючі способи виділення контурів за допомогою засобів розробки.

Розроблена інтелектуальна система формування зображень. Описані основні файли програми та програмні методи, які були в них реалізовані.

Тестування створеної програми виконувалася на різних об’єктах в цифрових зображень та показала ефективність виділення контурів.

Отже, розроблена програма надає можливість спростити процес обробки та редагування зображень, оскільки програма дозволяє завантажувати зображення, автоматично визначати контури предметів на зображенні і створювати викрійку обраного об'єкта за введеними параметрами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лисенко Г. Л. Сучасні тенденції у вирішенні задач виявлення та розпізнавання об`єктів на зображеннях [Текст] / Г. Л. Лисенко, М. Г. Тарновський, Л. В. Кузьменко // *Оптико-електронні інформаційно-енергетичні технології*. 2017. № 1. С. 18-23.

2. Anatomy on pattern recognition /Mayank Parasher, Shruti Sharma, A.K Sharma, J. P. Gupta // Indian Journal of Computer Science and Engineering. URL: https://www.researchgate.net/publication/228721547_ANATOMY_ON_PATTERN_RECOGNITION.

3. Методи обробки зображень. URL: https://nmetau.edu.ua/file/07_7.5_lbr_gr_rbr_.pdf

4. Методи розпізнавання образів: від простих до складних URL:https://www.simbirsoft.com/blog/metody-raspoznavaniya-obrazov-ot-prostykh-do-slozhnykh/

5. Березький О. М. Аналіз і синтез зображень на основі теорії алгебротопологічних структур : автореф. дис. на здобуття наук. ступеня докт. техн. наук : спец. 05.13.23 «Системи та засоби штучного інтелекту» / О. М. Березький. Львів, 2012. 38 с.

6. Функціональні та нефункціональні вимоги. URL: http://surl.li/plkbs

7. 10 найкращих бібліотек Python для GUI. URL: https://www.unite.ai/uk/10-best-python-libraries-for-gui/

8. Що таке бібліотека OpenCV? URL: https://www.geeksforgeeks.org/opencv-overview/.

9. Вступ до OpenCV. Комп'ютерний зір URL: https://itmaster.biz.ua/programming/vision/opencv.html

10. Алгоритм Кенні. URL: https://uk.wikipedia.org/wiki/Алгоритм_Кенні