ДОДАТОК А
АПРОБАЦІЯ НАУКОВИХ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

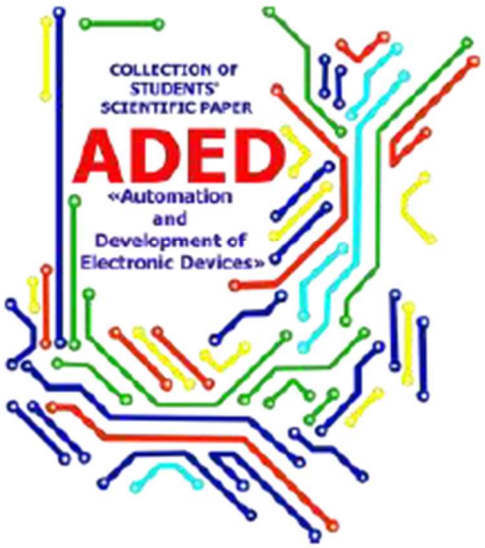
http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2024

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(КІТАР)



ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2024

(Випуск 2)

[електронне видання]

Харків 2024

ЗМІСТ

УДК 004.932

ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ БІБЛІОТЕКИ OPENCV ДЛЯ ПОШУКУ ДЕФЕКТІВ НА ПОВЕРХНІ ДРУКОВАНИХ ВИРОБІВ ЗА ТЕХНОЛОГІЄЮ FDM/FFF

В.В. Погребняк
Кафедра КІТАР, Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: viktoriia.pohrebniak@nure.ua

Анотація: У статті наведено методи обробки зображень за допомогою функцій бібліотеки OpenCV. Для проведення даного дослідження були розглянуті основні методи контролю якості у 3D-друці, зокрема із застосуванням технологій комп'ютерного зору. Виконано експериментальні дослідження впливу різноманітних фільтрів на точність виявлення дефектів. На базі проведеного дослідження буде розроблено рекомендації для оптимізації процесів контролю якості у 3D-друці.

Ключові слова: 3D-друк, 3D-принтери, FFF/FDM технологія, лінеарізація, фільтрація, згладжування.

RESEARCH ON IMAGE PROCESSING METHODS USING THE OPENCV LIBRARY FOR DEFECT DETECTION ON THE SURFACE OF FDM/FFF PRINTED PRODUCTS

Viktoriia Pohrebniak
Department of KITAP, Kharkiv National University of Radioelectronics
Ukraine, 61166, Kharkiv, 14 Nauky Ave.,
E-mail: viktoriia.pohrebniak@nure.ua

Annotation: The article presents image processing methods using OpenCV library functions. The research examines key quality control methods in 3D printing, particularly computer vision technologies. Experimental studies were conducted to analyze the impact of various filters on defect detection accuracy. Based on the findings, recommendations will be developed to optimize quality control processes in 3D printing.

Keywords: 3D printing, 3D printers, FFF/FDM technology, linearization, filtering, smoothing.

В сучасному світі технологія 3D-друку швидко набуває популярності і стає невід'ємною частиною багатьох галузей, включаючи медицину, автомобілебудування, авіакосмічну та будівельну промисловість. Однією з ключових проблем, з якою стикаються користувачі 3D-друку, є якість виготовлених деталей, зокрема наявність дефектів, що можуть впливати на функціональні характеристики продукту.

Для вирішення цієї проблеми дедалі більшого значення набувають програмні засоби для аналізу якості 3D-друку, розробка яких дозволяє автоматично виявляти дефекти на зображеннях готових виробів та проводити їх якісну оцінку. Програмні інструменти для аналізу дефектів підвищують стандарти виробництва, забезпечують контроль за кожним етапом виготовлення і дозволяють компаніям дотримуватись вимог сучасних ринків, де висока якість є обов'язковою умовою.

Для того щоб визначити чи має надрукована деталь дефект за допомогою програмного засобу, необхідно виконати попередню обробку зображень деталей, виготовлених методом 3D-друку. Методи обробки зображень та їх вплив на вихідні зображення і буде досліджено в даній статті.

Методи обробки зображень для виявлення дефектів у 3D-друці включають різноманітні технології, які допомагають визначати проблеми на різних стадіях друку.

В даній статті розглянемо наступні методи:

- лінеаризація;
- фільтрація;
- згладжування.

Лінеаризація зображення – це процес перетворення кольорового або градаційного зображення у форму, де інтенсивність пікселів зводиться до лінійного спектра. Цей метод використовується для усунення нерівномірностей у контрастності та яскравості зображення, що забезпечує стабільну базу для подальшої обробки та аналізу дефектів. Лінеаризація виконується для нормалізації зображення з метою вирівнювання контрасту, що полегшує подальше виявлення недоліків.

Процес лінеаризації часто починається із калібрування освітлення та корекції градацій зображення для вирівнювання контрасту. Зазвичай використовуються алгоритми корекції гамма, рівні сірого та адаптивні алгоритми для визначення рівномірного розподілу інтенсивностей пікселів. Деякі алгоритми лінеаризації застосовують методи на основі середніх значень, інші – гауссове розмиття для рівномірного розподілу пікселів по зображенню. Інструменти, які використовуються для лінеаризації, можуть включати такі бібліотеки, як OpenCV для Python або MATLAB з його набором для обробки зображень.

Результати лінеаризації дозволяють полегшити аналіз дефектів у 3D-друці, зокрема виявлення аномалій на поверхні об'єктів. Лінеаризоване зображення можна аналізувати за допомогою порогової обробки (бінаризації), де кожен піксель виводиться як «чорний» або «білий» залежно від інтенсивності, щоб чіткіше виділити контури або дефекти. Цей підхід допомагає виявити, наприклад, мікротріщини чи розшарування на друкованому об'єкті.

Після лінеаризації для подальшого виявлення дефектів застосовуються алгоритми порівняння аналогових точок та інші методи комп'ютерного зору, що аналізують лінійні розміри, відхилення та текстури поверхні.

Лінеаризацію зображення за допомогою бібліотеки OpenCV виконують наступним чином:

```
from PIL import Image, ImageOps, ImageEnhance
import numpy as np
import cv2

# Load the uploaded image
image_path = '/mnt/data/defekt-3d-druku-7.png.pagespeed.ce.DP8-awdMlm-2.png'
input_image = Image.open(image_path)

# Convert to grayscale
gray_image = ImageOps.grayscale(input_image)

# Enhance edges for linearization effect
enhancer = ImageEnhance.Contrast(gray_image)
enhanced_image = enhancer.enhance(2.0)

# Convert to numpy array for OpenCV processing
image_np = np.array(enhanced_image)

# Apply edge detection to emphasize contours (using Canny Edge Detection for linearization)
```

```
edges = cv2.Canny(image_np, 100, 200)

# Convert the edges back to PIL image
edges_image = Image.fromarray(edges)

# Save the processed image
output_path = '/mnt/data/linearized_3D_printed_part.png'
edges_image.save(output_path)
output_path
```

Розглянемо ці процеси на прикладі надрукованої деталі (рисунок 1).

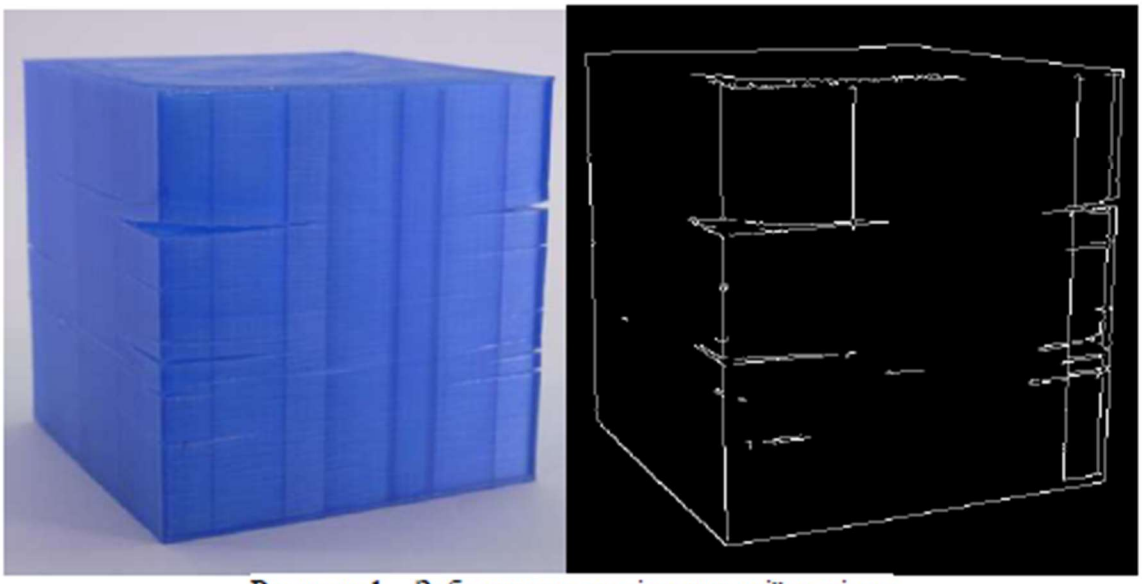


Рисунок 1 – Зображення до лінеаризації та після

Як видно на рисунку 1, виконана лінеаризація дозволяє побачити деталі структури та виявити можливі дефекти. Лінеаризація полягає у виділенні контурів та ліній на зображенні, щоб спростити аналіз і зробити дефекти більш помітними.

На цьому зображенні виділені контури, що можуть допомогти побачити деталі структури та виявити потенційні дефекти, наприклад, розшарування чи нерівності шарів

Фільтрація зображень – це процес обробки цифрових зображень, що дозволяє видалити шум, покращити якість зображення та виявити важливі особливості для подальшого аналізу. У контексті аналізу зображень 3D-друкованих деталей, фільтрація допомагає зосередитися на контрастних ділянках, підвищити чіткість контурів і виділити можливі дефекти, такі як нерівності шарів, тріщини чи інші артефакти, пов'язані з якістю друку. Фільтрація забезпечує виидалення шумів за допомогою фільтрів (наприклад, гаусового або медіанного) для чіткішого зображення об'єкту.

Фільтрація може бути виконана за допомогою різних методів, залежно від мети. Основні типи фільтрів включають:

– середньозважений фільтр використовується для згладжування зображення та зменшення шуму. Він працює шляхом заміни кожного пікселя на середнє значення інтенсивностей пікселів навколо нього. Цей метод ефективний для видалення дрібного шуму, але може знизити чіткість деталей;

– медіанний фільтр, що також зменшує шум, особливо імпульсний (спеклінговий) шум. Він замінює значення пікселя на медіанне значення в його околі, що дозволяє зберегти контури на зображенні;

– гауссовий фільтр згладжує зображення з меншим впливом на чіткі контури порівняно з середньозваженим фільтром. Застосовується для м'якого видалення шуму та покращення загальної якості зображення;

– лапласіанний фільтр використовується для виявлення контурів та виділення різких змін інтенсивності. Допомагає краще побачити грані та межі об'єктів.

Фільтрацію зображення за допомогою бібліотеки OpenCV виконують наступним чином:

```
from PIL import ImageOps, ImageEnhance

# Convert image to grayscale for edge detection
gray_image = ImageOps.grayscale(image)

# Enhance edges with Sobel filter by using FIND_EDGES and additional contrast enhancement
sobel_filtered_image = gray_image.filter(ImageFilter.FIND_EDGES)
enhanced_sobel_image = ImageEnhance.Contrast(sobel_filtered_image).enhance(2.0)
# Increase contrast for clarity

# Save the Sobel filtered image for download
sobel_filtered_image_path = "/mnt/data/sobel_filtered_image.png"
enhanced_sobel_image.save(sobel_filtered_image_path)

# Show the enhanced Sobel filtered image
enhanced_sobel_image.show()
sobel_filtered_image_path
```

Після фільтрації отримане зображення може бути менш зашумленим і з чіткішими контурами, що допомагає алгоритмам обробки зображень точніше виявляти дефекти. У випадку з 3D-друкованими деталями, результати фільтрації дозволяють більш ефективно проводити наступні етапи обробки, такі як лінеаризація та бінаризація (рисунок 2).

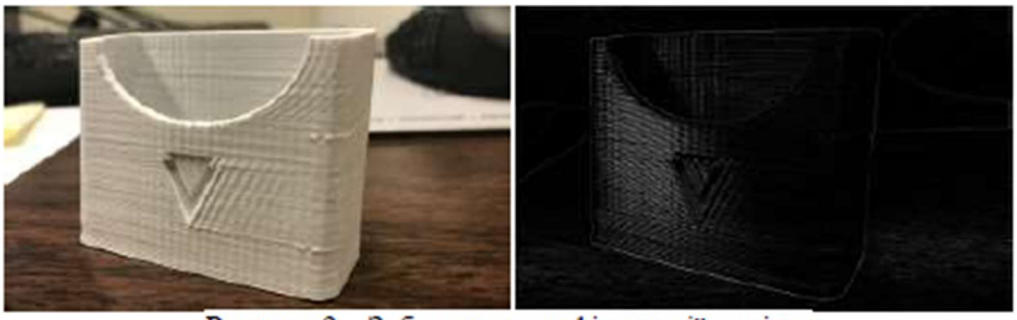


Рисунок 2 – Зображення до фільтрації та після

На рисунку 2 видно зображення після застосування фільтра Собеля, на якому чітко виділені контури 3D-моделі. Різкі зміни яскравості – це місця, де текстура або поверхня змінюється, наприклад, на краях або рельєфних частинах моделі. Фільтр підкреслив ці контури, зробивши

їх більш видимими, і також допоміг акцентувати увагу на дрібних деталях і можливих дефектах поверхні.

Після лінеаризації та фільтрації зображення стає більш придатним для алгоритмів виявлення дефектів, які можуть аналізувати чітко виділені контури, визначати точність розмірів і знаходити відхилення. Оброблені зображення можуть бути використані для створення карт дефектів, що допомагає визначити, які саме проблеми виникають під час 3D-друку.

Згладжування – це процес, який допомагає зменшити високочастотний шум, що може заважати точному розпізнаванню форм або дефектів на зображенні. Наприклад, при 3D-друці дефекти можуть проявлятися у вигляді дрібних нерівностей або артефактів на поверхні, і завдяки згладжуванню такі артефакти стають менш помітними або їх можна краще ізолювати для подальшого аналізу.

Процес згладжування зображення є важливою операцією у обробці зображень, оскільки він дозволяє зменшити шум та інші дрібні артефакти, які можуть спотворювати аналіз об'єкта, що зображений. Згладжування спрямоване на те, щоб зробити перехід між сусідніми пікселями більш поступовим, таким чином зменшуючи різкість змін інтенсивності, але при цьому зберігаючи основні форми об'єктів.

Згладжування зазвичай досягається шляхом застосування різних фільтрів, наприклад таких як:

– гауссовий фільтр;
– медіанний фільтр;
– фільтр середнього значення, який замінює кожен піксель середнім значенням інтенсивності його сусідніх пікселів, що зменшує варіативність інтенсивностей.

Фільтрацію зображення за допомогою бібліотеки OpenCV виконують наступним чином:

```
from PIL import Image, ImageFilter

# Load the image provided by the user
input_image_path = "/mnt/data/defekt-3d-druku-7.png.pagespeed.ce.DP8-awdMlm-2.png "
original_image = Image.open(input_image_path)

# Apply Gaussian blur for smoothing
smoothed_image = original_image.filter(ImageFilter.GaussianBlur(radius=2))

# Save the smoothed image
smoothed_image_path = "/mnt/data/smoothed_3D_printed_part.png"
smoothed_image.save(smoothed_image_path)

smoothed_image_path
```

Згладжене зображення з меншим рівнем шуму і плавнішими контурами допомагає краще виділяти основні дефекти, такі як нерівності поверхні або відхилення форми деталей, надрукованих на 3D-принтері. Після згладжування можна використовувати методи лінеаризації або інші техніки виділення контурів, щоб краще ізолювати ці дефекти для детального аналізу (рисунок 3).

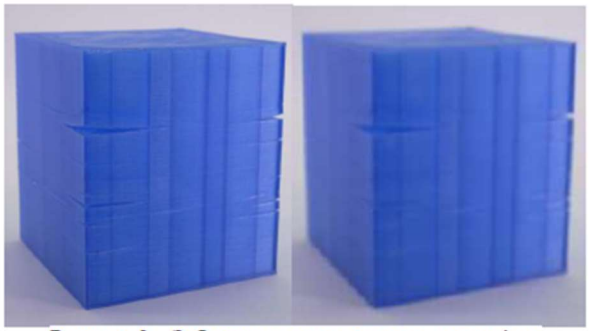


Рисунок 3 – Зображення до згладжування та після

Після згладжування зображення стає менш «зашумленим» і готовим для подальшого оброблення, наприклад, для бінаризації, виділення контурів або інших методів сегментації, що дозволяють чіткіше ідентифікувати потенційні дефекти на поверхні надрукованих деталей. Згладжене зображення допомагає уникнути хибнопозитивних дефектів, які могли б виникнути через дрібні шуми або текстурні артефакти.

Використання таких методів у комплексі дозволяє точно ідентифікувати різні види дефектів, як-от тріщини, розшарування або деформації, що виникають в процесі друку. Це особливо корисно для налаштування параметрів друку та покращення якості виробів у FFF/FDM технології.

ВИСНОВКИ. У даній науковій статті було досліджено методи обробки зображень для подальшого пошуку дефектів на поверхні друкованих виробів за технологією FDM/FFF, виконано збір даних, проведено експериментальні дослідження та детально описано отримані результати.

А саме виконано наступні етапи обробки за допомогою бібліотеки OpenCV:

- лінеаризація;
- фільтрація;
- згладжування.

По-перше, лінеаризація виявилася ефективним методом для нормалізації зображення, вирівнювання контрасту та підготовки даних до подальшого аналізу. Вона забезпечує чітке виділення контурів та спрощує процес виявлення дефектів.

По-друге, фільтрація дозволяє видалити шум, підвищити якість зображень і покращити ідентифікацію аномалій, таких як тріщини, нерівності шарів або інші дефекти, що виникають під час друку.

Крім того, згладжування допомагає зменшити рівень шуму та забезпечити плавні переходи між пікселями, що сприяє більш точному аналізу об'єктів та зменшенню ризику хибнопозитивних результатів.

Отже, можна стверджувати, що застосування методів лінеаризації, фільтрації та згладжування у комплексі дозволяє суттєво підвищити якість виявлення дефектів у деталях, надрукованих за допомогою FFF/FDM технологій, і є перспективним напрямом для оптимізації процесів контролю якості у 3D-друці.

ПЕРЕЛІК ПОСИЛАНЬ

1. Real-time defect detection for FFF 3D printing using lightweight model deployment [Електронний ресурс] /– Режим доступу: www / URL: https://link.springer.com/article/10.1007/s00170-024-14452-4

2. Gonzalez, R. C., & Woods, R. E. Digital Image Processing. Pearson, 2018.

3. Bradski, G., & Kaehler, A. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, 2008.

Консультант роботи: Нікітін Дмитро Олександрович, старший викладач каф. КІТАР Харківського Національного Університету Радіоелектроніки

Науковий керівник: Омаров Мурад Анверович, проф. каф. КІТАР Харківського Національного Університету Радіоелектроніки

ДОДАТОК Б
КОД ПРОГРАМИ ДЛЯ ОБРОБКИ ЗОБРАЖЕНЬ

Код для виконання лінеаризації зображень:

```
from PIL import Image, ImageOps, ImageEnhance
import numpy as np
import cv2

# Завантаження зображення
image_path = '/mnt/data/defekt-3d-druku-7.png.pagespeed.ce.DP8-awdMlm-2.png'
input_image = Image.open(image_path)

# Конвертація у відтінки сірого
gray_image = ImageOps.grayscale(input_image)

# Покращення країв для ефекту лінеаризації
enhancer = ImageEnhance.Contrast(gray_image)
enhanced_image = enhancer.enhance(2.0)

# Перетворення в масив numpy для обробки OpenCV
image_np = np.array(enhanced_image)

# Застосування визначення країв, щоб підкреслити контури (використовуючи Canny Edge Detection для лінеаризації)
edges = cv2.Canny(image_np, 100, 200)

# Перетворення країв назад на PIL зображення
edges_image = Image.fromarray(edges)

# Збереження опрацьованого зображення
output_path = '/mnt/data/linearized_3D_printed_part.png'
edges_image.save(output_path)
output_path
```

Код для виконання фільтрації зображень:

```
from PIL import ImageOps, ImageEnhance

# Перетворіть зображення на градації сірого для виявлення країв
gray_image = ImageOps.grayscale(image)

# Покращення країв фільтром Sobel за допомогою FIND_EDGES і додаткового покращення контрасту
sobel_filtered_image = gray_image.filter(ImageFilter.FIND_EDGES)
enhanced_sobel_image=ImageEnhance.Contrast(sobel_filtered_image).enhance(2.0)
# Збільшення контрастності для чіткості

# Збереження відфільтрованого зображення для завантаження
sobel_filtered_image_path = "/mnt/data/sobel_filtered_image.png"
enhanced_sobel_image.save(sobel_filtered_image_path)
```

Код для виконання згладжування зображень:

```
from PIL import Image, ImageFilter

# Завантаження зображення для обробки
input_image_path = "/mnt/data/defekt-3d-druku-7.png"
original_image = Image.open(input_image_path)

# Застосування розмиття за Гаусом для згладжування
smoothed_image = original_image.filter(ImageFilter.GaussianBlur(radius=2))

# Збереження згладженого зображення для завантаження
smoothed_image_path = "/mnt/data/smoothed_3D_printed_part.png"
smoothed_image.save(smoothed_image_path)
smoothed_image_path
```

Код запуску Python програми:

```python
import cv2
import hashlib
import numpy as np
from tkinter import Tk, Button, Label, filedialog, messagebox

# Функція для вибору шляху до файлу
def load_image_path(prompt):
    file_path = filedialog.askopenfilename(
        title=prompt,
        filetypes=[("Image files", "*.png *.jpg *.jpeg")]
    )
    if not file_path:
        print(f"{prompt}: Файл не обрано.")
        return None
    print(f"{prompt}: Файл успішно обрано:", file_path)
    return file_path

# Функція для обчислення MD5-хешу файлу
def calculate_md5(file_path):
    hasher = hashlib.md5()
    with open(file_path, 'rb') as f:
        buf = f.read()
        hasher.update(buf)
    return hasher.hexdigest()

# Функція для порівняння зображень
def compare_images(image1, image2):
    # Змінюємо розмір зображень до однакового розміру для порівняння
    image1_resized = cv2.resize(image1, (300, 300))
    image2_resized = cv2.resize(image2, (300, 300))

    # Обчислюємо абсолютну різницю між зображеннями
    diff = cv2.absdiff(image1_resized, image2_resized)

    # Перетворюємо різницю у чорно-біле зображення для аналізу
    gray_diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
    _, threshold_diff = cv2.threshold(gray_diff, 30, 255, cv2.THRESH_BINARY)

    # Рахуємо кількість "білих" пікселів (відмінностей)
    non_zero_count = cv2.countNonZero(threshold_diff)
    total_pixels = threshold_diff.size

    # Визначаємо відсоток схожості
    similarity = (1 - non_zero_count / total_pixels) * 100
    print(f"Схожість між зображеннями: {similarity:.2f}%")

    # Повертаємо результат
    return similarity
```

Код основного алгоритму програми:

```python
# Основна функція для запуску програми
def main():
    root = Tk()
    root.title("Інструмент пошуку дефектів")
    root.geometry("400x300")
    root.configure(bg="black")  # Фонова тема - чорний

    # Функція для запуску порівняння
    def on_compare():
        # Завантаження еталонного зображення
        reference_path = load_image_path("Оберіть еталонне зображення")
        if not reference_path:
            return

        # Завантаження нового зображення
        new_path = load_image_path("Оберіть зображення нової деталі")
        if not new_path:
            return

        # Перевірка шляхів
        if reference_path == new_path:
            messagebox.showerror("Помилка", "Однакові файли. Оберіть інший файл для порівняння.")
            return

        # Перевірка MD5-хешу
        reference_hash = calculate_md5(reference_path)
        new_hash = calculate_md5(new_path)
        if reference_hash == new_hash:
            messagebox.showerror("Помилка", "Файли ідентичні. Оберіть інше зображення для порівняння.")
            return

        # Завантаження зображень
        reference_image = cv2.imread(reference_path)
        new_image = cv2.imread(new_path)

        # Порівняння зображень
        similarity = compare_images(reference_image, new_image)

        if similarity > 80:
            messagebox.showinfo("Результат", "Деталь не має дефектів.")
        else:
            messagebox.showwarning("Результат", "Деталь має дефекти або значні відмінності.")
```

Код створення інтерфейсу програми:

```python
    # Заголовок
    title_label = Label(root, text="Пошук дефектів", font=("Arial", 16, "bold"), bg="black", fg="white")
    title_label.pack(pady=10)

    # Кнопка для запуску порівняння
    compare_button = Button(
        root,
        text="Оберіть зображення",
        font=("Arial", 12),
        bg="#006400",  # Зелений фон
        fg="white",  # Білий текст
        activebackground="#228B22",  # Темно-зелений при натисканні
        height=2,
        width=20,
        command=on_compare  # Виклик функції порівняння
    )
    compare_button.pack(pady=20)

    # Підпис
    footer_label = Label(
        root,
        text="© 2025 Defect Identifier Tool",
        font=("Arial", 10),
        bg="black",
        fg="white"
    )
    footer_label.pack(side="bottom", pady=10)

    root.mainloop()

if __name__ == "__main__":
    main()
```

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮИК.505300.006 ПЗ	Пояснювальна записка	95 с.,
			ф. А4
		Графічні документи	
2		Додаток А	ф. А4
		Апробація наукових результатів	
			10 с.,
3		Додаток Б	
		Код розробленого ПЗ	3 с.
4		Додаток В	
		Демонстраційний матеріал	
		(у форматі *.pptx)	10 слайдів

Змін.	Арк.	№ документа	Підп.	Дата	ГЮІК.505300.006 ВД		
Розроб.		Погребняк В.В.		15.01.25	Розробка системи візуального контролю пошуку дефектів на поверхні друкованих виробів за технологією FDM/FFF	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Омаров М.А.		15.01.25			
Н. контр.		Теслюк С.І.		15.01.25			
Затв.		Невлюдов І.Ш.		20.01.25		ХНУРЕ Кафедра КІТАР	