ECE 4050: Project 2 Report – Spring/Summer 2023

July 21, 2023

David Baron-Vega

Access ID: gf7068

OpenCV: Applications of Basic Image Processing

Table Of Contents:

I: Abstract:

This report outlines the development and implementation of a software console application capable of manipulating and processing digital images using the OpenCV open-source library. Written in C++ (**NOTE: Version 17**) using Visual Studio 2022 and using the capabilities and predefined functions available within the OpenCV library, the software hosts a user-friendly terminal UI that allows the user to navigate through a collection of images and apply various image processing techniques. These techniques include increasing/decreasing brightness, increasing/decreasing contrast, and the application of a median filter for noise reduction. Testing of the program and its functionalities demonstrates that the software operates effectively, smoothly transitioning between images and successfully applying multiple changes to the images which can be used to enhance or modify the visual characteristics of the images. The changes made by the user on any given image in the specified directory can be saved as a separate file, saved under the file name and type specified by the user.

Image processing has tons of applications across a broad spectrum of fields, including but not limited to: medical imaging (as is the topic of this project), remote sensing and object detection, machine vision and robotics, photography, and even social media (I have read that many of the algorithms used to identify what type of content a user engages with the most utilize computer vision and OpenCV!) . Just as in all realms of engineering and scientific research, the potential applications of image processing are limited only by the creativity and innovation of the designer.

II: Introduction:

In this project, we were tasked with constructing a program capable of applying a few fundamental image processing techniques on a collection of user-defined images. The aim of this assignment was not only to understand and implement algorithms behind these techniques, but also to gain hands-on experience with the OpenCV library, one of the most powerful and widely used libraries in image processing and computer vision.

The requirement of creating a user-friendly interface was accomplished using basic terminal-based UI prompts, which allows the user to navigate between images and apply desired modifications. This emphasized the need for a solid understanding of object-oriented programming and modularity, as functions of one class/file were called by functions of another class and within a different file. Through this project, we practiced using the programming skills taught in this course and some linear algebra concepts shown in class necessary to make image processing take place.

III: Methods and Software:

The software created is structured around two major classes: `ImageManager` and `UIManager`. The `ImageManager` class is responsible for performing all the processing of the images, implementing the editing functions, and storing the current state of the modified image. On the other hand, the `UIManager` class handles the interaction with the user through the command-line interface, enabling image navigation and editing based on user input. These two classes interact constructively, with methods from each being called upon in the other to complete the task and respond appropriately to user input.

The main.cpp file serves simply as the entry point for the program, initializing an instance of the start(); function from `UIManager`, from which all other necessary functions can be called via user input. The architecture of the entire program is quite simple, and the heavy lifting is really done by the predefined functions provided by the OpenCV library.

Below are detailed descriptions of all the functions declared and defined within the `ImageManager` class:

Loading the data:

1. `loadImages`: This function populates the images vector with Mat objects representing each image file from the provided filepaths. If an image cannot be loaded correctly, an error message is displayed and the image is skipped.

Moving through the ‘list’ of images:

2. `displayCurrentImage`, `nextImage`, `prevImage`: These functions allow navigation through the loaded images. `displayCurrentImage` shows the currently indexed image, while `nextImage` and `prevImage` increment or decrement the current image index, respectively, cycling through the images (matricies!) in our Images vector.

3. `increaseBrightness`, `decreaseBrightness`: These functions adjust the brightness of an image by adding or subtracting a constant value to each pixel. They are useful for correcting

underexposed or overexposed images. The value I selected to increment/decrement the value of the brightness of each pixel was 25, or roughly 10% of the maximum of 225 brightness that a grey-scale image/pixel can have.

4. `increaseContrast`, `decreaseContrast`: These functions adjust the contrast of an image by scaling the pixel values. They are used to enhance or soften the distinction between different elements of an image. The numerical arguments passed to this function represent percentages, but to be completely honest I did not dive deeper into the theory of how this works. I played around with the percentages until I achieved changes that were easily noticeable but not radical.

5. `applyMedianFilter`: This function applies a 3x3 (9-pixel definition) median filter to an image, useful for reducing "salt-and-pepper" noise. It replaces each pixel's value with the median value of the neighboring pixels. As explained in lecture, an odd-value matrix is ideal for this type of filtering as it assures that the matrix has one singular center value that can be ‘averaged’ so to speak, be the surrounding pixels.

6. `saveImage`: This function saves the **modified** image under a user-specified filename.

On the other hand, the `UIManager` class directs the user interface, controlling the interaction between the user and the `ImageManager`. Through a series of user prompts, it enables the user to select images, apply image edits, navigate between images, and save modifications. The few methods defined within the UIManager class are self-explanatory so I will not bore you by explaining these simple functions. These functions are simply responsible for creating the starting point of the program and outputting the menu options to the user, thus granting access to all the functions from the ImageManager class.

<u>Memory Allocation and Its Importance:</u>

Memory allocation is the process of reserving space in the computer's memory to store data and instructions. This process allows applications to have a dedicated space for their execution. It is the basis for storing variables, arrays, and objects.

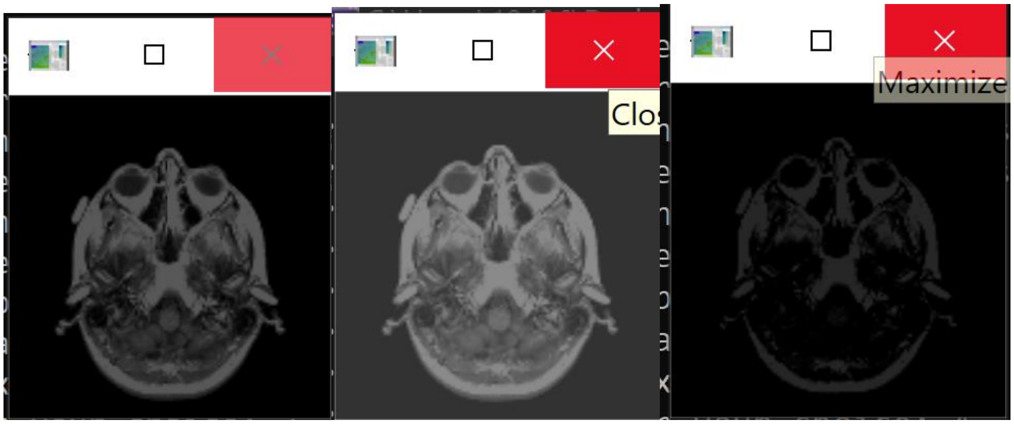
In the program, memory allocation plays a significant role in managing the images. When an image file is loaded into the software, memory is allocated to store the image's pixel data in a format that the OpenCV library can understand (in this case, the cv::Mat object). The size of the allocated memory depends on the image's dimensions and the color/depth. The reason memory allocation is important is that it allows for efficient use of resources. By allocating only the memory needed to store the image, the software avoids wasting memory space. In the program, memory allocation and deallocation are taken care of by the OpenCV library when creating and destroying cv::Mat objects. However, the principles of efficient memory use are still important and something we as software designers should understand to create efficient programs that do not slow down our machines or bloat over time.

IV: Results, Discussion, Conclusion:

The image processing software I have created showed its ability to perform basic image processing techniques effectively on the provided grayscale MRI images. The software manipulated brightness and contrast successfully and could apply a median filter, as per the project requirements.

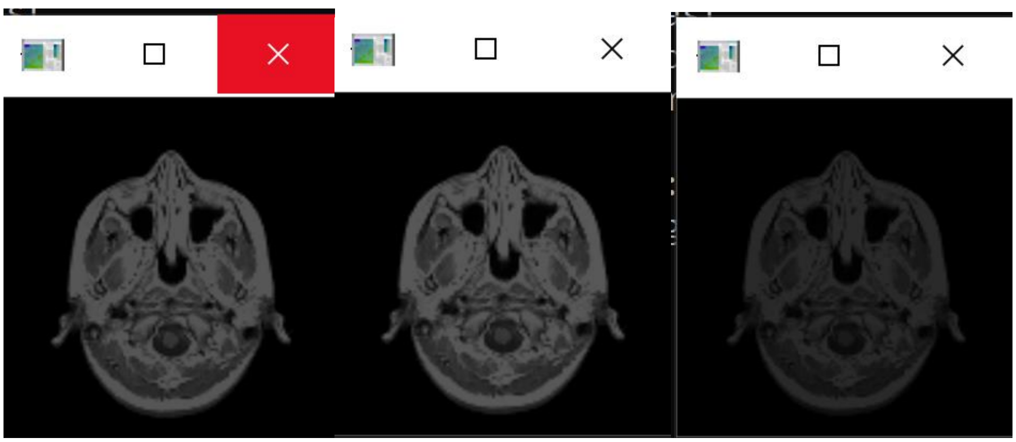
The software was tested using 28 grayscale MRI images, and regardless of the image dimensions or specific patterns in the image, the software functioned as intended without any issues. On adjusting the brightness, the software demonstrated that it could both brighten and darken the images effectively. Similarly, the program was able to dim an overly bright image without losing its essential structural details.

Provided Image, Brightened Image, Darkened Image:



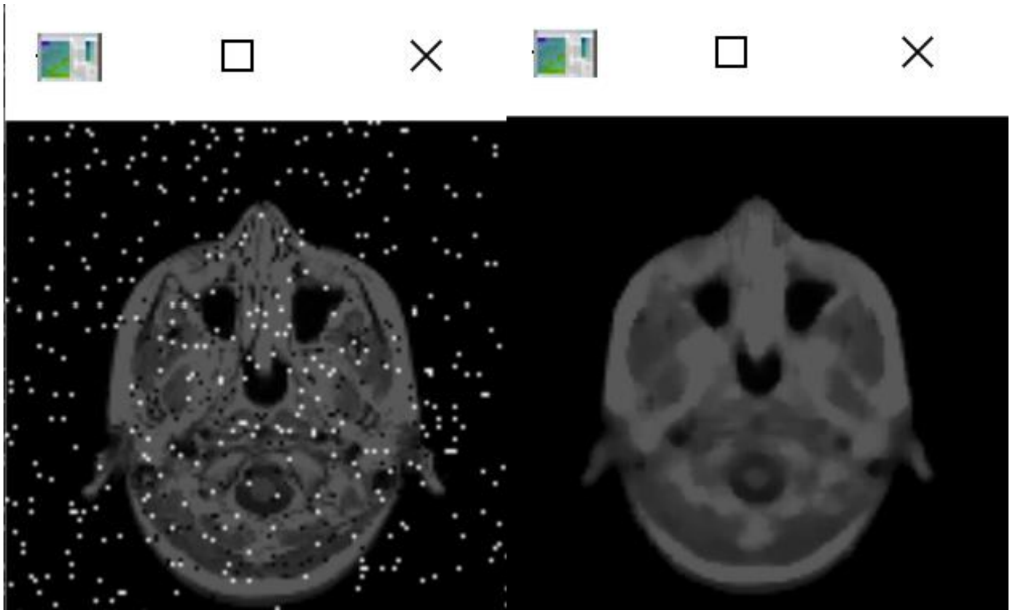
The contrast adjustment feature of the software also produced the expected results. When the contrast was increased, the grayscale intensities became more distinguishable, making the images clearer. Conversely, reducing the contrast brought the grayscale intensities closer, which made the images blurrier but still maintained their overall structure.

<u>Provided Image, +Contrast Image, -Contrast Image:</u>



The median filter, implemented as a 3x3 pixel-matrix as per the project requirement, was effective in noise reduction. This feature was useful in smoothing out the salt-and-pepper noise that can be often seen in MRI images.

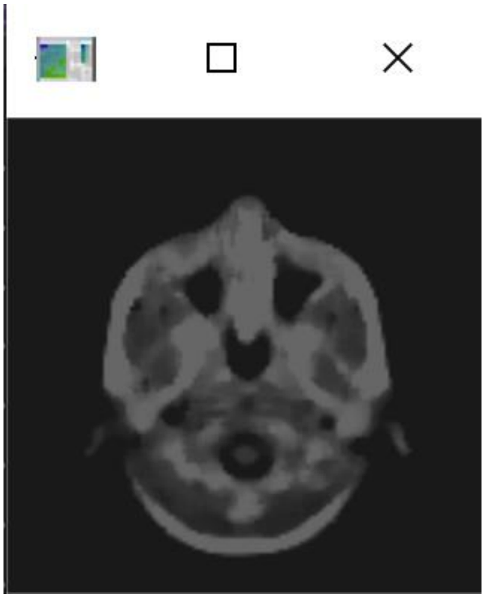
<u>Before and After Applying Median Filter:</u>



The software's console-based user interface was simple but effective. The UIManager class guided the user through each step of the process, from loading the image, applying modifications, to finally saving the processed image. This allowed for an acceptable user experience, enabling the user to iterate through images and apply different modifications without having to restart the program.

The before and after images of the median filter in use show that, even after filtering, the images may benefit from a modulation in brightness and contrast. The image below shows the same image after it has been filtered, increased contrast, and decreased brightness.

Noisy Image after filter, +20% contrast, -10% brightness:



The software’s ability to stack multiple changes and save these changes demonstrates the program’s full capabilities. In the image above, compared to the image which only applied filtering, we can see different regions of the brain highlighted and contrasted with better accuracy.

Other Image Processing Techniques Applicable to MRI Imaging:

Object Detection and Machine Learning

OpenCV, apart from its image processing capabilities, also has a capable set of tools for machine learning and object detection. This includes prebuilt functions for classification, clustering, and regression. In medical imaging and MRI scans, object detection algorithms could be used to identify and isolate specific regions of interest within the image. For example, these algorithms could be trained to detect different anatomical structures like the brain, spine, or other organs. with these OpenCV machine learning and object detection functionalities, we could enhance our software's ability to analyze MRI scans, enabling it to detect organs or even abnormalities, and learn from the features in the scans that can be used to train the machine learning models.

To conclude: the software developed has effectively demonstrated the basic image processing techniques. The application, while straightforward and simple, offers valuable insights into the potential of digital image processing. It must be noted, though, that this is a fundamental representation of the possibilities in the field of image processing. There are many more advanced techniques that can be explored for a wider range of applications.

Thank you.

V: Appendix of Code:

Below are screen captures of all the code written and used for execution. The files and code submitted are exactly equivalent to what is shown below.

Main.cpp File:

```
main.cpp    UIManager.h    UIManager.cpp    ImageManager.h    ImageManager.cpp
ECE4050.Project2.OpenCV    (Global Scope)

//
// David Baron-Vega
// Access ID: GF7068
// Created in July of 2023 for Spring/Summer ECE4050.
// ECE4050.Project2.OpenCV.cpp : This file contains the 'main' function. Program execution begins and ends there.
//



#include <vector>
#include <string>
#include <opencv2/core/core.hpp> //accessing core functionalities of OpenCV necessary for all OpenCV applications.
#include <opencv2/highgui/highgui.hpp> //access to a GUI created by OpenCV.
#include <opencv2/imgproc.hpp>  //Access to image process functions and capabilties defined by OpenCV.
#include "ImageManager.h"       // Including all of the written header files to access functions.
#include "UIManager.h"
#include <iostream> //standard reading in/out to console for c++.
#include <filesystem> //This is a open-source library available for C++ 17 or later compatible compilers.
                      //This library may allow us to simplify our code and navigate images more efficiently (both lines of code and compile-time wise.)

    using namespace cv;     // allows us to use classes and function defined in the CV libraries.
    using namespace std;    // basic I/O namespace. Getting in the habit of explicitly writing out namespaces for greater flexibility, though.
    namespace fs = filesystem; //creating an alias for std::filesystem to make the code look cleaner.


//The entire functionality of this program is created and described within the UIManager.cpp file and the ImageManger.cpp file
//All that is needed to initialize and access all the neccessary classes and functions can be accessed within this one simple start command!


    int main()
    {
        UIManager uimanager;
        uimanager.start();
        return 0; //return value will appear when program executes correctly.
    }
```

ImageManager.h File:

```cpp
//declaration of our ImageManager class and all of the member functions necessary for performing the required image processing.
#ifndef IMAGE_MANAGER_H // These guards prevent the header file from being read/overwritten more than once.
#define IMAGE_MANAGER_H

#include <opencv2/core/core.hpp>
#include <vector>
#include <string>

using namespace cv;
using namespace std;

class ImageManager
{
public:
    ImageManager(); //const.
    ~ImageManager(); //deconst.

    //Below are all the key member functions (methods) that will handle the image processing and movement from image to image:

    void loadImages(const std::vector<std::string>& filepaths);
    cv::Mat displayCurrentImage();
    cv::Mat nextImage();
    cv::Mat prevImage();
    cv::Mat increaseBrightness();
    cv::Mat decreaseBrightness();
    cv::Mat increaseContrast();
    cv::Mat decreaseContrast();
    cv::Mat applyMedianFilter();
    void saveImage(const std::string& filename);


    //objects of our class that will help us with storing images in the 'images' vector, referencing them via the index,
    //and objects that will allow us to save edited images without overwriting original images.
    std::vector<cv::Mat> images;
    int current_image_index;
    cv::Mat currentImage;
    cv::Mat modifiedImage;

private:

};

#endif  // IMAGE_MANAGER_H
```

ImageManager.cpp file:

main.cpp | UIManager.h | UIManager.cpp | ImageManager.h | **ImageManager.cpp**

ECE4050.Project2.OpenCV — (Global Scope)

```cpp
#include "ImageManager.h"
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

// Constructor and deconstructor of our ImageManager class:
ImageManager::ImageManager()
{
    // This object allows us to select and sort through the directory/vector of images.
    current_image_index = 0;

    // Initializing our new Mat object to an empty matrix for subsequent image modifications.
    modifiedImage = cv::Mat();
}
ImageManager::~ImageManager()
{
}

// Defining the created functions:

/* This 'load images' function below, which belongs to the class ImageManager, allows us to index and vectorize
the string of filepaths to access different images easily. */
void ImageManager::loadImages(const vector<string>& filepaths)
{
    for (const auto& filepath : filepaths)
    {
        Mat image = imread(filepath);
        if (image.empty())
        {
            cout << "No images were not found in the provided filepath." << endl;
            continue; // Skip to the next iteration
        }

        images.push_back(image); //This will add filepaths of images to our images vector, which we can reference to access all the images in the directory.
    }

    // Reset current image index after loading new images
    current_image_index = 0;
}

/* This 'display current image' function below, which belongs to the class ImageManager, allows us display the
currently indexed image using the imshow function available throught the OpenCV libraries. */
cv::Mat ImageManager::displayCurrentImage()
{
    if (images.empty())
    {
        cout << "No images loaded." << endl;
        return cv::Mat();  // Returns an empty matrix if there are no images
    }
```

```cpp
        return cv::Mat();  // Returns an empty matrix if there are no images
    }

    currentImage = images[current_image_index];
    modifiedImage = currentImage;

    // Return the current image
    return currentImage;
}

//Function to select next image in the images vector.
cv::Mat ImageManager::nextImage()
{
    if (images.empty())
    {
        cout << "No images loaded." << endl;
        return cv::Mat();  // Returns an empty matrix if there are no images
    }

    // Moving to the next image, wrapping around at the end using modulo of our vector size
    current_image_index = (current_image_index + 1) % images.size();
    currentImage = images[current_image_index];
    modifiedImage = currentImage;

    // Return the new current image
    return currentImage;
}

//Function to select previous image in the images vector.
cv::Mat ImageManager::prevImage()
{
    if (images.empty())
    {
        cout << "No images loaded." << endl;
        return cv::Mat();  // Returns an empty matrix if there are no images
    }

    // Move to the previous image, wrapping around at the start using modulo of our vector size.
    current_image_index = (current_image_index - 1 + images.size()) % images.size();
    currentImage = images[current_image_index];
    modifiedImage = currentImage;

    // Return the new current image
    return currentImage;
}

//This function will increase the brightness of the currently displayed image.
cv::Mat ImageManager::increaseBrightness()
{
    // Increasing the brightness by adding a constant to every pixel, using ConvertTo OpenCV function:
    modifiedImage.convertTo(modifiedImage, -1, 1, 25); // increase brightness by 25

    // Return the modified image
    return modifiedImage;
}

//This function will decrease the brightness of the currently displayed image. Very similar to previous function.
cv::Mat ImageManager::decreaseBrightness()
{
    // Decreasing the brightness by subtracting a constant from every pixel:
    modifiedImage.convertTo(modifiedImage, -1, 1, -25);

    // Return the modified image
    return modifiedImage;
}

//This function will increase the contrast of the currently displayed image.
cv::Mat ImageManager::increaseContrast()
{
    // Increasing the contrast by scaling every pixel:
    modifiedImage.convertTo(modifiedImage, -1, 1.2, 0); // increase contrast by 20%

    // Return the modified image
    return modifiedImage;
}
```

```cpp
}

//This function will decrease the contrast of the currently displayed image.
cv::Mat ImageManager::decreaseContrast()
{
    modifiedImage.convertTo(modifiedImage, -1, 0.8, 0); // decrease contrast by 20%

    // Return the modified image
    return modifiedImage;
}

//This function is responsible for applying the median filter. This uses the available function for this provided by OCV. Will edit to function made from scratch further down.
cv::Mat ImageManager::applyMedianFilter()
{
    // Apply a median filter to the image:
    cv::medianBlur(modifiedImage, modifiedImage, 5); // using a 5x5 kernel

    // Return the modified image
    return modifiedImage;
}

// Function that allows us to save the edited image as a new file with user-input given name:
// NOTE: The new, modified image is saved into the solution folder and can be found there with the given name. Adding the .png or .jpg is very important or else it wont work!!
void ImageManager::saveImage(const std::string& filename)
{
    cv::imwrite(filename, modifiedImage);
}

```

UIManager.h file:

main.cpp | UIManager.h | UIManager.cpp | ImageManager.h | ImageManager.cpp

ECE4050.Project2.OpenCV | UIManager

```cpp
//This file contains all the class and member function definitions for the
#ifndef UI_MANAGER_H // These guards prevent the header file from being read/overwritten more than once.
#define UI_MANAGER_H

#include "ImageManager.h"
#include <string>

using namespace std;
using namespace cv;




class UIManager
{
public:
    UIManager();
    ~UIManager();

    void start();   //function called in the main, which provides routing to all other necessary functions.
    void displayMenu();
    void displayImage(const cv::Mat& image);
    ImageManager image_manager;  //**creating an object of our class here that allows us to reference the selected image and modify it temporarily and accordingly.

private:
    //I wanted to keep some of the member functions for UIManager and Image Manager private but was having some compiling issues
    //when nesting the class functions within one another, so I left them all public to avoid any issues.

};

#endif  // UI_MANAGER_H

```

UIManager.cpp file:

main.cpp | UIManager.h | UIManager.cpp | ImageManager.h | ImageManager.cpp

ECE4050.Project2.OpenCV | UIManager | ~UI

```cpp
//Definitions of all the functions declared within UIManager.h are found here.
#include "UIManager.h"
#include <iostream>
#include <filesystem>
#include <opencv2/core/core.hpp> //accessing core functionalities of OpenCV necessary for all OpenCV applications.
#include <opencv2/highgui/highgui.hpp> //access to a GUI created by OpenCV.
#include <opencv2/imgproc.hpp>

using namespace std;
using namespace cv;


UIManager::UIManager()  //class constructor
{

}

UIManager::~UIManager() //class deconstructor
{

}


//Function used to display the very first image in the file directory,
//and sequentially, any other image accessed by the user, using the index of the images vector:
void UIManager::displayImage(const cv::Mat& image)
{
    // Checking for failure/error to locate image
    if (image.empty())
    {
        std::cout << "Could not open or find the image!" << std::endl;
        return;
    }

    // Showing our image inside the created window with OCV functionalities:
    cv::imshow("Window", image);

    // Waiting an indefinite ammount of time for any keystroke in the window.
    cv::waitKey(0);
}

//Our 'root' function, that uses the displayMenu() function to call all other functions based on user input at the terminal:

void UIManager::start()
{
    // Defining the path to our images directory:
    std::string imagesDirectory = "C:\\Users\\12486\\Desktop\\ECE4050.Project2.OpenCV\\mridata";

    // Iterating over the files in the directory and storing each of their paths into our 'images' vector:
    std::vector<std::string> filepaths;
    for (const auto& entry : std::filesystem::directory_iterator(imagesDirectory))
    {
        filepaths.push_back(entry.path().string());
    }

    // Passing the filepaths to the loadImages function
    image_manager.loadImages(filepaths);

    // Continue with the display menu, where all of our imagemanager functions can be accessed:
    displayMenu();
}
```

```cpp
void UIManager::displayMenu()
{
    // Initial display of the first image
    displayImage(image_manager.displayCurrentImage());

    // Menu loop
    while (true)
    {
        std::cout << "\n\n=== Menu ===\n\n";
        std::cout << "1. Next Image\n";
        std::cout << "2. Previous Image\n";
        std::cout << "3. Increase Brightness\n";
        std::cout << "4. Decrease Brightness\n";
        std::cout << "5. Increase Contrast\n";
        std::cout << "6. Decrease Contrast\n";
        std::cout << "7. Apply Median Noise Filter\n";
        std::cout << "8. Save Current Image\n";
        std::cout << "9. Exit\n";
        std::cout << "Enter your choice: ";

        int choice;
        std::cin >> choice;

        switch (choice)
        {
        case 1:  // Next Image
        {
            displayImage(image_manager.nextImage());
            break;
        }
        case 2:  // Previous Image
        {
            displayImage(image_manager.prevImage());
            break;
        }
        case 3:  // Increase Brightness
        {
            displayImage(image_manager.increaseBrightness());
            break;
        }
        case 4:  // Decrease Brightness
        {
            displayImage(image_manager.decreaseBrightness());
            break;
        }
        case 5:  // Increase Contrast
        {
            displayImage(image_manager.increaseContrast());
            break;
        }
        case 6:  // Decrease Contrast
        {
            displayImage(image_manager.decreaseContrast());
            break;
        }
        case 7:  // Apply Median Noise Filter
        {
            displayImage(image_manager.applyMedianFilter());
            break;
        }
        case 8:  // Save Current Image
        {
            std::string filename;
            std::cout << "Enter the filename to save the image (include the extension, e.g., '.jpg'): ";
            std::cin >> filename;
            image_manager.saveImage(filename);
            std::cout << "Image saved as " << filename << ".\n";
            break;
        }
        case 9:  // Exit
            return;
        default:
            std::cout << "Invalid choice. Please enter a number between 1 and 9.\n";
        }
    }
}
```