IG.2409 Multimedia Application: Final Project

C.IMP

APICHONKIT Witchaya 62820
BIEGAS Jakub 62824
BIKTI Nizar 62825
SHARMA Rahul 62841

Docents: Dominic YOLIN, Ilaria RENNA

ISEP
Institut supérieur d'électronique de Paris

Summer Semester 2023

Contents

1 Introduction **2**

2 Overview **3**

3 Erode and Dilate **6**
3.1 Erode 6
3.2 Dilate 6
3.3 Erode and Dilate on Images 7
3.4 Erode and Dilate on Videos 8

4 Resize **10**
4.1 Resize on Images 10
4.1.1 Resizing an Image by using width and height 10
4.1.2 Resize on Images by Factor 11
4.2 Resize on Videos 13
4.2.1 Resize on Videos by Width and Height 13
4.2.2 Resize on Videos by Factor 14

5 Canny **16**
5.1 Canny Edge Detection on Images 16
5.2 Canny Edge Detection on Videos 17

6 Lighten and Darken **19**
6.1 Lighten and Darken on Images 19
6.2 Lighten and Darken on Videos 20

7 Face Detection **22**
7.1 Face Detection on Images 22
7.2 Face Detection on Video 24

8 Gaussian Blur **26**
8.1 Gaussian Blur 26
8.1.1 Kernel 26
8.1.2 Convolution 26
8.2 Gaussian Blur on Images 26
8.3 Gaussian Blur on Videos 28

9 Conclusion **30**

Chapter 1

Introduction

This report provides an overview of the development process and outcomes of our project to create a small GIMP-like image editor - **GIMPSEP-C-IMP** - using the OpenCV library in C++. Our project aimed to develop a modular and configurable image editor with basic functionalities. During the development of the project, we utilized various OpenCV functions and image processing techniques to implement the required features:

- Dilate and Erode;
- Resize;
- Lighten and Darken;
- Panorama and Stitching;
- Canny Edges Detection;
- Face Detection;
- Gaussian Blur.

Throughout the Multimedia Application course, we gained a solid understanding of the C++ programming language, including its syntax, data types, structures, functions, pointers, references, and object-oriented programming (OOP) principles. This foundation in C++ was crucial for effectively utilizing the OpenCV library. Then, we learnt many valuable skills in the field of computer vision as well as image and video processing with OpenCV library, including resizing images and videos, adjusting their brightness and contrast, rotating images, and applying transformations. These skills were essential for implementing the required functionalities in our image editor project. w

Deliverables

Link to the GitHub project: C.IMP - GIMPSEP

Chapter 2

Overview

The architecture of our application follows a modular and object-oriented design, incorporating two main classes: Gimpsep and GimpsepVideo. The **main** function of our application serves as the entrypoint of the program. It runs an infinite loop that continuously parses commands input by a user. It utilizes command-line parsing techniques to extract the desired command and any associated parameters from the user's input.

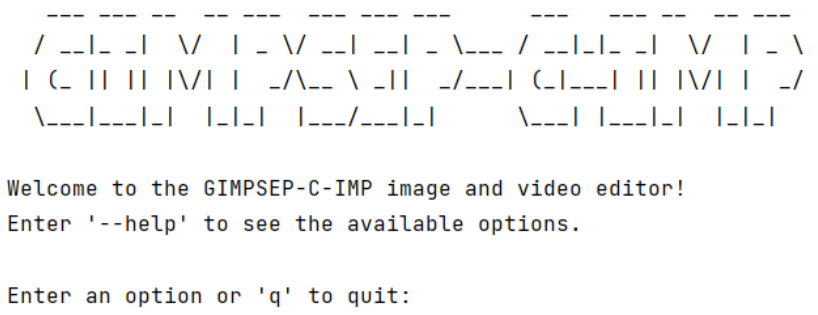

Figure 2.1: *Welcome Screen*

If the user enters the –**help** command, the application displays the manual, providing an overview of available commands and their usage.

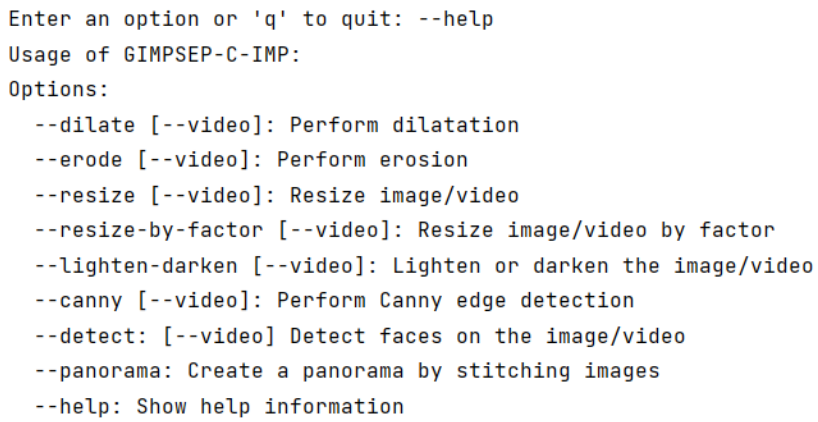

Figure 2.2: *Application Manual*

If the command corresponds to an existing function, the main function asks users for additional parameters and calls a corresponding member function of either **Gimpsep** or **GimpsepVideo** classes to perform the desired operation. All of the operations can be performed on images or videos, except the **–panorama** option which can be applied only on images. The user can always choose whether to display windows showing the results of a given operation.

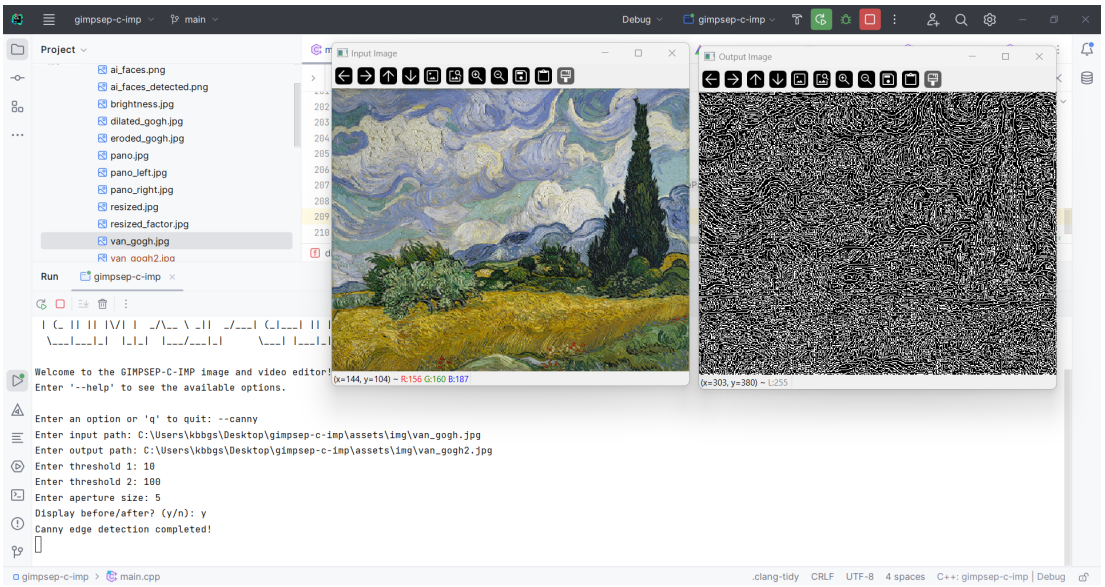

Figure 2.3: *Application Manual*

Every method in **Gimpsep** and **GimpsepVideo** class is **static**. This allows calling member functions easily from other places in the code.

The **Gimpsep** class is declared, encapsulating the functionalities of the image editor. It contains **private** and **public** sections. The private section includes two static member functions, **readImage** and **showImages**, which are helper functions. The public section contains the declarations of various member functions that implement the image manipulation functionalities. These function declarations represent the image manipulation functionalities. They are declared as **static** member functions, indicating that they can be called without instantiating an object of the class. These functions perform operations such as:

- dilatation
- erosion
- resizing
- brightness adjustment
- stitching
- canny edge detection
- face detection
- gaussian blur

The **GimpsepVideo** class is very similar to the **Gimpsep** class. The main difference is that it allows video manipulation instead of image manipulation. It contains private and public sections as well. The private section includes two static member functions, **readVideo** and **showVideos**. The public section contains the declarations of various member functions that implement the same operations as in the **Gimpsep** class but modified to allow video manipulation.

Chapter 3

Erode and Dilate

3.1 Erode

The erode function performs a morphological operation that reduces the size of foreground regions in an image by replacing each pixel with the minimum value in its neighborhood. The neighborhood of a pixel is defined by a structuring element, which is a small matrix or kernel that determines the shape and size of the neighborhood. In our case, we chose to use the `cv::MORPH_CROSS` as structuring element, so that the resulting effect could be easily recognizable by the user. In a future release we could let the user chose it. Some of the common use cases for the erode function are: noise reduction, object segmentation, edge detection and feature extraction.

```
void Gimpsep::erode(String &inputPath,
                    String &outputPath,
                    int size,
                    char verbose)
```

3.2 Dilate

The dilate function performs a morphological operation that expands or enlarges the size of foreground regions in an image by replacing each pixel with the maximum value in its neighborhood. It achieves this by adding pixels to the boundaries of the foreground regions, thereby increasing their size and thickness. Similarly to the erode function, the implemented structuring element is the `cv::MORPH_CROSS`. The dilate function is commonly used in image processing tasks such as noise removal, object segmentation, boundary extraction and feature enhancement.

```
void Gimpsep::dilate(String &inputPath,
                    String &outputPath,
                    int size,
                    char verbose)
```

3.3 Erode and Dilate on Images

Parameters

Given that the structure of the erode and dilate functions is very similar, they will be discussed collectively.

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output edges file.
- `size` (int): size of structuring element.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.
2. Apply the erode or dilate function using the respective OpenCV function. The provided size is used to initialize the size of the structuring element.
3. Save the resulting image to the specified `outputPath`.
4. If the `verbose` flag is set to 'y', show the original image and the resulting edges using the `showImages` function.

Example Usage

```
String inputPath = "path/to/input/image.jpg";
String outputPath = "path/to/output/image_out.jpg";
int size = 20;
char verbose = 'y';

void Gimpsep::erode(String &inputPath,
                    String &outputPath,
                    int size,
                    char verbose)
```

Figure 3.1: *Eroded image*

Figure 3.2: *Dilated image*

3.4 Erode and Dilate on Videos

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output edges file.
- `size` (int): size of structuring element.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.

   (c) Apply the erode or dilate function using the respective OpenCV function. The provided size is used to initialize the size of the structuring element.

   (d) Write the resulting images to the output video file.

3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.

4. Print a completion message indicating that the Canny edge detection process is finished.

5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
String inputPath = "path/to/input/image.jpg";
String outputPath = "path/to/output/image_out.jpg";
int size = 20;
char verbose = 'y';

void GimpsepVideo::erode(String &inputPath,
                    String &outputPath,
                    int size,
                    char verbose)

void GimpsepVideo::dilate(String &inputPath,
                    String &outputPath,
                    int size,
                    char verbose)
```

Chapter 4

Resize

Resizing of images refers to the process of changing the dimensions of an image. It allows to scale an image up or down to a desired size, either by specifying new width and height dimensions or by providing a scaling factor.

4.1 Resize on Images

4.1.1 Resizing an Image by using width and height

```
static void resizeImage(String &inputPath,
                        String &outputPath,
                        int width,
                        int height,
                        char verbose);
```

The **resizing function** applies the resize algorithm to the input image and saves the resulting image with the desired width and height given by the user in the specified output file.

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output image file.
- `width` (int): The desired width of the resized image.
- `height` (int): The desired height of the resized image.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.

2. Apply theResizing algorithm using the OpenCV `cv::resize` function, with the desired width and height.

3. The cv::INTERLINEAR interpolation method is used to resize the image smoothly

4. The resized image is saved to the output path specified by outputPath using `cv::imwrite`.

5. If the verbose flag is set to 'y', the `Gimpsep::showImages` function is called to display the original and resized images..

6. Finally, a message is printed to indicate the successful completion of the resizing operation.

Example Usage

```
Enter an option or 'q' to quit: --resize
String inputPath: "/path/to/input/image.jpg"
String outputPath: "/path/to/input/image_out.jpg"
int width: 300
int height: 300
Display before/after? (y/n): y
Resized image saved

Gimpsep::resizeImage(inputPath,outputPath,
                       width,  height, verbose);
```

Figure 4.1: *Resized image*

4.1.2 Resize on Images by Factor

```
static void resizeImage(String &inputPath,
                        String &outputPath,
```

```
                    double factor,
                    char verbose);
```

the **Gimpsep::resizeImage** function has been implemented to resize images using a scaling factor. This function allows users to adjust the size of an input image by a specified factor and save the image in the specified output file.

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output edges file.
- `Factor` (Double): The scaling factor for resizing. .
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.
2. Apply the resizing algorithm using the OpenCV `cv::resize` function, with the desired factor.
3. The cv::INTERLINEAR interpolation method is used to resize the image smoothly.
4. The resized image is saved to the output path specified by outputPath using `cv::imwrite`.
5. If the verbose flag is set to 'y', the `Gimpsep::showImages` function is called to display the original and resized images..
6. Finally, a message is printed to indicate the successful completion of the resizing operation.

Example Usage

```
Enter an option or 'q' to quit: --resize-by-factor
String inputPath: "/path/to/input/image.jpg"
String outputPath: "/path/to/input/image_out.jpg"
double factor: 2
Display before/after? (y/n): y
Resized image saved

Gimpsep::resizeImage(inputPath,outputPath,
                     factor, verbose);
```

4.2 Resize on Videos

4.2.1 Resize on Videos by Width and Height

```
static void resize(String &inputPath,
                   String &outputPath,
                   int width,
                   int height,
                   char verbose);
```

The **GimpsepVideo::resize** function, allows users to resize videos by specifying the desired width and height dimensions.

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `width` (int): The desired width of the resized video frames.
- `height` (int): The desired hight of the resized video frames.
- `verbose` (char): Determines whether to show intermediate videos. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.
   (c) Apply the Resizing algorithm to the current frame using the `OpenCV cv::resize function` function, with the provided height and width.
   (d) Write the resulting frames to the output video file.
3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.
4. Print a completion message indicating that the Resizing process is finished.
5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
Enter an option or 'q' to quit: --resize --video
String inputPath: "/path/to/input/video.mp4"
String outputPath: "/path/to/input/video_out.mp4"
int width: 300;
int height: 300;
Display before/after? (y/n): y

Gimpsep::resize(inputPath,outputPath, width,  height, verbose);
```

4.2.2 Resize on Videos by Factor

```
static void resize(String &inputPath,
                   String &outputPath,
                   double factor,
                   char verbose);
```

The **GimpsepVideo::resize** function, allows users to resize videos by applying a scaling factor.

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `Factor` (Double): The scaling factor for resizing. .
- `verbose` (char): Determines whether to show intermediate videos. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.
   (c) The current width and height of the input video frames are retrieved using the cv::VideoCapture object, and the new dimensions are calculated by multiplying the current dimensions by the scaling factor.
   (d) Apply the Resizing algorithm to the current frame using the `OpenCV cv::resize function`, with the provided scaling factor.

(e) Write the resulting frames to the output video file.

3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.

4. Print a completion message indicating that the Resizing process is finished.

5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
Enter an option or 'q' to quit: --resize-by-factor --video
String inputPath: "/path/to/input/video.mp4"
String outputPath: "/path/to/input/video_out.mp4"
double factor: 2;
Display before/after? (y/n): y

Gimpsep::resize(inputPath,outputPath, factor, verbose);
```

Chapter 5

Canny

The Canny edge detection algorithm is a technique to identify and extract edges from images. It aims to locate significant changes in pixel intensity that correspond to object boundaries, resulting in a binary image where the edges are highlighted. By adjusting the threshold values, the sensitivity and accuracy of the edge detection can be controlled. The canny edge detection is commonly used in various computer vision tasks, such as image segmentation, object detection, and feature extraction.

5.1 Canny Edge Detection on Images

```
void Gimpsep::cannyEdgeDetection(String &inputPath,
                                 double threshold1,
                                 double threshold2,
                                 int apertureSize,
                                 char verbose)
```

The `cannyEdgeDetection` function applies the Canny edge detection algorithm to the input image and saves the resulting image in the specified output file.

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output edges file.
- `threshold1` (double): The first threshold for the procedure.
- `threshold2` (double): The second threshold for the procedure.
- `apertureSize` (int): The aperture size.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.
2. Apply the Canny edge detection algorithm using the OpenCV `Canny` function, with the provided thresholds and aperture size.
3. Save the resulting edges to the specified `outputPath`.
4. If the `verbose` flag is set to 'y', show the original image and the resulting edges using the `showImages` function.

Example Usage

```
String inputPath = "path/to/input/image.jpg";
String outputPath = "path/to/output/image_out.jpg";
double threshold1 = 10.0;
double threshold2 = 100.0;
int apertureSize = 9;
char verbose = 'y';

Gimpsep::cannyEdgeDetection(inputPath, outputPath,
                            threshold1, threshold2,
                            apertureSize, verbose);
```

Figure 5.1: *Canny Edges Detected image.*

5.2 Canny Edge Detection on Videos

```
void GimpsepVideo::cannyEdgeDetection(String &inputPath,
                                      String &outputPath,
                                      double threshold1,
                                      double threshold2,
                                      int apertureSize,
                                      char verbose)
```

The `cannyEdgeDetection` function applies the Canny edge detection algorithm to a video file, frame by frame, and saves the resulting edges in a new video file.

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `threshold1` (double): The first threshold for the procedure.
- `threshold2` (double): The second threshold for the procedure.
- `apertureSize` (int): The aperture size.
- `verbose` (char): Determines whether to show intermediate videos. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.
   (c) Apply the Canny edge detection algorithm to the current frame using the OpenCV `Canny` function, with the provided thresholds and aperture size.
   (d) Write the resulting edges to the output video file.
3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.
4. Print a completion message indicating that the Canny edge detection process is finished.
5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
String inputPath = "path/to/input/video.mp4";
String outputPath = "path/to/output/video_out.mp4";
double threshold1 = 10.0;
double threshold2 = 200.0;
int apertureSize = 7;
char verbose = 'n';

GimpsepVideo::cannyEdgeDetection(inputPath, outputPath,
                                 threshold1, threshold2,
                                 apertureSize, verbose);
```

Chapter 6

Lighten and Darken

The lighten / darken algorithm is a technique to adjust the brightness of images and videos. It converts the input array of pixel values to the output array of the same type with an additional factor.

6.1 Lighten and Darken on Images

```
void Gimpsep::lightenDarken(String &inputPath, String &outputPath,
                            double factor, char verbose)
```

The `ligtenDarken` function applies lighten / darken algorithm to the input image and saves the resulting image in the specified output file.

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output image file.
- `factor` (double): The lighten / darken factor.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.
2. Apply the lighten / darken algorithm using the OpenCV `convertTo` function, with the provided factor.
3. Save the resulting image to the specified `outputPath`.
4. If the `verbose` flag is set to 'y', show the original image and the resulting image using the `showImages` function.

Example Usage

```
String inputPath = "path/to/input/image.jpg";
String outputPath = "path/to/output/image_out.jpg";
double factor = 50.0;
char verbose = 'y';

Gimpsep::lightenDarken(inputPath, outputPath,
                              factor, verbose);
```

Figure 6.1: *Lightened Image.*

6.2 Lighten and Darken on Videos

The `lightenDarken` function applies the lighten / darken algorithm to a video file, frame by frame, and saves the resulting frames in a new video file.

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `factor` (double): The lighten / darken factor.
- `verbose` (char): Determines whether to show intermediate videos. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.

(b) If the frame is empty (end of video), break the loop.

(c) Apply the lighten / darken algorithm to the current frame using the OpenCV `convertTo` function, with the provided factor.

(d) Write the resulting frames to the output video file.

3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.

4. Print a completion message indicating that the brightness adjustment process is finished.

5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
String inputPath = "path/to/input/video.mp4";
String outputPath = "path/to/output/video_out.mp4";
double factor;
char verbose = 'n';

GimpsepVideo::lightenDarken(inputPath, outputPath,
                                    factor, verbose);
```

Chapter 7

Face Detection

Face detection is a popular computer vision technique that involves locating and identifying human faces within images or video frames. OpenCV provides pre-trained models and algorithms specifically designed for face detection, making it easier to incorporate this functionality into our application.

The classifier and the model

We utilize the Haar cascade classifier, a machine learning-based technique used to identify specific objects or patterns within images. The model is initialized by loading the weights from a pre-trained model trained to detect faces, whose weights are stored in the 'haarcascade_frontalface.xml' file. The Haar cascade classifier utilizes a trained model that consists of multiple stages, each containing several weak classifiers. These weak classifiers are based on Haar-like features, which are simple rectangular features that capture variations in intensity within an image. The classifier scans the image using a sliding window technique, evaluating the presence of Haar-like features at different positions and scales. If a region matches the criteria for a face, it is considered a detection. The Haar cascade has several limitations, such as high false-positive detection and an overall accuracy smaller than to CNN based classifier. Nonetheless, they are suitable to recognize faces given that they are fast, simple to implement and do not require much computing power.

7.1 Face Detection on Images

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `cascadeModel` (String): The path to the pre-trained weights to be loaded in the model (not passed by the user).
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The implementaion follows these steps:

1. Load the pre-trained model into the cascade classifier from the file: assets/-models/haarcascade_frontalface.xml

2. Read the image from the specified `inputPath` and pass it to the detectAnd-Draw method.

3. Convert the image to grayscale.

4. Apply the loaded Haar cascade classifier to the grayscale image.

5. The face detection coordinates are saved in an array of rectangles.

6. Each detection (rectangle) is then drawn on the original image in. The result is the original image, where all the detected faces are marked by a blue rectangle.

7. The resulting image is then saved to the `outputPath`.

8. If the `verbose` flag is set to 'y', show the original image and the resulting edges using the `showImages` function.

Example Usage

```
String inputPath = "path/to/input/image.jpg";
String outputPath = "path/to/output/image_out.jpg";
cascadeModel =  "../assets/models/haarcascade_frontalface.xml";
char verbose = 'y';

void Gimpsep::faceDetection(String &inputPath,
                            String &outputPath,
                            String cascadeModel,
                            char verbose)
```

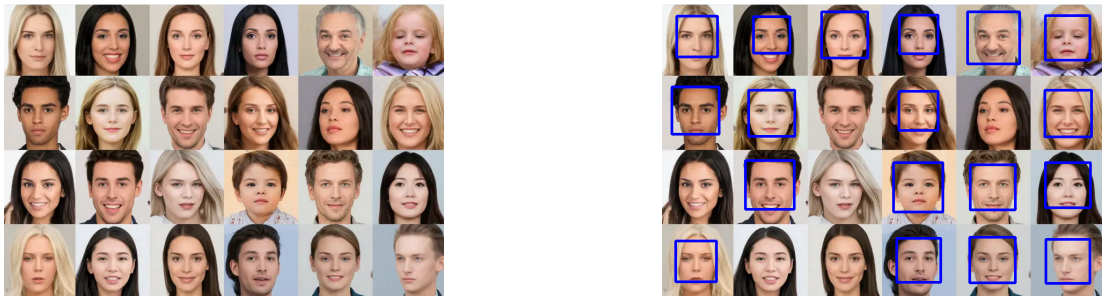

Figure 7.1: *Faces Detection.*

7.2 Face Detection on Video

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `cascadeModel` (String): The path to the pre-trained weights to be loaded in the model (not passed by the user).
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.
   (c) Pass the frame to the detectAndDraw method
   (d) Convert the image to grayscale.
   (e) Apply the loaded Haar cascade classifier to the grayscale image. The face detection coordinates are saved in an array of rectangles.
   (f) Loop on each detection (coordinates) and the rectangle are then drawn on the colored frame.
   (g) Write the resulting frame to the output video file.
3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.
4. Print a completion message indicating that the Canny edge detection process is finished.
5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
String inputPath = "path/to/input/video.mp4";
String outputPath = "path/to/output/video_out.mp4";
cascadeModel =  "../assets/models/haarcascade_frontalface.xml";
char verbose = 'y';
```

```
void GimpsepVideo::faceDetection(String &inputPath,
                                 String &outputPath,
                                 String cascadeModel,
                                 char verbose)
```

Chapter 8

Gaussian Blur

8.1 Gaussian Blur

Gaussian blur, or smoothing, is the most commonly used smoothing technique to eliminate noises in images and videos. In this technique, an image should be convolved with a Gaussian kernel to produce the smoothed image.

You may define the size of the kernel according to your requirement. But the standard deviation of the Gaussian distribution in X and Y direction should be chosen carefully considering the size of the kernel such that the edges of the kernel is close to zero. We will use the 3 x 3 and 5 x 5 Gaussian kernels. It's important to choose a right size of the kernel to define the neighborhood of each pixel. If it is too large, small features of the image may be disappeared and the image will look blurred. If it is too small, you cannot eliminate noises in the image.

The Gaussian blur operation is provided in OpenCV by void cv::GaussianBlur

The Gaussian blur operation involves convolving the image with a Gaussian kernel.

8.1.1 Kernel

Kernel is a matrix with an odd height and an odd width. It is also known as convolution matrix, mask or filter. Kernels are named based on their value distribution. Kernel is used to define a neighbourhood of a pixel in a image filtering operation.

8.1.2 Convolution

Convolution is a mathematical operation performed on images by sliding a kernel across the whole image and calculating new values for each pixels based on the value of the kernel and the value of overlapping pixels of original image.

8.2 Gaussian Blur on Images

```
static void gaussianBlur(String& inputPath,
                         String& outputPath,
                         cv::Size kernelSize,
                         char verbose);
```

Gaussian blur is a commonly used image filtering technique that applies a Gaussian function to blur an image. It is used to reduce noise, smooth image details, and remove high-frequency components.

Parameters

- `inputPath` (String): The path to the input image file.
- `outputPath` (String): The path to save the output image file.
- `Kernel Size` (cv::Size): The size of the Gaussian kernel specified as a cv::Size object with width and height.
- `verbose` (char): Determines whether to show intermediate images. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the image from the specified `inputPath`.
2. Apply gaussian algorithm using the OpenCV `cv::GaussianBlur` function, with the desired Kernel Size [3x3 and 5x5].
3. The resized image is saved to the output path specified by outputPath using `cv::imwrite`.
4. If the verbose flag is set to 'y', the `Gimpsep::showImages` function is called to display the original and resized images..
5. Finally, a message is printed to indicate the successful completion of the Gaussian blur operation.

Example Usage

```
Enter an option or 'q' to quit: --gaussian
String inputPath: "/path/to/input/image.jpg"
String outputPath: "/path/to/input/image_out.jpg"
Kernel Size: 3x3
Display before/after? (y/n): y

Gimpsep::gaussianBlur(inputPath, outputPath,  kernelSize,verbose)
```


Figure 8.1: *Gaussian Blurred image.*

8.3 Gaussian Blur on Videos

```
static void gaussianBlur(const std::string& inputPath,
                         const std::string& outputPath,
                         const cv::Size& kernelSize,
                         char verbose);
```

The **GimpsepVideo::gaussianBlur** function is responsible for applying Gaussian blur to each frame of a video

Parameters

- `inputPath` (String): The path to the input video file.
- `outputPath` (String): The path to save the output video file.
- `Kernel Size` (cv::Size): The size of the Gaussian kernel specified as a cv::Size object with width and height.
- `verbose` (char): Determines whether to show intermediate videos. Set to 'y' for yes, 'n' for no.

Algorithm

The function follows these steps:

1. Read the input video file and initialize the output video writer using the `readVideo` function, which returns a pair of `cv::VideoCapture` and `cv::VideoWriter`.
2. Iterate over each frame in the input video:
   (a) Read the next frame from the input video.
   (b) If the frame is empty (end of video), break the loop.
   (c) Apply the Gaussian algorithm to the current frame using the `OpenCV cv::GaussianBlur function`, with the provided Kernel size.
   (d) Write the resulting frames to the output video file.

3. Release the resources associated with the input video, output video, and temporary video capture and video writer objects.

4. Print a completion message indicating that the Gaussian Blur process is finished.

5. If the `verbose` flag is set to 'y', show the original input video and the resulting output video using the `showVideos` function.

Example Usage

```
Enter an option or 'q' to quit: --gaussian --video
String inputPath: "/path/to/input/video.mp4"
String outputPath: "/path/to/input/video_out.mp4"
Kernel Size: 3x3
Display before/after? (y/n): y

Gimpsep::gaussianBlur(inputPath, outputPath,  kernelSize,verbose)
```

Chapter 9

Conclusion

In conclusion, our project to develop a small GIMP-like image editor, **GIMPSEP-C-IMP**, using the OpenCV library in C++ has been successfully completed. Throughout the development process, we utilized various OpenCV functions and image processing techniques to implement the required features, including dilatation/erosion, resizing, lighten/darken, panorama/stitching, canny edge detection, and face detection.

Furthermore, we acquired valuable skills in the field of computer vision, as well as image and video processing, through our work with the OpenCV library. These skills included resizing images and videos, adjusting their brightness and contrast, rotating images, and applying various transformations. The application of these skills was crucial in implementing the required functionalities in our project.

Through this project, we have not only enhanced our technical abilities but also gained practical experience in software development, project management, and teamwork. We have learned the importance of effective planning, communication, and collaboration to ensure the successful completion of a complex project.

Overall, we are proud of the outcomes achieved in this project and confident in the quality of our outcome.