


# Semester Project Report

## GIMPSEP



MAILLEY Charles – 62458  
MOREAU Lucie – 62665  
VETTORATO Maxime – 62660

## Table of Content

I.	Introduction .....	3
II.	Functionalities .....	4
1.	Media Selection .....	4
2.	Image Transformations .....	5
a.	Brightness .....	5
b.	Rotation .....	5
c.	Resizing .....	5
d.	Cropping .....	6
e.	Canny Edge Detection .....	6
f.	Dilatation and Erosion .....	6
g.	Image Stitching and Panorama Creation .....	7
3.	Backward and Forward Steps .....	8
a.	Image Vector .....	8
b.	Backward Step: Control Z .....	8
c.	Forward Step: Control Y .....	8
d.	Reset .....	8
4.	Image Saving .....	9
III.	Code Structure .....	10
IV.	Graphic Interface .....	11
1.	Image .....	11
2.	Video and camera .....	11
3.	File Dialog .....	12
V.	Conclusion .....	13

# I. Introduction

This report outlines the development of a basic **GIMP-like image editor**, created as part of the IG.2409 - Multimedia Application semester project. The project offers several advantages, including its modularity, which facilitates easy code division for group work, and its adaptability, allowing the implementation of various functions and features based on team members interests.

The primary goal of this project is to provide a simple and effective tool for image manipulation using the **OpenCV** library. The core functionalities to be implemented include dilation and erosion, resizing, lightening and darkening of images, panorama stitching, and canny edge detection. Additional functionalities, such as video manipulation, advanced GUI, and face recognition, could also be implemented, requiring more advanced work.

The project is undertaken by a group of three working collaboratively and using a **GitHub repository** for code management. Each member of the group created their own branch to develop and test their respective parts of the work before merging all the changes into the main branch. Different IDEs have been used based on our personal preferences and the application has been developed in **C++**.

This report will detail the different development stages, the implemented functions, the image processing techniques used, and the results obtained. It will also highlight the challenges faced and the provided solutions, offering reflections on potential improvements and future extensions of the project.

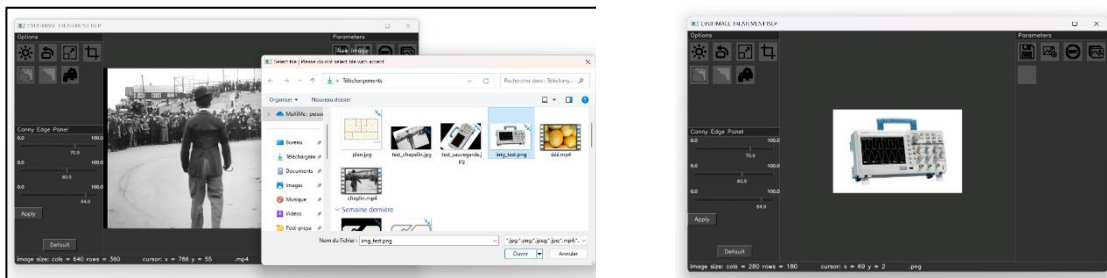
## II. Functionalities

### 1. Media Selection

Before applying any Image transformation, it is necessary to open an image or a video. In our application there is 2 ways to achieve that goal:

- ❖ Import an image/video using file dialog:

Users are free to select their media using our graphical interface, which includes a file dialog system powered by [TinyFileDialogs](#). This library allows users on both Windows and Linux to utilize the native file browser to specify file paths.



- ❖ Use the computer camera:

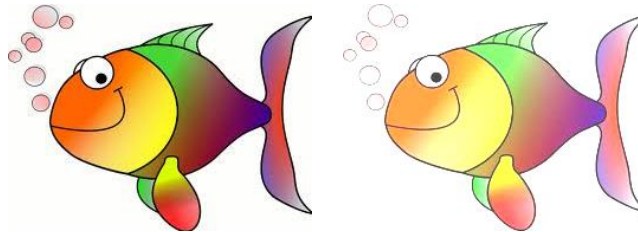
Alternatively, the user can use their camera. This option activates a video feed on the screen, and the user can stop the video to capture the last displayed frame and use it for any modifications.



## 2. Image Transformations

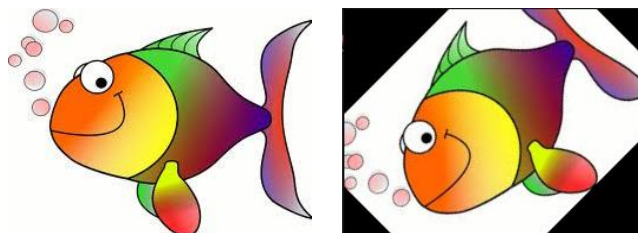
### a. Brightness

The brightness function **adjusts the lightness or darkness** of an image by adding a brightness factor to every single pixel. This brightness factor is the only parameter of the function and is an integer between -250 and 250, with 0 representing the original brightness level. Negative values darken the image, while positive values lighten it.



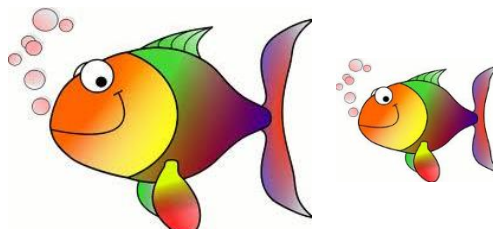
### b. Rotation

Rotating an image consists in **rotating the entire image left around a specified center point by an angle  $\alpha$** . The necessary parameters for this algorithm are a center point, designated by two integers (a,b) representing pixel coordinates in the image, and a rotation angle between 0° and 360°. By default, the center point is the center of the image, and the image's initial orientation is 0°. An angle of 180° flips the image upside-down, and 360° returns the image to its original orientation.



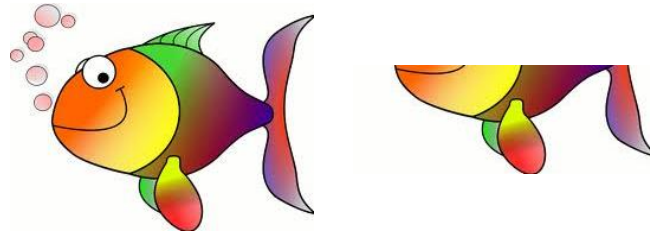
### c. Resizing

Resizing an image involves **scaling it up or down by a resizing factor**. This factor is a float between 0 and 5, where 0 makes the image disappear, 1 keeps it at its original size, values less than 1 shrink the image, and values greater than 1 enlarge it.



#### d. Cropping

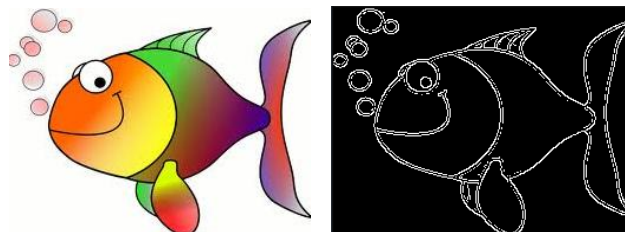
Cropping an image involves **cutting out the borders to retain a specific rectangular portion of the image**. The algorithm requires four values designating the columns and rows from which to cut. Initially, these values correspond to the minimum and maximum extents of the original image. After cropping, the resulting image becomes the new image on which future transformations will be applied.



#### e. Canny Edge Detection

Canny edge detection **identifies the edges** in an image based on two thresholds, iterating until stabilization. The process begins by choosing a low threshold to detect all edge points and a high threshold to detect only edge center points. Pixels above the high threshold are labeled as "edge points." For other pixels located between the two thresholds, if they have an edge point nearby, they are also labeled as "edge points." This algorithm iterates until no new points are labeled.

A **Gaussian blur** is applied before the Canny edge detection algorithm to reduce noise and blur the edges, making edge points easier to detect. The associated parameters are the size of a structural element and a blur factor between 0 and 1, with 0 representing no blur and 1 representing total blur.



#### f. Dilatation and Erosion

Erosion and dilation are morphological operations used to process binary images. Erosion **reduces the size of white structures** by eroding their boundaries, while dilation **increases the size of white structures** by expanding their boundaries. Both operations take a parameter that determines the size of the structuring element used with an integer between 1 and 99. Larger values result in more significant changes: greater erosion for erosion operations, causing the white structures to shrink more, and greater dilation for dilation operations, causing the white structures to grow more.



### g. Image Stitching and Panorama Creation

Image stitching involves **combining multiple images** to create a single panorama by recognizing overlapping patterns or common parts. The algorithm identifies key points and matches them across the images, aligning them accurately to recreate a large view panorama from multiple normal-sized images.

Input:



Output:



### 3. Backward and Forward Steps

#### a. Image Vector

To allow the user to undo modifications, a vector filled with images is created step by step as the user uses the application. The image vector has a fixed capacity, **storing up to 30 images**, enabling the user to revert up to 30 steps. This feature is especially useful during the cropping process, as it allows recovery from cuts that remove parts of the image.

#### b. Backward Step: Control Z

Our application supports the common **Ctrl+Z shortcut** for **undoing the last action**. When these keys are pressed, the previous image in the vector is displayed. Users can continue pressing Ctrl+Z to step back through their modifications until they reach the beginning of the image vector. Any changes made after one or multiple undo actions will apply to the currently displayed image.

#### c. Forward Step: Control Y

The **Ctrl+Y** shortcut is also implemented, allowing users to **redo actions that were undone**. The number of forward steps is limited to the most recent modification made before any undo actions. This feature helps users navigate back to their latest changes efficiently.

#### d. Reset

The reset function clears all previous modifications by emptying the image vector and displaying the original image. This provides a clean slate for users to start over with their edits.

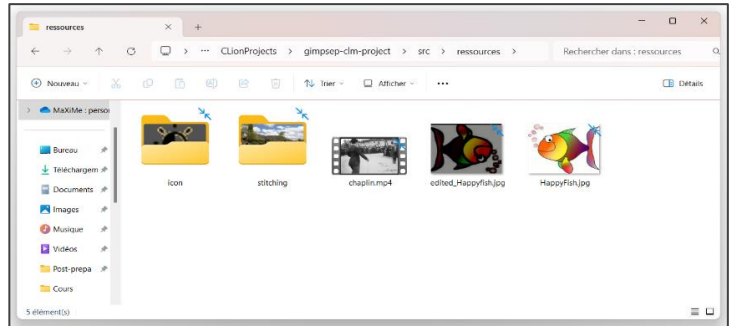
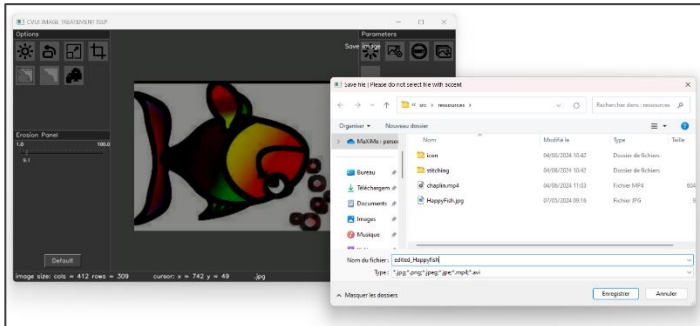


## 4. Image Saving

The image saving functionality in our application is designed to provide users with a flexible and user-friendly way to save their edited images **in various formats**. The method is implemented in the *saveImage* function of the *ImageApp* class.

It first opens a save file dialog using TinyFileDialogs library's *tinyfd\_saveFileDialog* function, allowing the user to specify the location and name of the file to save.

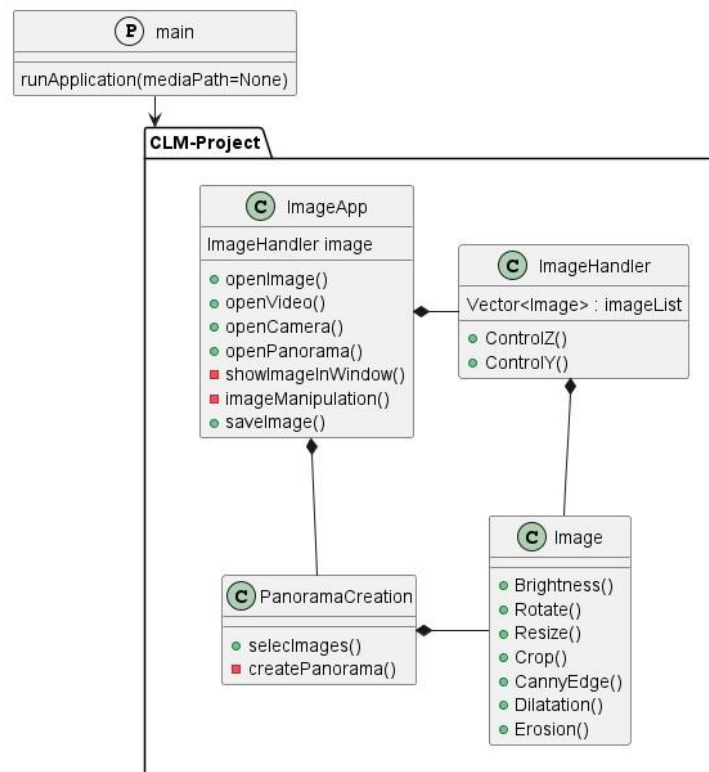
Then the path is stored as a string and the media is save at the path location.



### III. Code Structure

This application is designed with four components and is run through the main:

- **ImageApp**: This is the main class of the application. It provides the ability to open an image, video, camera, or panorama. Once the media is opened, any modifications can be saved.
- **ImageHandler**: This class manages a list of images. It comes equipped with undo (ControlZ()) and redo (ControlY()) operations, allowing easy navigation through changes.
- **Image**: This class represents an individual image. It offers a variety of manipulations such as adjusting brightness, rotating, resizing, cropping, applying a Canny Edge detection filter, and performing dilation and erosion operations. This provides a wide range of tools for image manipulation.
- **PanoramaCreation**: This class is for creating a panorama. Multiple images can be selected, and a panorama will be created.



The application uses three libraries:

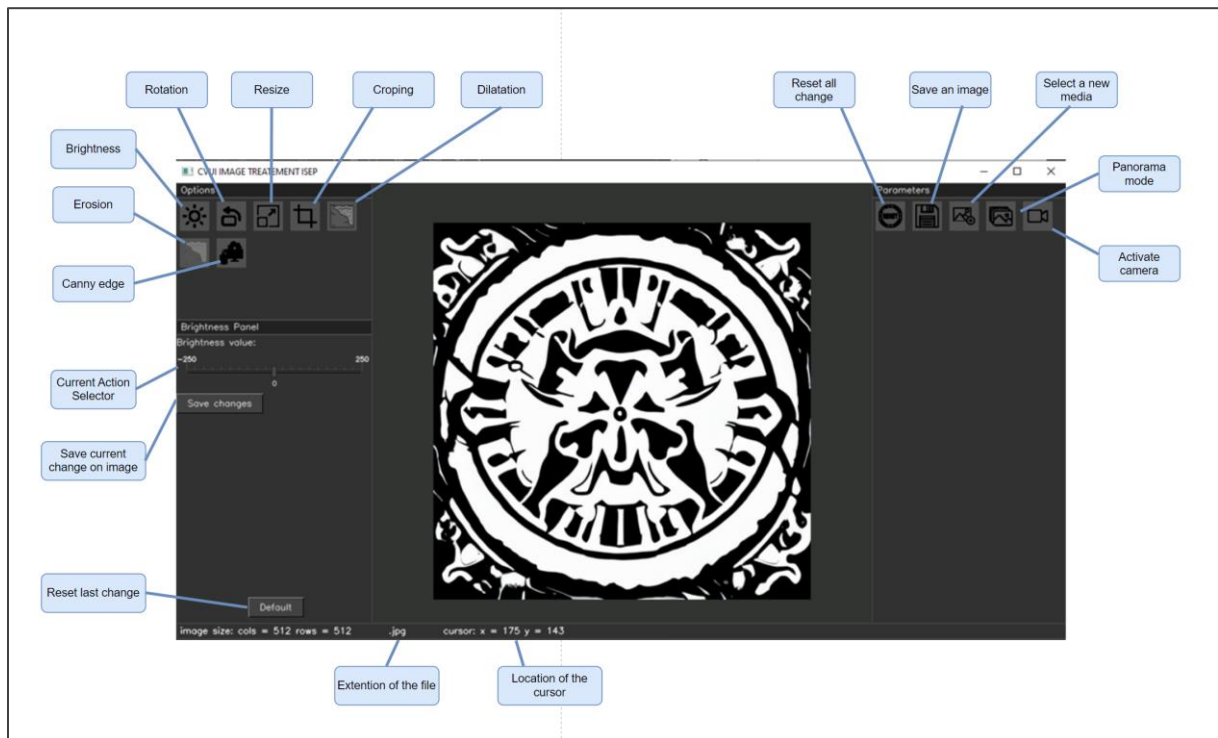
- OpenCV for image processing
- CVUI for the GUI Application
- Tinyfiledialogs to select the media file

## IV. Graphic Interface

### 1. Image

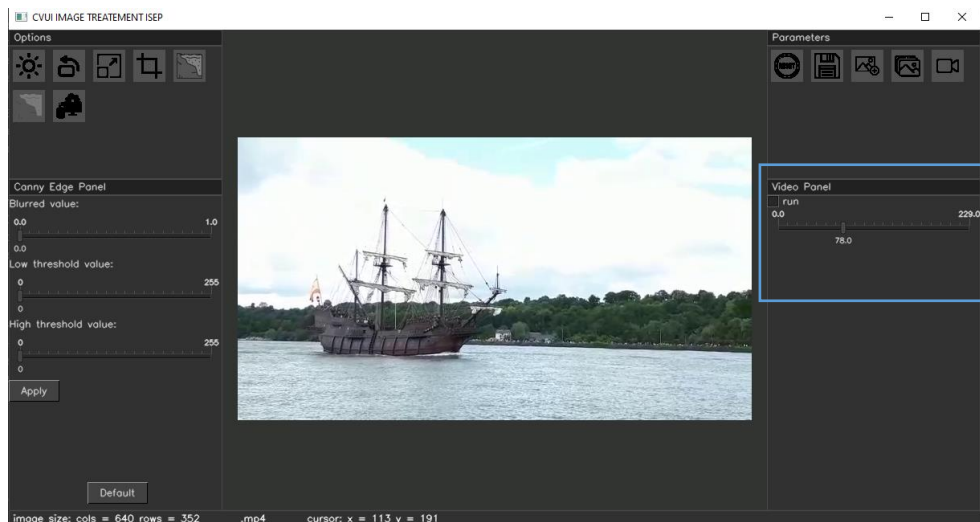
Our graphical interface is designed to be efficient and user-friendly. It is divided into three sections:

- ❖ The **Left Section** is dedicated to all image modification actions. It includes all the previously mentioned modification tools, as well as sliders for easy application of changes and a save button.
- ❖ The **Center Section** is the workspace where the current image being edited is displayed.
- ❖ The **Right Area** manages media-related tasks. It includes options for importing media and panoramas, activating the camera, saving the modified image to the device, and performing a total reset.



### 2. Video and camera

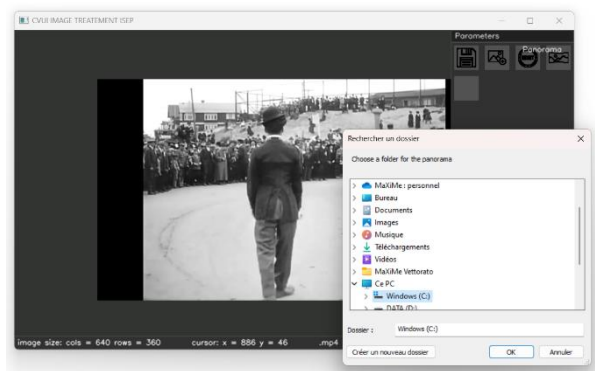
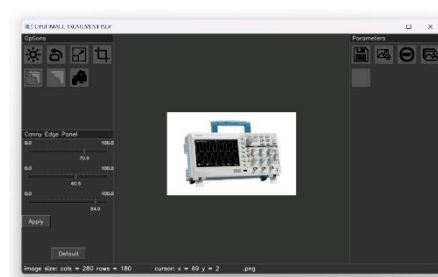
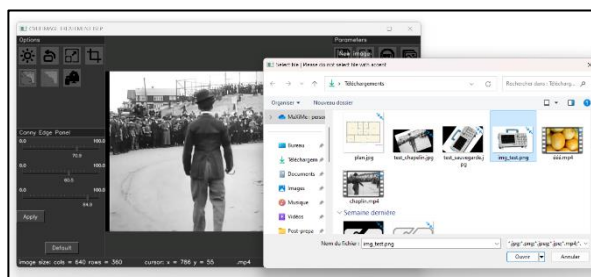
The application provides the capability to open either a video file or the camera. A dedicated section is incorporated in the **Bottom Right Area** to handle tasks related to video and camera. This includes features to pause the video and navigate through the frames of the video.



### 3. File Dialog

To complete our graphical interface, improve ergonomics, and create additional user comfort, we have added a file dialog system with the device's file browser. This system comes from the TinyFileDialogs library and is suitable for use on Windows and Linux.

This system allows for sending images or videos, saving modified images, and selecting directories containing images for stitching.



## V. Conclusion

The general distribution and organization of the task was as follows:

	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6
Charles	Git creation, computer set up, etc.	Dilatation, Erosion	Panorama	User interface base ImageApp	User interface merged with use of all functions (class: Image(), ImagePanorama(), ImageHandler())	User interface and merging of last functions (Open, save, image) + report
Maxime		Lighten, Darken		Search on filedialog library and implementation (window + linux)	Select media, save image	Select media, save image + report
Lucie		Rotation, resizing, cropping	Canny edge detection	Merging of all image functionalities in a class Image	Image vector, undo and redo	Report

Overall, we can say that the team collaborated effectively and faced few disagreements. The main challenges we encountered were related to working on **different operating systems (OS)**. The final application runs perfectly on Windows but is not totally Linux friendly due to three simple commands:

- Closing the application using the X uses an OpenCV command that does not work with Linux: even though the application is open, the `cv::getWindowProperty()` displays a -1 breaking the general loop and closing the application right after its opening. We have not managed to find an alternative for this function in Linux, meaning that the only way to make it work is to delete that condition and exist with the escape key
- The ctrl+Z key is defined as the ascii value 27 but in Linux, this corresponds to number 122
- The ctrl+Y key is defined as the ascii value 26 but in Linux, this corresponds to number 121

In order to bypass these issues, the code must include the detection of the OS and act differently depending on the case. Unfortunately, seeing we have encountered other issues in the creating of the Makefile for Linux, the Linux version is not available.

We are aware that with extended time, we might have added some more functionalities or deepen our work with like:

- Implementing all the changes on to a video instead of only images
- Adding the image recognition function
- Creating a functional application on Linux

But for the time being, we are satisfied with the functioning functionalities of our application.