Multimedia Application Final Project

Team members:

LUO Weisheng(10756)
ZHANG Kexin(61080)
YUAN Yao (62848)

Date :
11/06/2023

Table of Contents

What We Learned

Throughout this course, we have learned a variety of concepts and skills related to Multimedia Applications. The course focused on providing a comprehensive understanding of fundamental C++ programming principles and basic operations using OpenCV.

Firstly, we gained a solid foundation in C++ programming language, covering topics such as variables, data types, control structures (loops and conditionals), pointer and reference, functions, and object-oriented programming (OOP) concepts. We learned how to write efficient and structured code, enhancing our problem-solving abilities.

Moreover, we delved into the world of Multimedia Applications by exploring the capabilities of OpenCV. OpenCV is a powerful open-source library widely used for computer vision tasks and multimedia processing. We acquired knowledge about image and video manipulation, including loading, saving, resizing, and applying various image filters and transformations. Additionally, we grasped the concept of feature detection and extraction, as well as object tracking and recognition techniques.

During practical assignments and projects, we had the opportunity to apply our theoretical knowledge and develop hands-on experience. We developed diverse multimedia functions, such as image editing functions, video processing algorithms, and even simple computer vision applications like face detection and tracking.

In summary, this course equipped us with a strong understanding of C++ programming and its application in Multimedia Applications. We gained proficiency in utilizing the OpenCV library for image and video manipulation, as well as exploring computer vision concepts. This knowledge and experience will undoubtedly serve as a solid foundation for further exploration and development in the field of multimedia.

Explaining the Functions of OpenCV

Function name: MainWindow::MainWindow(QWidget *parent)

Function: Constructor, initializes the MainWindow object. Set the initial brightness level and initialize the timer, connect to the updateFrame slot function.

How it's exploited in the code: The initial brightness level is set to 5. Then create a new QTimer object and connect its timeout signal to the updateFrame slot function of the MainWindow class.

There are no OpenCV functions used in this piece of code, but it does use some classes and functions from the Qt library:

QTimer: It's a timer class. Here, a new QTimer object is created.

connect(): This is a Qt function that connects signals to slots. Here, it connects the timeout signal of the timer to the updateFrame slot function of MainWindow.

QTimer::timeout: This is a signal from QTimer. When the timer reaches the preset interval, this signal is emitted.

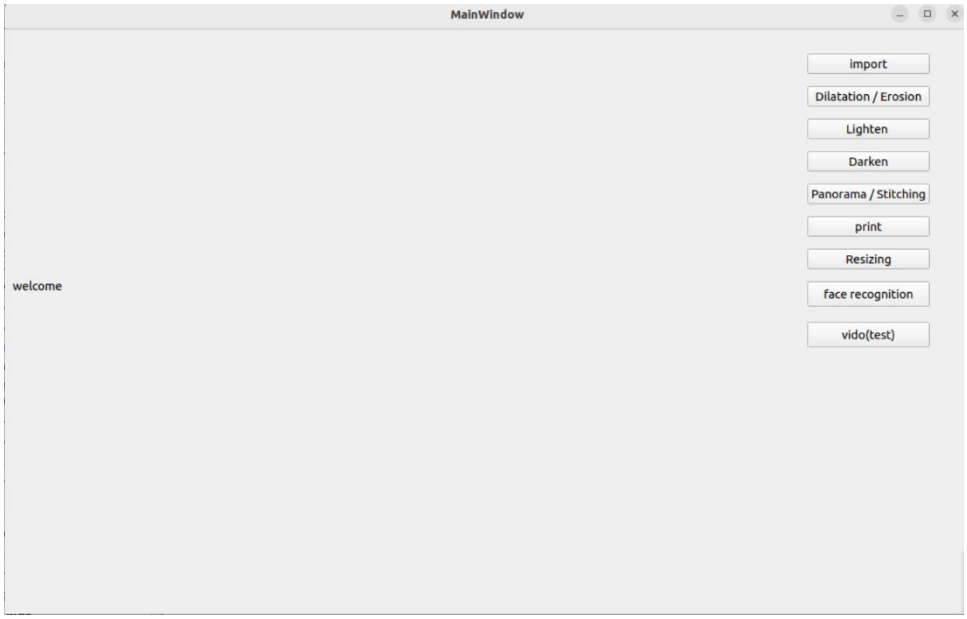

Figure 1 QT Interface

Function name: MainWindow::on_pushButton_showImage_clicked()

Function: This function is executed when the user clicks the "Show Image" button, and it is used to select, load, and display images.

How it's exploited in the code: First, QFileDialog::getOpenFileName is called to get the path of the image the user wants to open. Then, cv::imread is used to load the image into mCVimage. After that, cv::cvtColor is called to convert the image from BGR color space to RGB color space. Finally, the image from OpenCV is converted to a Qt image and displayed on the QLabel.

The OpenCV functions used include:

cv::imread: This is a function to read images. Here, it's used to load an image from the path selected by the user.

cv::cvtColor: This is a color conversion function. Here, it's used to convert the image from BGR color space to RGB color space.

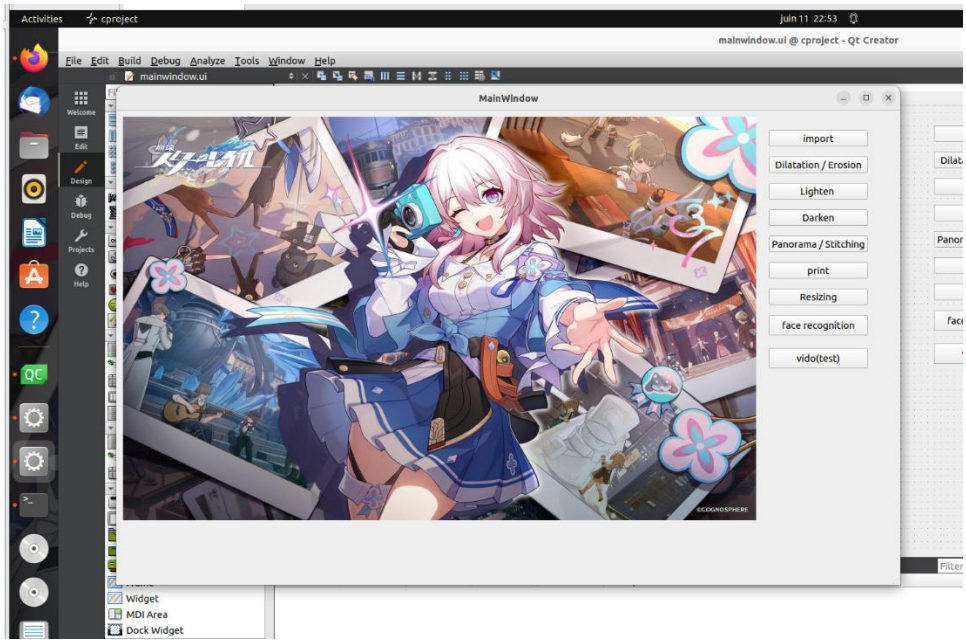

Figure 2Image Input

Function name: MainWindow::on_ Brighten _clicked()

Function: Handles the click event on the "liangdu" button in the user interface, the main task is to increase the brightness of the image.

How it's exploited in the code: First, check if there is an image to process. Then, check if the brightness level has reached the maximum value. If the conditions are met, increase the brightness level, adjust the brightness, and display the processed image on the interface.

Used OpenCV functions:

cv::Mat::empty(): Checks whether the matrix (here, the image) is empty. Here, if mCVimage is empty, it means there is no picture to process.

cv::Mat::operator+: Matrix addition operation, used to adjust image brightness. Here, it adds mCVimage with a cv::Scalar calculated from the brightness level to get the image with increased brightness.

cv::Scalar: In OpenCV, Scalar is used to represent a 4-element vector, commonly used to represent pixel color (BGR) or brightness. Here, it's used to adjust the image brightness.

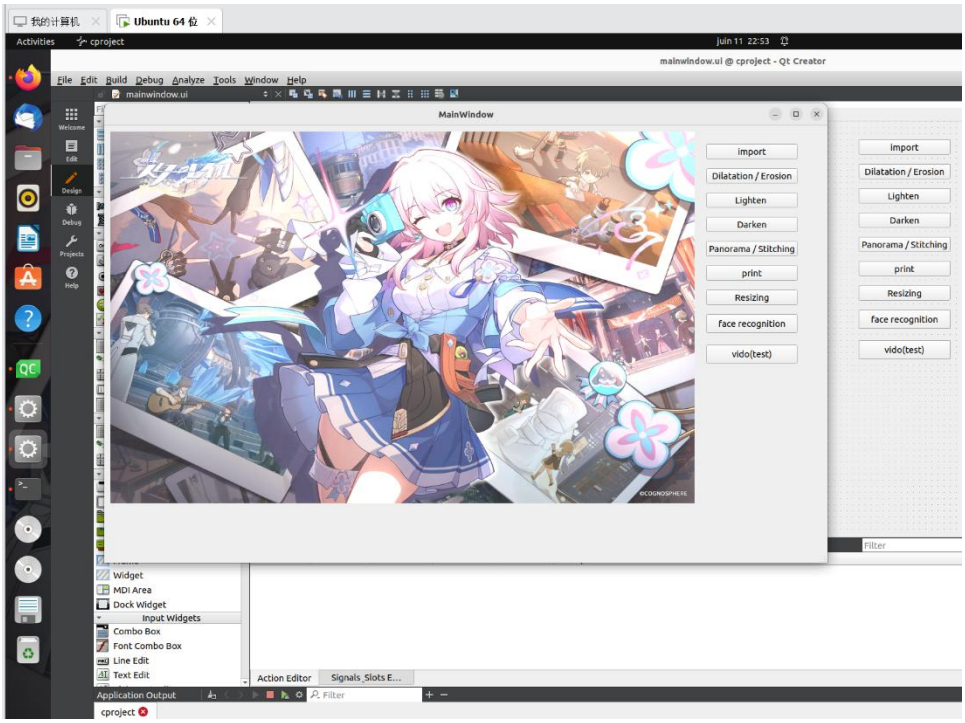

Figure 3 Brightness Increasing

Function name: MainWindow::on_ darken _clicked()

Function: Handles the click event on the "andu" button in the user interface, the main task is to decrease the brightness of the image.

How it's exploited in the code: First, check if there is an image to process. Then, check if the brightness level has reached the minimum value. If the conditions are met, decrease the brightness level, adjust the brightness, and display the processed image on the interface.

Used OpenCV functions:

cv::Mat::empty(): Checks whether the matrix (here, the image) is empty. Here, if mCVimage is empty, it means there is no picture to process.

cv::Mat::operator+: Matrix addition operation, used to adjust image brightness. Here, it adds mCVimage with a cv::Scalar calculated from the brightness level to get the image with decreased brightness.

cv::Scalar: In OpenCV, Scalar is used to represent a 4-element vector, commonly used to represent pixel color (BGR) or brightness. Here, it's used to adjust the image brightness.

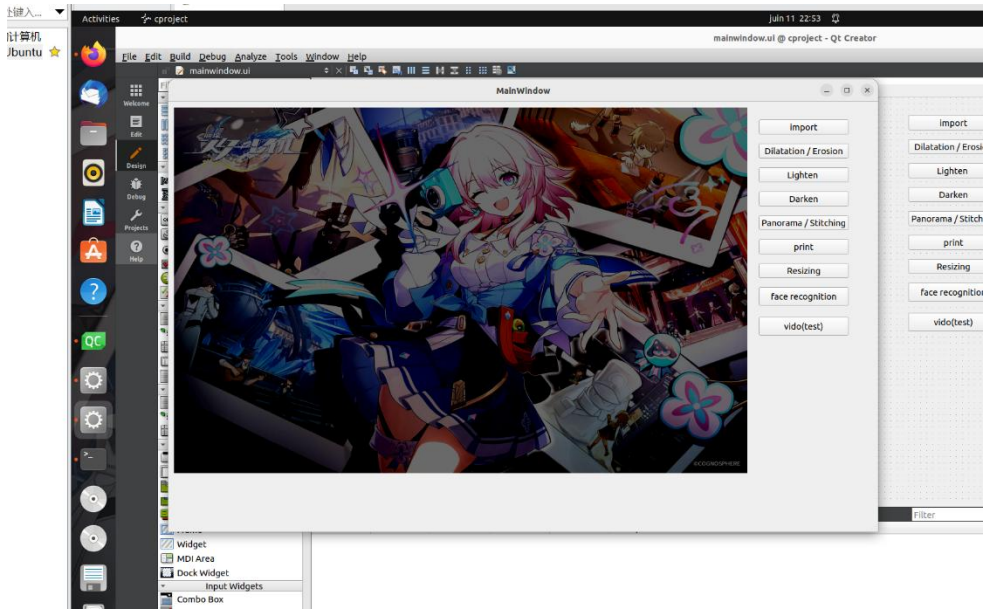

Figure 4 Brightness Decreasing

Function name: MainWindow::on_dilatation_clicked()

Function: This function is executed when the user clicks the "Dilatation" button, and it is used to perform a dilatation operation on the image to enhance the white or highlighted areas in the image.

How it's exploited in the code: First, a structuring element is created, which is a small matrix used to determine the nature of the dilatation operation. Then, the cv::dilate function is used to perform the dilatation operation on the image, producing a dilated image. Finally, the processed image is converted to QImage and displayed on QLabel.

The OpenCV functions used include:

cv::getStructuringElement: This is a function to create a structuring element. The structuring element is a small matrix that is used to determine the nature of morphological operations such as dilatation and erosion.

cv::dilate: This is a function for dilatation operation. It takes the original image and the structuring element as input, and then generates a dilated image.

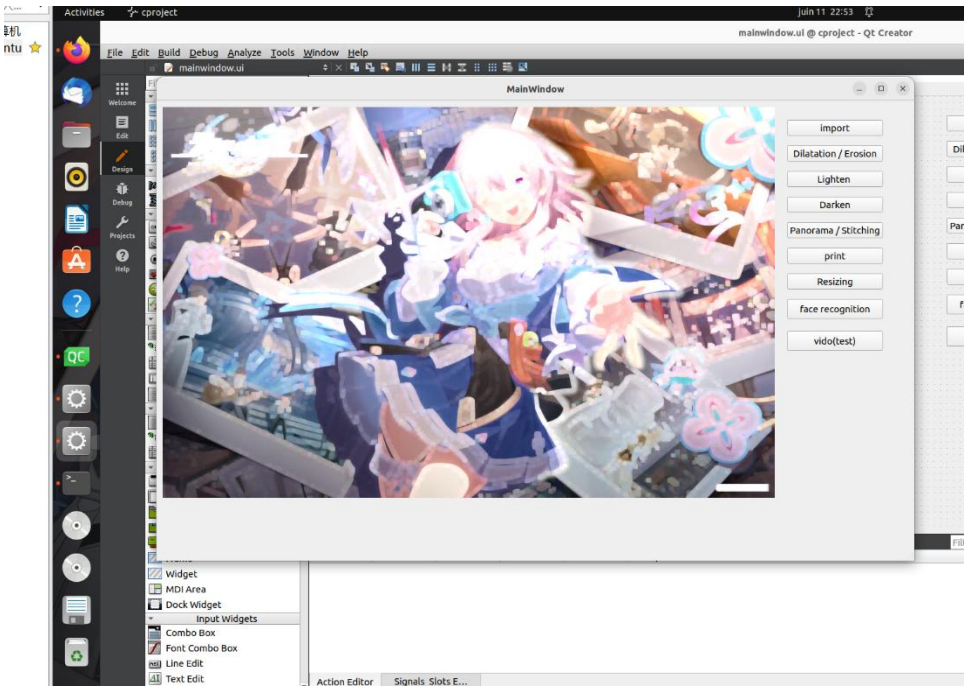

Figure 5 Dilatation

Function name: MainWindow::on_ mergeimages _clicked()

Function: This function is executed when the user clicks the "Merge" button, and it is used to horizontally concatenate two images.

How it's exploited in the code: First, the second image is loaded and read, and its color space is converted from BGR to RGB. Then, if the height of the first image and the second image are different, the height of the second image is adjusted to be the same as the first image. Next, the cv::hconcat function is used to horizontally concatenate the two images. Finally, the processed image is converted to QImage and displayed on QLabel, and it is saved as a file.

The OpenCV functions used include:

cv::imread: This is a function to read images. It can load image files from a specified path.

cv::cvtColor: This is a color space conversion function. It can convert an image from one color space to another, for example, from BGR to RGB.

cv::resize: This is a function to resize images. It can change the size of an image according to a specified scale.

cv::hconcat: This is an image merging function. It can horizontally concatenate two images of the same height.

cv::imwrite: This is a function to write images. It can save an image as a file at a specified path.

Figure 6 Panorama, Image Concatenation

Function name: MainWindow::on_print_clicked()

Function: This function is executed when the user clicks the "Save" button, and it is used to save the currently loaded image to the path specified by the user.

How it's exploited in the code: First, it checks whether mCVimage is empty. If it is empty, it means that no image has been loaded and there is no need to save. Then, it calls the QFileDialog::getSaveFileName() function to let the user choose the path to save the image. If the path is empty, the function returns; otherwise, it converts the color space of the image from RGB to BGR, and then calls the cv::imwrite() function to save the image to the path chosen by the user.

The OpenCV functions used include:

cv::cvtColor: This is a color space conversion function. It can convert an image from one color space to another, for example, from RGB to BGR.

cv::imwrite: This is a function to write images. It can save an image as a file at a specified path.

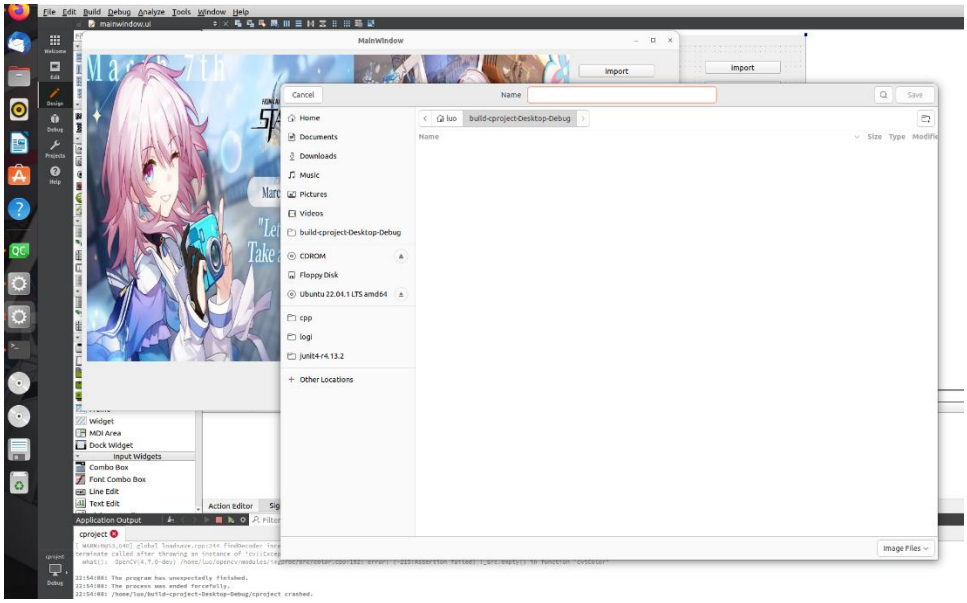

Figure 7 Image Output

Function name: MainWindow::on_ Stitching _clicked()

Function: This function is executed when the user clicks the "Change Size" button, and it is used to adjust the size of the currently loaded image to the size specified by the user.

How it's exploited in the code: First, it checks whether mCVimage is empty. If it is empty, it means that no image has been loaded and there is no need to adjust the size. Then, it calls the QInputDialog::getInt() function to let the user input the new width and height. If the user clicks "OK", it then calls the cv::resize() function to change the size of the image. After that, it converts the resized image into QImage and displays it on QLabel. Finally, it lets the user choose the path to save the image and saves the resized image.

The OpenCV functions used include:

cv::resize: This is a function to adjust the size of an image. It can resize the image to a specified size.

cv::cvtColor: This is a color space conversion function. It can convert an image from one color space to another, for example, from RGB to BGR.

cv::imwrite: This is a function to write images. It can save an image as a file at a specified path.

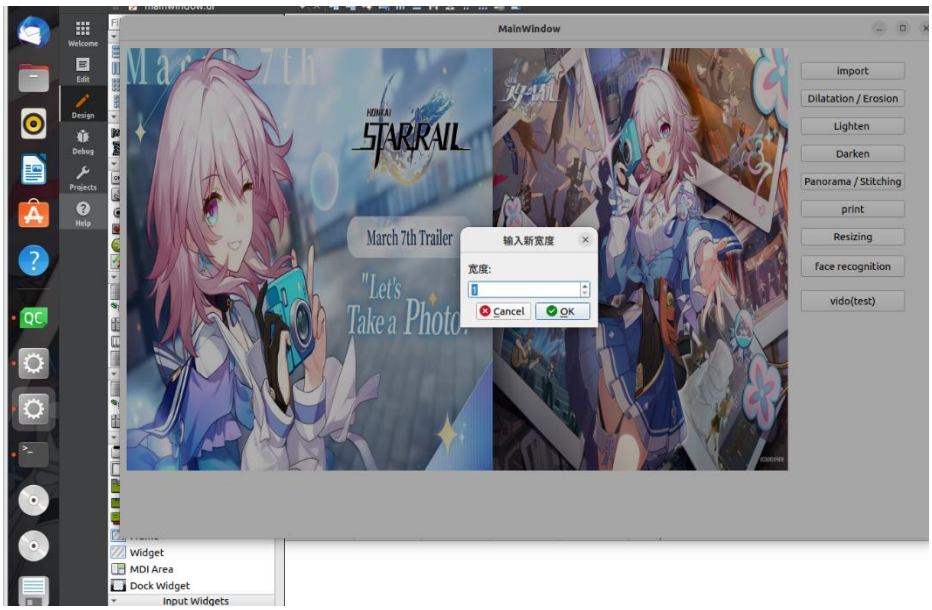

Figure 8 Image Resizing

Function name: MainWindow::on_ Detection_clicked()

Function: Handles the click event on the "renlian" button in the user interface. The main task is to detect faces in an image and draw rectangles around the detected faces.

How it's exploited in the code:

The code loads a face detection model using the faceDetector.load() function. If the model fails to load, an error message is displayed, and the function returns.

It checks if there is an image available to process. If the image is empty, a warning message is displayed, and the function returns.

The image is converted to grayscale using the cv::cvtColor() function with the conversion code cv::COLOR_BGR2GRAY.

The face detection algorithm is applied to the grayscale image using the faceDetector.detectMultiScale() function. Detected face regions are stored in the faces vector of rectangles.

A loop is used to iterate over the detected faces in the faces vector. For each face, a rectangle is drawn on the original image (mCVimage) using the cv::rectangle() function, with a blue color (represented by cv::Scalar(255, 0, 0)).

The processed image is converted to a QImage object, and it is displayed on a QLabel component in the user interface using ui->label->setPixmap(QPixmap::fromImage(image.rgbSwapped())).

Used OpenCV functions:

faceDetector.load(): This function loads a face detection model from the specified file path.

cv::cvtColor(): This function converts the image from one color space to another. In this code, it converts the image from BGR (color) to grayscale.

faceDetector.detectMultiScale(): This function applies the face detection algorithm to the grayscale image and detects multiple faces.

cv::rectangle(): This function draws rectangles on an image. In this code, it is used to draw rectangles around the detected faces.
QImage(): This constructor creates a QImage object from the image data, specifying the dimensions, data pointer, and pixel format.
ui->label->setPixmap(): This function sets the pixmap of a QLabel component, displaying the processed image on the user interface.

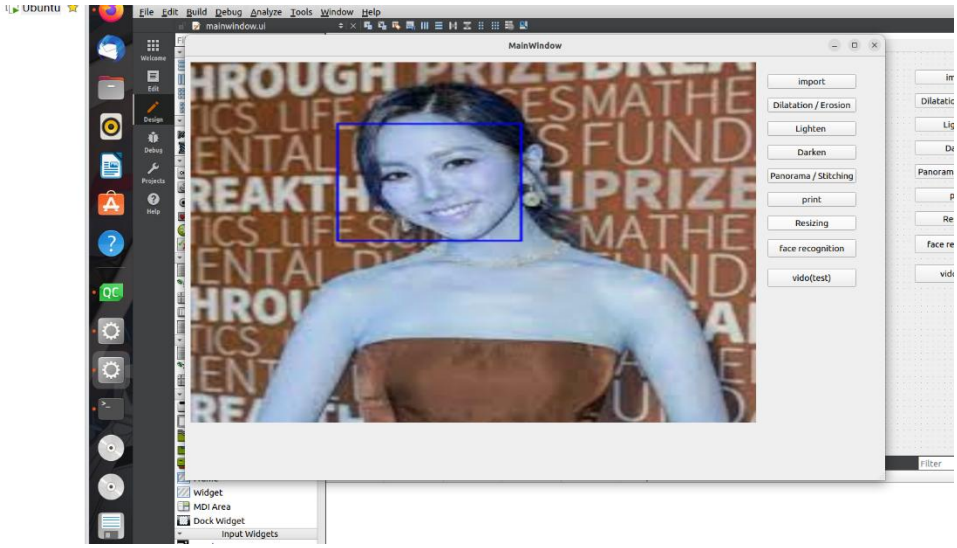

Figure 9 Face Detection and Recognition

Function name: clipVideo()

Function: Clips a video file by extracting a specified range of frames and saving them to a new video file.
How it's exploited in the code:
The function takes the input file path, output file path, start frame number, and end frame number as parameters.
It opens the input video file using cv::VideoCapture and checks if it is successfully opened. If not, an error message is displayed, and the function returns.
The function retrieves the frame width, frame height, and frames per second (fps) of the input video using the appropriate cap.get() functions.
It opens the output video file for writing using cv::VideoWriter with the MJPG codec, specified fps, and the same frame dimensions as the input video.
If the output video file fails to open, an error message is displayed, and the function returns.
Inside a loop, the function reads each frame from the input video using cap >> frame.
If the frame is empty, indicating the end of the video, the loop is terminated.
If the current frame number falls within the specified range (startFrame to endFrame, inclusive), the frame is written to the output video file using writer.write(frame).
The frame number is incremented.
Once all frames are processed, the input and output video resources are released by calling cap.release() and writer.release().

Used OpenCV functions:
cv::VideoCapture: This class is used to read video files or video streams.
cv::VideoCapture::isOpened(): This function checks if the video capture is successfully opened.
cv::VideoCapture::get(): This function retrieves properties of the video, such as frame width, frame height, and frames per second.
cv::VideoWriter: This class is used to write video files.
cv::VideoWriter::open(): This function opens the video writer for writing with the specified parameters, including the output file path, codec, fps, and frame size.
cv::Mat: This class represents a matrix (image or video frame) in OpenCV.
cv::VideoCapture::operator>>: This operator reads the next frame from the video capture.
cv::VideoWriter::write(): This function writes a frame to the video file.
cv::VideoCapture::release(): This function releases the video capture resources.
cv::VideoWriter::release(): This function releases the video writer resources.

Function name: MainWindow::on_pushButton_clicked()

Here is the overview of the functionality below.
Function: Handles the click event on the "pushButton" in the user interface. The main task is to select a video file, specify the start and end times (in seconds) for clipping the video, and call the clipVideo() function to perform the video clipping operation.
How it's exploited in the code:
The code prompts the user to select a video file using the QFileDialog::getOpenFileName() function and stores the selected file path in the videoFile variable. If no file is selected, the function returns.
It displays a dialog box to input the start time (in seconds) for video clipping using QInputDialog::getInt(). The input value is stored in the startSecond variable. If the user cancels the input, the function returns.
Similarly, it prompts the user to input the end time (in seconds) for video clipping and stores the value in the endSecond variable. If the user cancels the input, the function returns.
The code creates a cv::VideoCapture object cap by converting the videoFile to a standard string. It also retrieves the frames per second (fps) of the video using cap.get(cv::CAP_PROP_FPS).
The start frame and end frame for video clipping are calculated based on the input start and end times and the fps of the video.
A string outputFile is defined as "output.avi". You may need to modify this code to allow the user to choose the output file name.
Finally, the clipVideo() function is called with the input video file, output file, start frame, and end frame as parameters to perform the video clipping operation.Used Qt functions:
QFileDialog::getOpenFileName(): This function displays a dialog for selecting a file and returns the selected file path.
QInputDialog::getInt(): This function displays a dialog for inputting an integer value and returns the entered value.
QString::toStdString(): This function converts a QString to a standard string.
std::string: This is the standard string class in C++.
Used OpenCV functions:

cv::VideoCapture: This class is used to read video files or video streams.
cv::VideoCapture::get(): This function retrieves properties of the video, such as frames per second (fps).
clipVideo(): This is a user-defined function (not shown in the code snippet) that performs the actual video clipping operation.

Encapsulation in Application Development

The .pro file is a project file for the Qt framework. It specifies the project's source files, header files, UI forms, and necessary libraries (including OpenCV in this case). This file is a crucial component of the application package.
Your header file, mainwindow.h, defines the MainWindow class, which inherits from QMainWindow. QMainWindow is part of the Qt component system and provides a main application window for your GUI application. MainWindow also has some member functions (slots) that are connected to events in your GUI (such as clicking a button).
The mainwindow.cpp file will contain the implementation of the MainWindow class. This application follows the paradigm of object-oriented programming (OOP), which is a programming model that revolves around objects rather than "actions" and organizes logic around data. In the context of Qt and this application:
MainWindow is an object (an instance of a class). This object has a state (such as data members like liangdu_level, andu_level, camera) and behavior (such as member functions like on_pushButton_showImage_clicked, updateFrame).
Polymorphism: The MainWindow class extends the QMainWindow class, so it can use its functions. This is one of the key features of OOP called inheritance.
Encapsulation: By declaring some members as private, they cannot be accessed directly from outside the class. Instead, they must be accessed through member functions (getters and setters).

Overcoming Difficulties in Application Development

There is a problem with the erosion and dilation functions, which results in the feature only being adjustable once.

- The problems we encountered are as follows:
- There's an issue with the erosion and dilation functions, which results in the entire feature only being adjustable once.
- Because of the use of a virtual machine, my RTX3080 isn't working properly (the utilization of the CUDA cores in the GPU is abnormal), which leads to model training failure. I can only train a very small model with OpenCV 3, so I had to switch to the built-in face recognition library haarcascade_frontalface_default.xml of OpenCV 4.

- There is a bug in the video part. We used OpenCV's library, but it seems that the call failed. We tried to use the FFmpeg library instead, but it still didn't work. In the end, we could only put a button for an unimplemented function in the program. The commented code is our attempt to implement the video clipping function.4.There's an issue with image formats. When dealing with different formats in merging images, color distortion occurs. Sometimes, we even have to turn them into black and white for normal splicing.
- Due to lack of manpower, there's insufficient time to improve our code after implementing the basic functions.

Team Work

In our project, Luo Weisheng serves as the team leader, exhibiting outstanding leadership abilities and technical skills. He not only excels in the design and development of QT but also demonstrates proficiency in code integration, face recognition, and video processing. His responsibilities include project planning and allocating tasks reasonably among the team members, ensuring that each person can maximize their strengths. Under Luo Weisheng's leadership, our team's workflow is smooth and the project progress is satisfactory.
As a team member, Zhang Kexin focuses on the development of panorama/stitching functions, as well as adjusting image brightness/darkness. She is in charge of handling image input, providing more flexibility to our program by processing images of various formats. In class, Zhang Kexin actively participates in discussions, shares her thoughts, and maintains an open mind to suggestions from other team members.
Yuan Yao is the team member responsible for adjusting image sizes and eroding images. His work ensures users can easily adjust image sizes and perform other editing operations. Yuan Yao also handles image output, including saving the processed images. His role in the team is not only limited to accomplishing his tasks but also actively cooperating with other team members, providing constructive feedback to improve our project.
In class, our team has regular meetings to discuss the project's progress promptly, share ideas, and provide modification suggestions. Through the excellent leadership of Luo Weisheng and the team collaboration of Zhang Kexin and Yuan Yao, our project has made significant progress.

Tools used

Qt

Qt is a cross-platform application development framework that allows developers to create high-quality applications for desktop, mobile, embedded systems, and the web. It provides a comprehensive set of libraries and tools for building graphical user interfaces (GUIs), handling networking, managing databases, and more. Qt uses C++ as its primary programming language and provides extensive support for platform-specific features, as well as a wide range of APIs and widgets for creating visually appealing and responsive applications.

OpenCV 3

OpenCV (Open Source Computer Vision) is an open-source computer vision and machine learning library. OpenCV 3.x is a major version of the library that offers a wide range of functionalities for image and video processing, feature detection and tracking, object recognition, and more. It provides a comprehensive collection of algorithms and tools for computer vision tasks, including image filtering, geometric transformations, object detection, and machine learning-based techniques. OpenCV 3.x supports multiple programming languages, including C++, Python, and Java.

OpenCV 4

OpenCV 4.x is the next major version of the OpenCV library, which introduces several new features and improvements over the previous versions. It continues to provide a rich set of computer vision algorithms and tools, along with enhanced performance and functionality. OpenCV 4.x introduces support for new deep learning frameworks, such as TensorFlow and PyTorch, and includes new modules for advanced computer vision tasks, such as computational photography and 3D vision. It also brings optimizations for modern hardware architectures, improved APIs, and better integration with other libraries and frameworks. Like previous versions, OpenCV 4.x supports multiple programming languages.

Conclusion

Throughout this semester, we have learned a great deal and gained valuable knowledge and skills through our project work. By engaging in practical projects, we have gained a deep understanding of the process of multimedia application development and learned to effectively utilize various tools and libraries.

In terms of programming, we acquired a solid foundation in the C++ programming language. We mastered concepts such as variables, data types, control structures, pointers and references, functions, and object-oriented programming. This knowledge provided us with a strong basis for writing efficient and structured code, enhancing our problem-solving abilities.

In the domain of image and video processing, we learned to utilize the OpenCV library. OpenCV is a powerful open-source computer vision library that offers a wide range of image and video processing capabilities. We learned to load and save images, adjust image sizes, apply various image filters and transformations, and perform computer vision tasks such as feature detection, object tracking, and recognition.

Throughout the project, we had the opportunity to apply the knowledge we gained in practical scenarios. We developed various multimedia features, including image editing, video processing algorithms, and even implemented simple computer vision applications such as face detection and tracking. Through hands-on experience and debugging, we deepened our understanding of the multimedia application development process and accumulated practical skills.

We encountered several challenges during the project, including technical and resource-related difficulties. For example, we faced issues with certain OpenCV functions, which we addressed by carefully reviewing the code, consulting documentation, seeking assistance from peers and instructors, and conducting further research. Additionally, due to limitations in GPU utilization within the virtual machine environment, some of our model training attempts failed. To overcome this challenge, we switched to using the built-in face recognition library provided by OpenCV and adjusted the project goals and implementation approach accordingly.

In terms of teamwork, we had an exceptional team leader and actively engaged team members. The team leader played a crucial role in project planning and task allocation, ensuring that each team member could contribute their strengths. The team members maintained effective communication and collaboration, engaging in discussions, sharing ideas, and accepting and providing constructive feedback. Through team meetings and discussions, we ensured smooth project progress and timely completion.

Through this project, we not only acquired technical knowledge and skills but also enhanced our abilities to overcome challenges, drive project progress, and collaborate effectively in a team setting. We learned the importance of cooperation, communication, and collaboration in tackling difficulties and advancing projects. The valuable experiences gained during this project will provide a solid foundation for our future learning and development in the field of multimedia.