1. **Xây dựng chương trình**

**2.1. Mô tả chung về chương trình**

Chương trình xử lý ảnh này được xây dựng bằng Python, sử dụng các thư viện OpenCV (cv2) cho xử lý ảnh, PyQt5 cho giao diện đồ họa và Matplotlib cho hiển thị biểu đồ histogram. Chương trình cung cấp các chức năng xử lý ảnh cơ bản như cân bằng histogram, tách ngưỡng, lấy âm bản, biến đổi logarit, tăng độ tương phản, và các phép lọc không gian như lọc trung bình, trung vị, Gaussian, Bilateral và Non-Local Means.

**2.2. Cấu trúc chương trình**

**2.2.1 Giao diện người dùng (GUI)**

class ImageProcessorApp(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Image Processing Application")  
 self.setGeometry(100, 100, 1200, 600)  
   
 # Initialize variables  
 self.original\_image = None  
 self.processed\_image = None  
   
 # Setup UI  
 self.initUI()

* **Cửa sổ chính**: ImageProcessorApp kế thừa từ QMainWindow, cung cấp giao diện chính cho người dùng.
* **Layout**: Sử dụng QVBoxLayout và QHBoxLayout để sắp xếp các thành phần giao diện.
* **Nút bấm**: Các nút bấm được tạo ra để thực hiện các thao tác xử lý ảnh, kết nối với các hàm xử lý tương ứng.

**2.2.2 Các hàm xử lý ảnh**

**Tải ảnh**

def upload\_image(self):  
 file\_path, \_ = QFileDialog.getOpenFileName(self, "Open Image File", "", "Images (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if file\_path:  
 self.original\_image = cv2.imread(file\_path)  
 self.display\_image(self.original\_image, self.original\_label)

* **Tải ảnh**: upload\_image cho phép người dùng chọn và tải ảnh từ thiết bị.

**Hiển thị ảnh**

def display\_image(self, image, label):  
 if image is None:  
 return  
 # Convert image to RGB format for display in QLabel  
 rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 height, width, channel = rgb\_image.shape  
 bytes\_per\_line = channel \* width  
 qimage = QImage(rgb\_image.data, width, height, bytes\_per\_line, QImage.Format\_RGB888)  
 pixmap = QPixmap.fromImage(qimage)  
 label.setPixmap(pixmap.scaled(label.width(), label.height()))

* **Hiển thị ảnh**: display\_image chuyển đổi ảnh từ định dạng OpenCV sang định dạng QImage để hiển thị trên giao diện.

**Cân bằng histogram**

def histogram\_equalization(self):  
 if self.original\_image is None:  
 return  
 gray\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)  
 equalized\_image = cv2.equalizeHist(gray\_image)  
 self.processed\_image = equalized\_image  
 self.display\_image(cv2.cvtColor(equalized\_image, cv2.COLOR\_GRAY2RGB), self.processed\_label)  
   
 # Hiển thị histogram  
 plt.figure(figsize=(10, 5))  
 plt.subplot(1, 2, 1)  
 plt.hist(gray\_image.ravel(), 256, [0, 256])  
 plt.title('Histogram ban đầu')  
 plt.subplot(1, 2, 2)  
 plt.hist(equalized\_image.ravel(), 256, [0, 256])  
 plt.title('Histogram sau cân bằng')  
 plt.show()

* **Cân bằng histogram**: histogram\_equalization sử dụng cv2.equalizeHist để cân bằng histogram của ảnh xám.

**Tách ngưỡng**

def thresholding(self):  
 if self.original\_image is None:  
 return  
 # Convert to grayscale  
 gray\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)  
 # Apply a binary threshold  
 \_, thresholded\_image = cv2.threshold(gray\_image, 127, 255, cv2.THRESH\_BINARY)  
 self.processed\_image = thresholded\_image  
 # Display the processed image  
 self.display\_image(cv2.cvtColor(thresholded\_image, cv2.COLOR\_GRAY2RGB), self.processed\_label)

* **Tách ngưỡng**: thresholding áp dụng ngưỡng nhị phân với giá trị ngưỡng cố định.

**Lấy âm bản**

def negative\_image(self):  
 if self.original\_image is None:  
 return  
 # Create a negative of the image  
 negative\_image = cv2.bitwise\_not(self.original\_image)  
 self.processed\_image = negative\_image  
 # Display the processed image  
 self.display\_image(negative\_image, self.processed\_label)

* **Lấy âm bản**: negative\_image sử dụng cv2.bitwise\_not để đảo ngược giá trị pixel.

**Biến đổi logarit**

def log\_transform(self):  
 if self.original\_image is None:  
 return  
 # Convert to grayscale  
 gray\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)  
 # Apply log transform  
 c = 255 / (np.log(1 + np.max(gray\_image)))  
 log\_transformed = c \* (np.log(1 + gray\_image.astype(np.float32)))  
 # Normalize to 8-bit range  
 log\_transformed = np.uint8(log\_transformed)  
 self.processed\_image = log\_transformed  
 # Display the processed image  
 self.display\_image(cv2.cvtColor(log\_transformed, cv2.COLOR\_GRAY2RGB), self.processed\_label)

* **Biến đổi logarit**: log\_transform áp dụng phép biến đổi logarit để nén dải động của ảnh.

**Tăng độ tương phản**

def increase\_contrast(self):  
 if self.original\_image is None:  
 return  
 # Convert to YUV color space  
 yuv\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2YUV)  
 # Equalize the histogram of the Y channel  
 yuv\_image[:, :, 0] = cv2.equalizeHist(yuv\_image[:, :, 0])  
 # Convert back to BGR color space  
 contrast\_image = cv2.cvtColor(yuv\_image, cv2.COLOR\_YUV2BGR)  
 self.processed\_image = contrast\_image  
 # Display the processed image  
 self.display\_image(contrast\_image, self.processed\_label)

* **Tăng độ tương phản**: increase\_contrast sử dụng không gian màu YUV để tăng cường độ tương phản.

**Lọc trung bình**

def mean\_filter(self):  
 if self.original\_image is None:  
 return  
 # Apply mean filter  
 mean\_filtered\_image = cv2.blur(self.original\_image, (5, 5))  
 self.processed\_image = mean\_filtered\_image  
 # Display the processed image  
 self.display\_image(mean\_filtered\_image, self.processed\_label)

* **Lọc trung bình**: mean\_filter áp dụng lọc trung bình với kernel 5x5.

**Lọc trung vị**

def median\_filter(self):  
 if self.original\_image is None:  
 return  
 # Apply median filter  
 median\_filtered\_image = cv2.medianBlur(self.original\_image, 5)  
 self.processed\_image = median\_filtered\_image  
 # Display the processed image  
 self.display\_image(median\_filtered\_image, self.processed\_label)

* **Lọc trung vị**: median\_filter sử dụng cv2.medianBlur với kích thước kernel 5.

**Lọc Gaussian**

def gaussian\_filter(self):  
 if self.original\_image is None:  
 return  
 # Apply Gaussian filter  
 gaussian\_filtered\_image = cv2.GaussianBlur(self.original\_image, (5, 5), 0)  
 self.processed\_image = gaussian\_filtered\_image  
 # Display the processed image  
 self.display\_image(gaussian\_filtered\_image, self.processed\_label)

* **Lọc Gaussian**: gaussian\_filter áp dụng lọc Gaussian với kernel 5x5.

**Lọc Bilateral**

def bilateral\_filter(self):  
 if self.original\_image is None:  
 return  
 # Apply bilateral filter  
 bilateral\_filtered\_image = cv2.bilateralFilter(self.original\_image, 9, 75, 75)  
 self.processed\_image = bilateral\_filtered\_image  
 # Display the processed image  
 self.display\_image(bilateral\_filtered\_image, self.processed\_label)

* **Lọc Bilateral**: bilateral\_filter sử dụng cv2.bilateralFilter với các tham số điều chỉnh.

**Lọc Non-Local Means**

def non\_local\_means\_filter(self):  
 if self.original\_image is None:  
 return  
 # Apply NonLocalMeans filter  
 non\_local\_means\_filtered\_image = cv2.fastNlMeansDenoisingColored(self.original\_image, None, 10, 10, 7, 21)  
 self.processed\_image = non\_local\_means\_filtered\_image  
 # Display the processed image  
 self.display\_image(non\_local\_means\_filtered\_image, self.processed\_label)

* **Lọc Non-Local Means**: non\_local\_means\_filter áp dụng lọc Non-Local Means để khử nhiễu.

**2.3. Quy trình xử lý ảnh**

1. **Tải ảnh**: Người dùng chọn ảnh từ thiết bị.
2. **Hiển thị ảnh gốc**: Ảnh được hiển thị trên giao diện.
3. **Chọn phương pháp xử lý**: Người dùng chọn phương pháp xử lý từ các nút bấm.
4. **Xử lý ảnh**: Hàm tương ứng được gọi để thực hiện xử lý.
5. **Hiển thị kết quả**: Ảnh đã xử lý được hiển thị bên cạnh ảnh gốc.

**2.4. Kết quả và minh họa**

* **Histogram Equalization**: Cân bằng histogram giúp tăng độ tương phản toàn cục của ảnh.
* **Thresholding**: Tách ngưỡng giúp phân đoạn ảnh thành hai vùng rõ rệt.
* **Negative Image**: Tạo ra ảnh âm bản, đảo ngược giá trị pixel.
* **Log Transform**: Nén dải động của ảnh, làm sáng các vùng tối.
* **Increase Contrast**: Tăng cường độ tương phản bằng cách cân bằng histogram của kênh Y.
* **Mean Filter**: Làm mờ ảnh, giảm nhiễu nhưng làm mờ biên.
* **Median Filter**: Loại bỏ nhiễu xung mà không làm mờ biên.
* **Gaussian Filter**: Làm mờ ảnh với kernel Gaussian, giữ biên mềm mại.
* **Bilateral Filter**: Khử nhiễu nhưng giữ biên rõ nét.
* **Non-Local Means Filter**: Khử nhiễu hiệu quả, giữ texture phức tạp.

**2.5. Kết luận**

Chương trình này cung cấp một nền tảng để thực hiện các thao tác xử lý ảnh cơ bản và nâng cao, giúp người dùng hiểu rõ hơn về các kỹ thuật xử lý ảnh và ứng dụng chúng trong thực tế. Việc xây dựng chương trình này không chỉ giúp nâng cao kỹ năng lập trình mà còn mở rộng kiến thức về xử lý ảnh số.