THE UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

Project Title: Sleek Image Processing Application using PyQt5 and OpenCV

Submitted To:

Dr. Saba Awan

Submitted By:

Ahsan Khizar	**22-SE-51**
Muhammad Sohaib	**22-SE-67**

Course:

Digital Image Processing Lab

Date:

14/05/2025

1. Introduction

This semester's project aimed to build a sleek, fully interactive image processing application using **Python**, **PyQt5**, and **OpenCV**. The project combines the theoretical knowledge of image processing algorithms with practical GUI design to allow end-users to apply real-time filters, enhancements, and drawing operations on images. The app is intuitive, lightweight, and responsive, offering both static image editing and live camera-based preview.

2. Objectives

- To bridge theoretical image processing concepts with firsthand implementation.
- To develop an aesthetically clean and modern GUI.
- To support real-time image editing with features like grayscale, blur, and edge detection.
- To include brush drawing functionality with color and size customization.
- To support undo, live webcam preview, and drag-and-drop image loading.
- To provide brightness and contrast adjustments through sliders.
- To add theme toggling between light and dark modes for user experience.

3. Methodology

The development process followed a modular and iterative approach:

- **Frontend Design:** Using **PyQt5**, all GUI components such as buttons, sliders, toolbars, and labels were laid out in a structured and visually appealing manner.
- **Backend Logic:** Core image processing functionalities were implemented using **OpenCV**, including grayscale conversion, Gaussian blur, edge detection, and brightness/contrast adjustments.
- **Event Handling:** PyQt5 event-driven programming handled mouse drawing, slider movements, color picking, and undo operations.
- **Live Preview:** OpenCV's video capture was integrated for real-time camera preview and frame-based processing.
- **Drawing Mechanism:** Mouse events combined with OpenCV drawing functions allowed the user to paint on images, with a brush size and color picker integrated.

- **Drag and Drop:** Added drag-and-drop support for quick image loading from the user's file system.

4. Features Implemented

Feature	Description
Image Loading/Saving	Load and save images in .png, .jpg, .bmp formats
Undo Functionality	Stepwise undo for image operations
Grayscale	Convert images to grayscale
Gaussian Blur	Apply customizable blur effect using a slider
Edge Detection	Use the Canny method for detecting edges
Brightness & Contrast	Controlled via sliders for dynamic adjustments
Drawing Mode	Freehand drawing on images with adjustable brush size and color
Color Picker	QColorDialog used to change drawing color
Live Camera Preview	Real-time feed using OpenCV’s VideoCapture
Theme Toggle	Switch between light and dark UI modes
Drag and Drop	Load image files directly by dragging them into the app

5. Challenges & Solutions

Challenge	Solution
Live preview interfering with UI layout	Adjusted layout resizing behavior and added constraints
Brush drawing alignment issues	Mapped mouse positions relative to label using proper scaling factors
Theme switching bugs	Used unified stylesheet toggling and ensured UI consistency
Undo not synchronizing with live preview	Prevented history updates during frame-based updates from the webcam
Multiple sliders affecting image quality	Ensured all image modifications use the original image as a base for recalculations

6. Results

The application was successfully developed and tested with a variety of images and webcam inputs. Below are some observed outcomes:

- **Responsive UI**: The GUI responded smoothly to all user inputs, including sliders, drawing, and file handling.
- **Accurate Filter Effects**: Grayscale, Gaussian blur, and edge detection were applied in real time with noticeable, clear transformations.
- **Drawing Tool**: The drawing functionality with live color picking and adjustable brush size worked seamlessly on both static and live images.
- **Real-Time Camera Integration**: The application smoothly captured and processed frames from the webcam, allowing real-time editing previews.
- **Undo & Theme Toggle**: Undo operations performed correctly with no lag or crashes, and theme switching updated all components dynamically.

7. Conclusion

The project successfully met all the outlined objectives and provided a smooth, interactive platform to apply image processing concepts in real-time. It serves as a strong demonstration of integrating **PyQt5 for GUI** and **OpenCV for image processing**, while also refining concepts such as event handling, real-time preview, and UI/UX design. The application not only reinforces foundational learning from the Digital Image Processing course but also opens further possibilities for development.

8. Future Work

While the current version is feature-rich and user-friendly, the application can be extended with the following ideas:

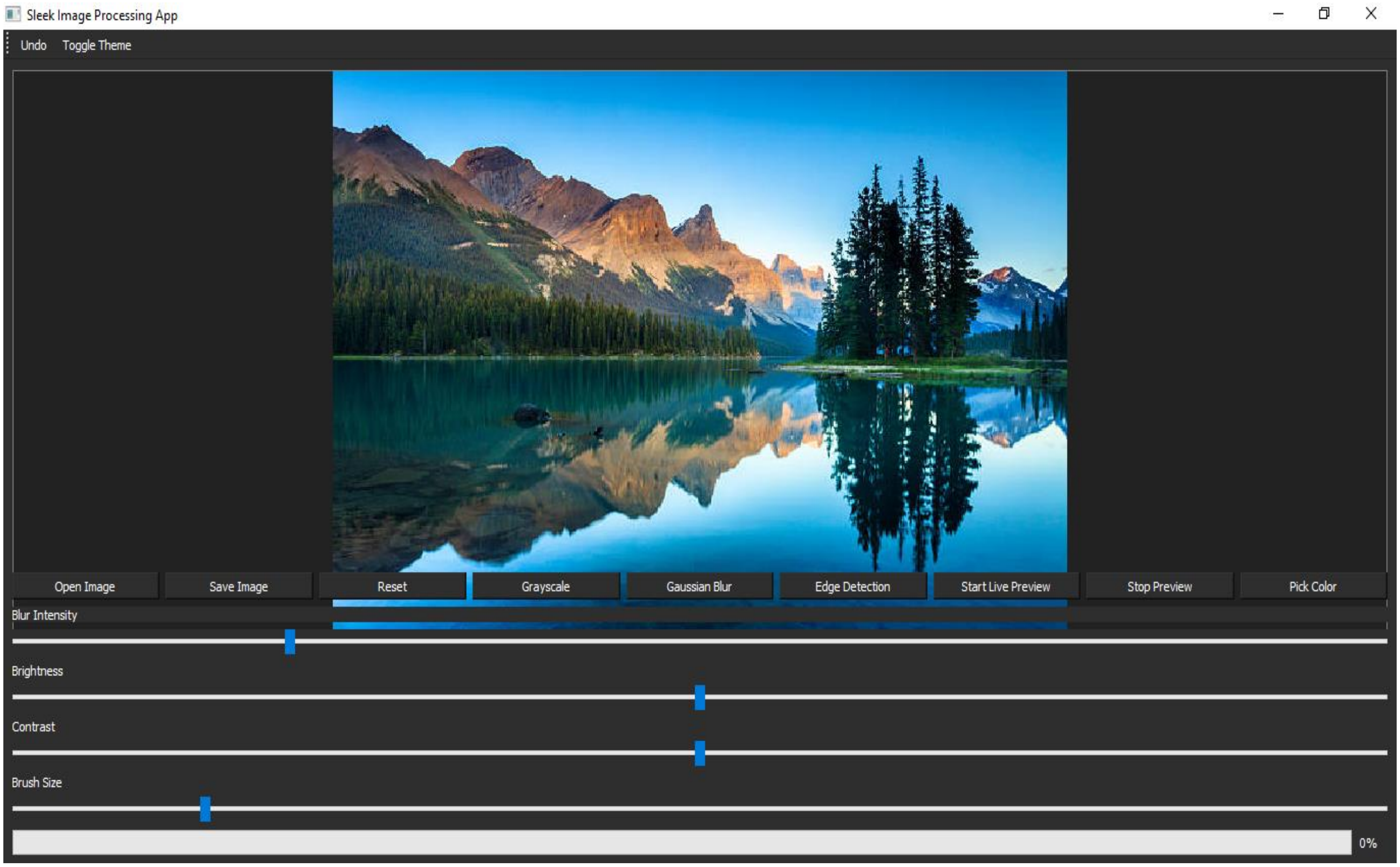
- **Image Transformation Tools:** Rotate, crop, scale, and mirror images.
- **Histogram Equalization:** For better contrast enhancement.
- **Advanced Filters:** Add options like median blur, sharpening, sepia, etc.

- **AI-based Filters:** Use ML models to apply style transfer or face detection.
- **Layered Drawing Canvas:** Enable layered editing for more complex compositions.
- **Export as PDF or Collage Maker**: Combine multiple edited images into a single layout.

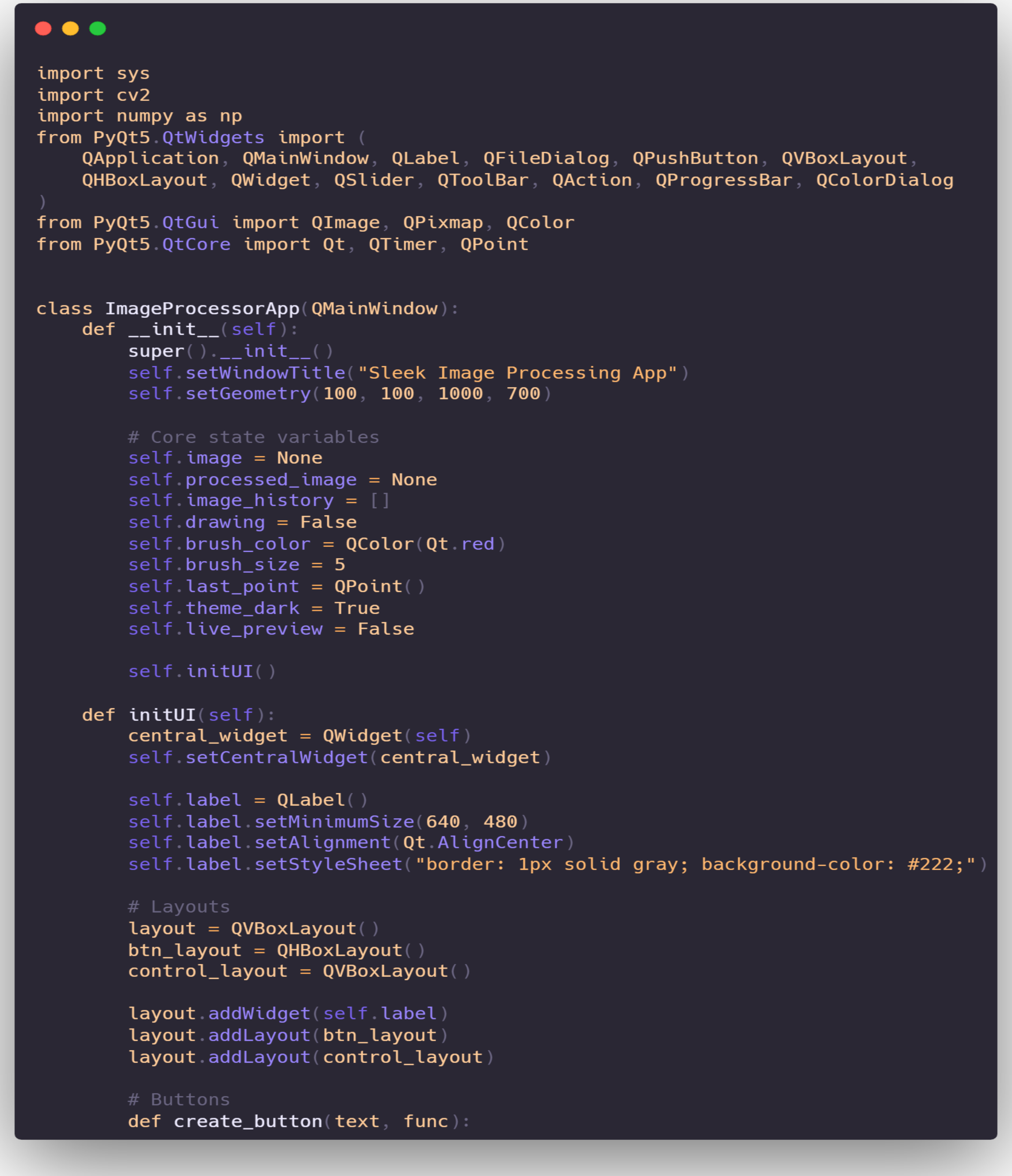
9. References

- OpenCV Documentation: https://docs.opencv.org/
- PyQt5 Documentation: https://doc.qt.io/qtforpython/
- NumPy Documentation: https://numpy.org/doc
- Canny Edge Detector: J. Canny, “A Computational Approach to Edge Detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986.
- Qt Designer Tools and Tutorials
- Stack Overflow and GitHub Discussions on PyQt5 and OpenCV Integration

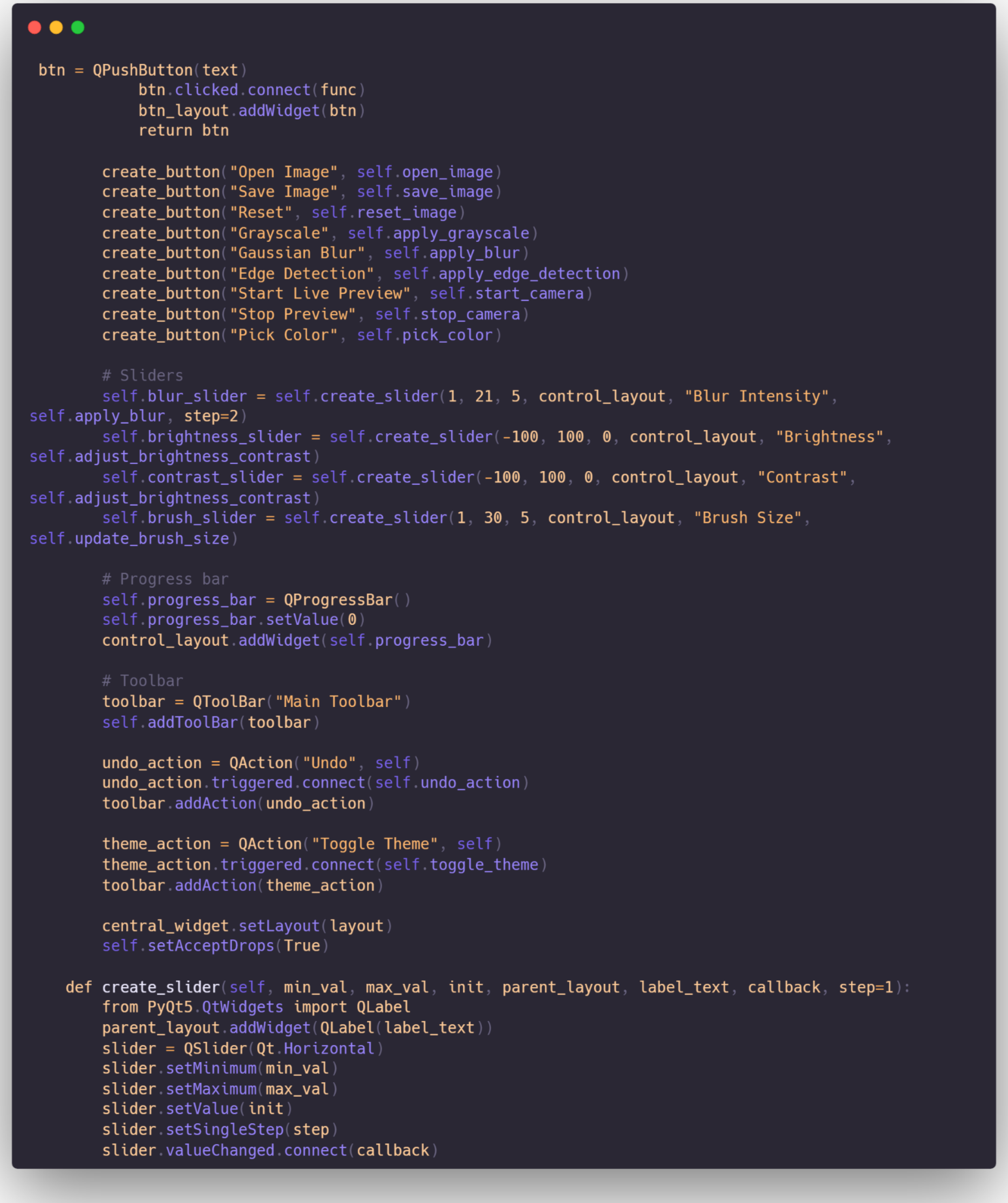
10. UI Screenshot



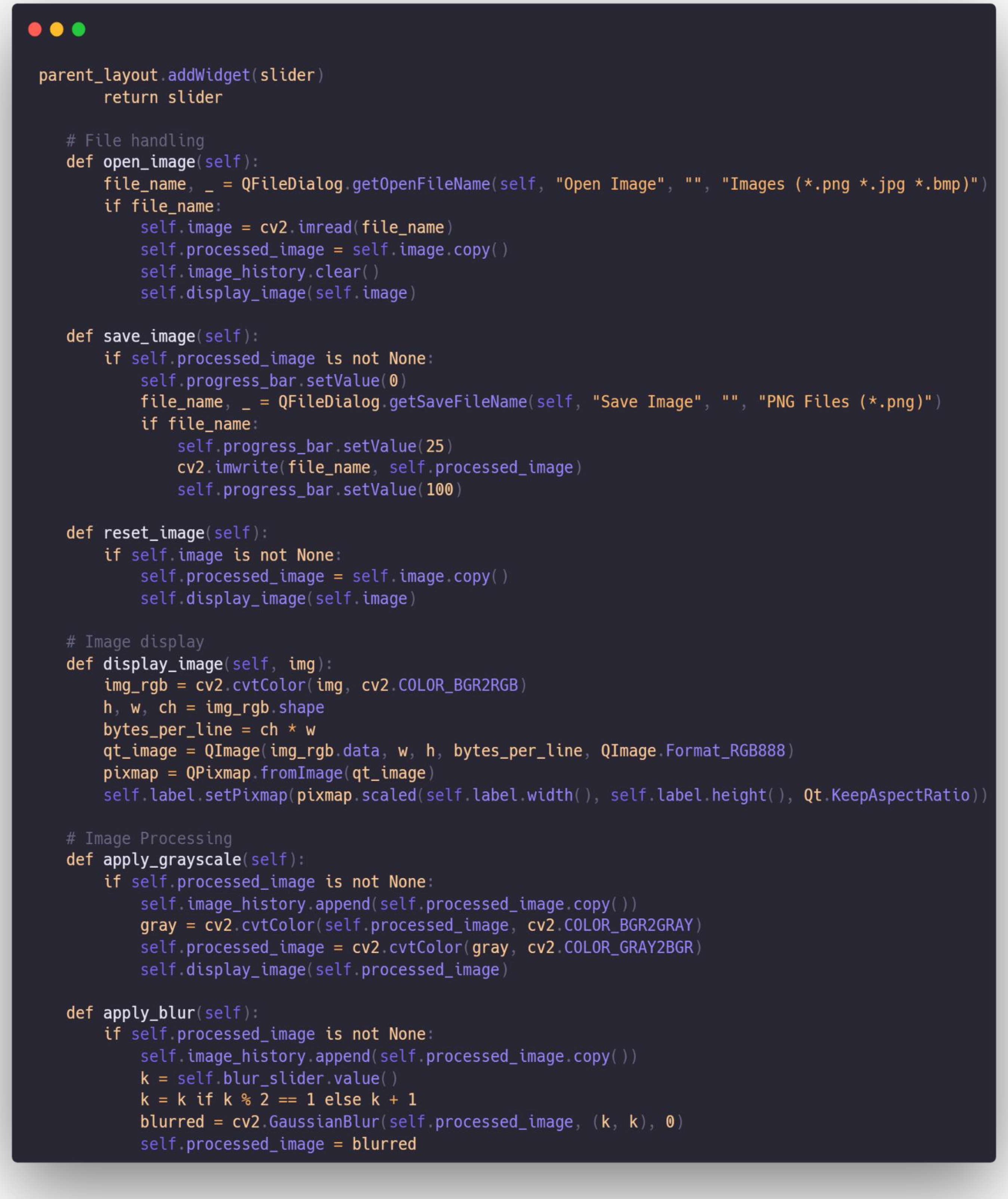
Appendix A – Full Source Code



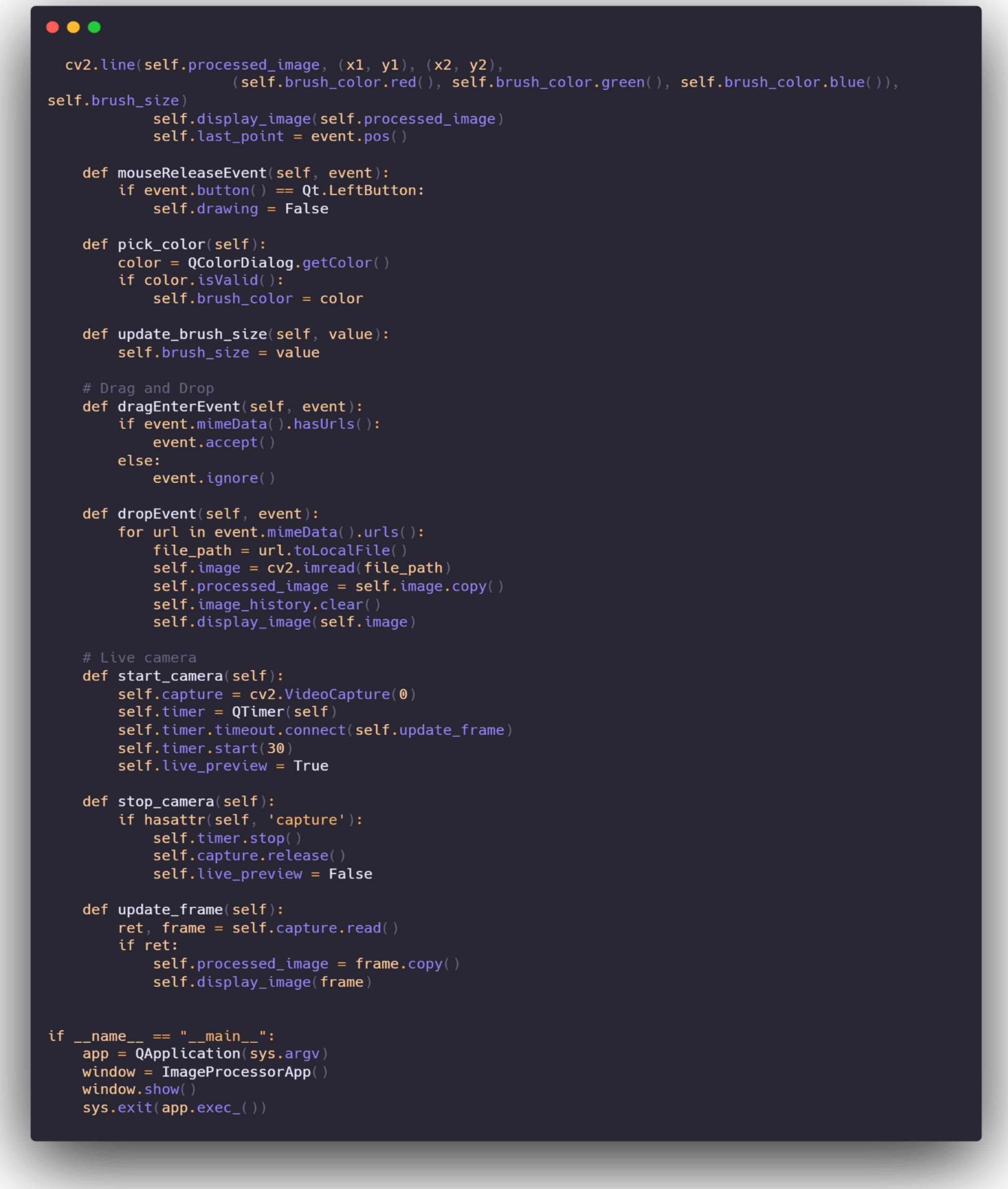
```
import sys
import cv2
import numpy as np
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QLabel, QFileDialog, QPushButton, QVBoxLayout,
    QHBoxLayout, QWidget, QSlider, QToolBar, QAction, QProgressBar, QColorDialog
)
from PyQt5.QtGui import QImage, QPixmap, QColor
from PyQt5.QtCore import Qt, QTimer, QPoint


class ImageProcessorApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Sleek Image Processing App")
        self.setGeometry(100, 100, 1000, 700)

        # Core state variables
        self.image = None
        self.processed_image = None
        self.image_history = []
        self.drawing = False
        self.brush_color = QColor(Qt.red)
        self.brush_size = 5
        self.last_point = QPoint()
        self.theme_dark = True
        self.live_preview = False

        self.initUI()

    def initUI(self):
        central_widget = QWidget(self)
        self.setCentralWidget(central_widget)

        self.label = QLabel()
        self.label.setMinimumSize(640, 480)
        self.label.setAlignment(Qt.AlignCenter)
        self.label.setStyleSheet("border: 1px solid gray; background-color: #222;")

        # Layouts
        layout = QVBoxLayout()
        btn_layout = QHBoxLayout()
        control_layout = QVBoxLayout()

        layout.addWidget(self.label)
        layout.addLayout(btn_layout)
        layout.addLayout(control_layout)

        # Buttons
        def create_button(text, func):
```



```
 btn = QPushButton(text)
            btn.clicked.connect(func)
            btn_layout.addWidget(btn)
            return btn

        create_button("Open Image", self.open_image)
        create_button("Save Image", self.save_image)
        create_button("Reset", self.reset_image)
        create_button("Grayscale", self.apply_grayscale)
        create_button("Gaussian Blur", self.apply_blur)
        create_button("Edge Detection", self.apply_edge_detection)
        create_button("Start Live Preview", self.start_camera)
        create_button("Stop Preview", self.stop_camera)
        create_button("Pick Color", self.pick_color)

        # Sliders
        self.blur_slider = self.create_slider(1, 21, 5, control_layout, "Blur Intensity",
self.apply_blur, step=2)
        self.brightness_slider = self.create_slider(-100, 100, 0, control_layout, "Brightness",
self.adjust_brightness_contrast)
        self.contrast_slider = self.create_slider(-100, 100, 0, control_layout, "Contrast",
self.adjust_brightness_contrast)
        self.brush_slider = self.create_slider(1, 30, 5, control_layout, "Brush Size",
self.update_brush_size)

        # Progress bar
        self.progress_bar = QProgressBar()
        self.progress_bar.setValue(0)
        control_layout.addWidget(self.progress_bar)

        # Toolbar
        toolbar = QToolBar("Main Toolbar")
        self.addToolBar(toolbar)

        undo_action = QAction("Undo", self)
        undo_action.triggered.connect(self.undo_action)
        toolbar.addAction(undo_action)

        theme_action = QAction("Toggle Theme", self)
        theme_action.triggered.connect(self.toggle_theme)
        toolbar.addAction(theme_action)

        central_widget.setLayout(layout)
        self.setAcceptDrops(True)

    def create_slider(self, min_val, max_val, init, parent_layout, label_text, callback, step=1):
        from PyQt5.QtWidgets import QLabel
        parent_layout.addWidget(QLabel(label_text))
        slider = QSlider(Qt.Horizontal)
        slider.setMinimum(min_val)
        slider.setMaximum(max_val)
        slider.setValue(init)
        slider.setSingleStep(step)
        slider.valueChanged.connect(callback)
```



```
parent_layout.addWidget(slider)
       return slider

   # File handling
   def open_image(self):
       file_name, _ = QFileDialog.getOpenFileName(self, "Open Image", "", "Images (*.png *.jpg *.bmp)")
       if file_name:
           self.image = cv2.imread(file_name)
           self.processed_image = self.image.copy()
           self.image_history.clear()
           self.display_image(self.image)

   def save_image(self):
       if self.processed_image is not None:
           self.progress_bar.setValue(0)
           file_name, _ = QFileDialog.getSaveFileName(self, "Save Image", "", "PNG Files (*.png)")
           if file_name:
               self.progress_bar.setValue(25)
               cv2.imwrite(file_name, self.processed_image)
               self.progress_bar.setValue(100)

   def reset_image(self):
       if self.image is not None:
           self.processed_image = self.image.copy()
           self.display_image(self.image)

   # Image display
   def display_image(self, img):
       img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
       h, w, ch = img_rgb.shape
       bytes_per_line = ch * w
       qt_image = QImage(img_rgb.data, w, h, bytes_per_line, QImage.Format_RGB888)
       pixmap = QPixmap.fromImage(qt_image)
       self.label.setPixmap(pixmap.scaled(self.label.width(), self.label.height(), Qt.KeepAspectRatio))

   # Image Processing
   def apply_grayscale(self):
       if self.processed_image is not None:
           self.image_history.append(self.processed_image.copy())
           gray = cv2.cvtColor(self.processed_image, cv2.COLOR_BGR2GRAY)
           self.processed_image = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
           self.display_image(self.processed_image)

   def apply_blur(self):
       if self.processed_image is not None:
           self.image_history.append(self.processed_image.copy())
           k = self.blur_slider.value()
           k = k if k % 2 == 1 else k + 1
           blurred = cv2.GaussianBlur(self.processed_image, (k, k), 0)
           self.processed_image = blurred
```



```
  cv2.line(self.processed_image, (x1, y1), (x2, y2),
                     (self.brush_color.red(), self.brush_color.green(), self.brush_color.blue()),
self.brush_size)
            self.display_image(self.processed_image)
            self.last_point = event.pos()

    def mouseReleaseEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.drawing = False

    def pick_color(self):
        color = QColorDialog.getColor()
        if color.isValid():
            self.brush_color = color

    def update_brush_size(self, value):
        self.brush_size = value

    # Drag and Drop
    def dragEnterEvent(self, event):
        if event.mimeData().hasUrls():
            event.accept()
        else:
            event.ignore()

    def dropEvent(self, event):
        for url in event.mimeData().urls():
            file_path = url.toLocalFile()
            self.image = cv2.imread(file_path)
            self.processed_image = self.image.copy()
            self.image_history.clear()
            self.display_image(self.image)

    # Live camera
    def start_camera(self):
        self.capture = cv2.VideoCapture(0)
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_frame)
        self.timer.start(30)
        self.live_preview = True

    def stop_camera(self):
        if hasattr(self, 'capture'):
            self.timer.stop()
            self.capture.release()
            self.live_preview = False

    def update_frame(self):
        ret, frame = self.capture.read()
        if ret:
            self.processed_image = frame.copy()
            self.display_image(frame)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ImageProcessorApp()
    window.show()
    sys.exit(app.exec_())
```