Bulk Image Manipulation Tool

Student: Kyle Clements

Major: Bachelor of Computer Science

Advisor: Dr. Sean Hayes

Expected Graduation Date: May, 2025

Date: April 08, 2025

1. Statement of Purpose

This project aims to develop a user-friendly bulk image manipulation tool using C++ and the Qt framework. The tool enables users to efficiently apply image transformations such as resizing, rotating, flipping, format conversion, and color corrections—including auto white balance—on multiple images simultaneously. The goal is to create a streamlined and professional-grade solution for digital marketers, photographers, and designers.

2. Research & Background

Bulk image manipulation tools are essential in today's digital ecosystem. Existing tools like Fotosizer often fall short in areas such as advanced color correction and automated enhancement. This project addresses these limitations by incorporating features like histogram-based editing, gamma correction, auto white balance, and batch exporting. The tool combines OpenCV for advanced processing and Qt for an intuitive cross-platform GUI, targeting high-efficiency workflows.

3. Project Language(s), Software, and Hardware

Languages: C++ Frameworks/Libraries: Qt (Widgets, Concurrent), OpenCV (for image processing), Boost (for C++ utilities) Software: Visual Studio Code, Qt Designer, GitHub Hardware: Windows 10 PC, Intel i7 CPU, 16GB RAM

4. Project Requirements

- Load and preview image sets from folders
- Perform individual or batch editing operations (rotate, flip, crop, resize)

- Implement color correction tools including histogram-based adjustments
- Include auto white balance and undo/redo support
- Support multiple formats (JPEG, PNG, BMP, XPM)
- Provide dark mode and responsive GUI
- Ensure stable multithreading with QtConcurrent
- Enable keyboard-based navigation (for image switching)
- Create a modular, extendable dialog architecture

5. Project Implementation Description & Explanation

The application is composed of three core modules:

1. **MainWindow** – Manages folder navigation, image previews, batch processing, and export operations.
2. **ImageDialog** – Supports single-image operations such as rotation, flipping, and fine-tuned edits.
3. **ColorLevelsDialog** – Allows advanced color manipulation using histograms, gamma adjustment, and channel selection.

Multi-threading via QtConcurrent enables smooth slider-based interactions without UI freezing. The back-end uses QFuture and QFutureWatcher to offload pixel-level operations without blocking the GUI. Pixel transformations (flip/rotate) are handled via Qtransform, and real-time feedback is enabled using non-blocking UI updates.

Color correction is powered by both coarse and fine sliders/spin boxes, with histogram feedback and interactive black/white point selection. Custom color channel targeting and logarithmic histogram scaling allow precision tuning. A live histogram view with overlays aids in visual comprehension of changes.

The export system features format selection (JPEG, PNG, XPM), quality control, aspect-ratio aware resizing (with Fit/Stretch modes), and options like reverse orientation, "no enlarge," and export dimension units in px or %.

Transformations are visually layered and reversible. The tool ensures that users can preview all changes before saving. The UI supports keyboard events to iterate through images in dialog mode, enhancing workflow speed.

6. Test Plan

Unit tests cover features like rotation, flipping, resizing, format conversion, auto white balance, and gamma correction. Regression tests verify that bug fixes or feature additions do not break existing functionality. Integration tests ensure seamless operation between modules such as dialog-to-main-window communication. User Acceptance Testing simulates real-world use, confirming that tasks such as bulk export and auto enhancement meet user expectations.

Unit tests for this project are embedded within the development process using modular test cases. Each image operation is tested independently to ensure output validity and stability across edge cases. For instance, gamma correction is tested with minimum, neutral, and extreme values to validate output brightness and histogram shifts. Regression testing follows each major code iteration to confirm that no existing feature is compromised by new additions.

7. Test Results

- Rotation, flipping, and resizing features passed all unit tests
- Histogram calculations matched expectations for gray-scale and RGB
- Auto white balance consistently improved image quality
- Regression tests confirmed continued stability after UI updates
- User acceptance testing confirmed usability and functionality goals were met
- Export dimensions and format settings were applied consistently
- Multi-image operations scaled reliably across image sets exceeding 100 files

8. Challenges Overcome

- Prevented UI crashes caused by improper threading using QFutureWatcher
- Resolved layout conflicts in dynamically generated Qt widgets
- Removed unstable preset feature after diagnosing persistent reinitialization bugs
- Developed intuitive workflows for real-time image adjustments using non-blocking threads

Another key challenge was ensuring that pixel-level edits reflected accurately in the UI without causing layout crashes due to race conditions. Debugging these asynchronous operations required careful management of future states and proper tear-down of QFutureWatcher objects before dialog closures.

When developing color adjustment logic, extensive calibration was necessary to balance accuracy and performance.
Additionally, maintaining synchronization between slider/spin box values and live histogram updates required multiple layers of signal-slot bindings, especially with black/white picker tools.

9. Future Enhancements

- Add crop and watermark features
- Integrate OpenCV for additional filters (blur, sharpen, edge detection)
- Enable user profiles and project presets in a robust, crash-safe manner
- Implement drag-and-drop and command-line interfaces for power users
- Extend platform support with mobile or web-based versions
- Create plugin architecture for user-defined filters
- Develop macro scripting for batch automation

Furthermore, an internal scripting engine could be introduced for automated workflows. This would allow advanced users to define presets or macros, enabling the batch application of user-defined filters, metadata edits, and export profiles. Consideration is also being given to integrating AI-driven color enhancement tools in collaboration with OpenCV's deep learning modules.

10. Defense Presentation Slides

Slides will accompany the live demonstration and summarize:

Bulk Image Manipulation Tool

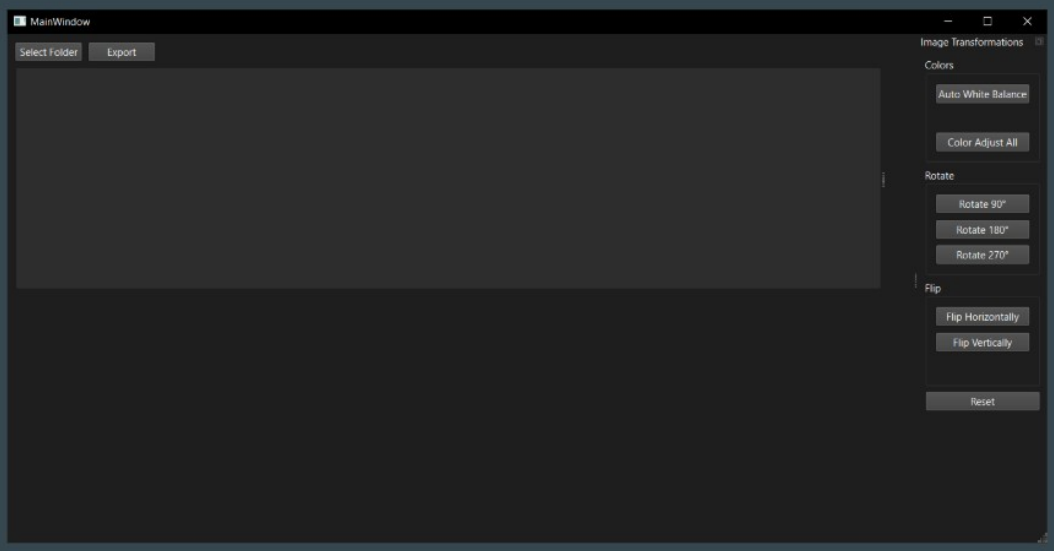
•••

Kyle Clements
Bachelor of Computer Science
Advisor: Dr. Sean Hayes

2

Project Overview

- Streamlined tool for batch image processing
- Designed for digital marketers, photographers, designers
- User-friendly interface with Qt
- Efficient multi-threaded performance in C++

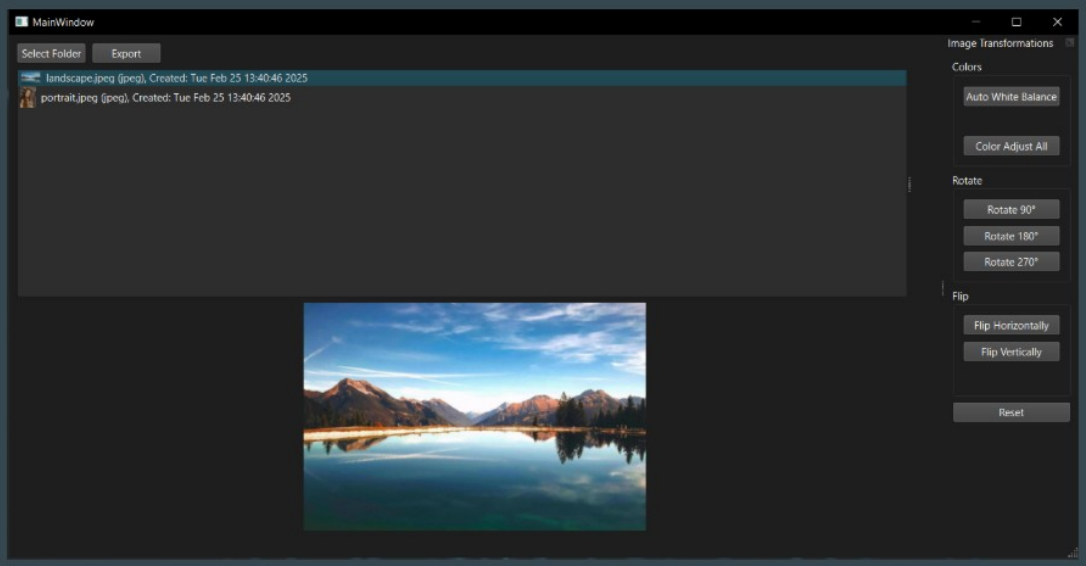


User Interface

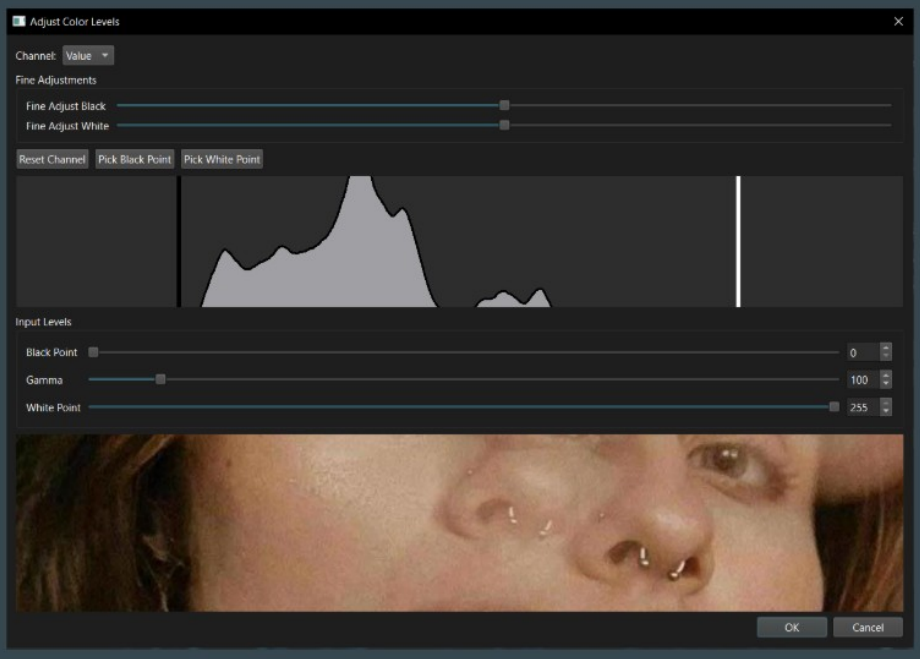
Features & Capabilities

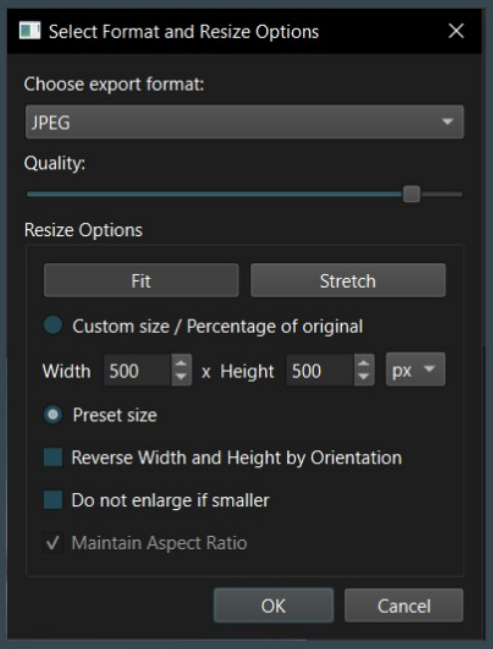
- Batch load, preview, and process image folders
- Rotate, flip, resize, and export in bulk
- Color correction: gamma, white/black points
- Auto white balance & histogram adjustment
- Support for JPEG, PNG, BMP, XPM

5


MainWindow
Select Folder
Export
landscape.jpeg (jpeg), Created: Tue Feb 25 13:40:46 2025
portrait.jpeg (jpeg), Created: Tue Feb 25 13:40:46 2025
Image Transformations
Colors
Auto White Balance
Color Adjust All
Rotate
Rotate 90°
Rotate 180°
Rotate 270°
Flip
Flip Horizontally
Flip Vertically
Reset

6


Adjust Color Levels
Channel:
Value
Fine Adjustments
Fine Adjust Black
Fine Adjust White
Reset Channel
Pick Black Point
Pick White Point
Input Levels
Black Point
Gamma
White Point
0
100
255
OK
Cancel



Architecture

- MainWindow: File management & export
- ImageDialog: Individual edit controls
- ColorLevelsDialog: Histogram + tone control
- Modular, event-driven Qt layout with signal/slot integration

9

Multithreading Approach

- Uses QFuture and QFutureWatcher
- Pixel-level edits offloaded from GUI thread
- Thread-safe transformations with real-time previews

10

Uses QFuture and QFutureWatcher

➔ Launches background image processing

➔ QFutureWatcher listens for task completion

➔ When finished, safely updates UI via onImageProcessingFinished()

```
{
//...
    futureWatcher = new QFutureWatcher<QImage>(this);
    connect(futureWatcher, &QFutureWatcher<QImage>::finished, this, &ColorLevelsDialog::onImageProcessingFinished);
}
```

11

Pixel-level edits offloaded from GUI thread

- ➔ Loops through every pixel to extract RGBA color values
- ➔ Lets the user drag sliders or switch images without lag
- ➔ This pixel loop runs in a background thread to keep the UI responsive

```
for (int y = 0; y < result.height(); ++y) {
    QRgb *scanLine = reinterpret_cast<QRgb*>(result.scanLine(y));
    for (int x = 0; x < result.width(); ++x) {
        QColor color = QColor::fromRgb(scanLine[x]);
        int channels[4];
        channels[0] = color.red();
        channels[1] = color.green();
        channels[2] = color.blue();
        channels[3] = color.alpha();
```

12

Thread-safe transformations with real-time previews

- ➔ Safely updates the UI after background image processing finishes
- ➔ Runs only after the thread is done and the result is ready

```
void ColorLevelsDialog::onImageProcessingFinished()
{
    workingImage = futureWatcher->result();
    imageView->setPixmap(QPixmap::fromImage(workingImage));
    updateHistogram();
}
```

Testing Results

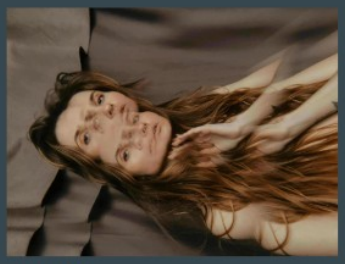
White Balancing -

Horizontal Shift

Vertical Shift

0°

90°



270°

180°