Project Report: Image Processing Application

1. Project Objective

This project is a software application designed for performing image processing tasks. It allows users to perform various operations on images such as conversion (e.g., RGB to grayscale, RGB to HSV), segmentation (e.g., Multi-Otsu, Chan-Vese, Morph Snakes), and edge detection (e.g., Roberts, Sobel, Scharr, Prewitt). The application provides an easy-to-use interface for loading images, applying transformations, and saving or exporting results.

2. Code Structure and Modularity

The project is structured with modularity in mind, where each functionality is placed in a separate file, making the codebase easy to maintain and extend. The following is an explanation of the main files and their roles in the application:

- main.py: Controls the startup and main flow of the application.
- processing.py: Contains the image processing classes that define operations such as RGB to grayscale conversion, Multi-Otsu segmentation, etc.
- qt_design.py: Defines the layout and design of the graphical user interface (GUI) using PyQt5.
- ui.py: Manages user interactions with the GUI, processes the loaded images, and displays them.
- utils.py: Provides utility functions for loading, saving, and exporting images, as well as error handling.
- commands.py: Implements the command pattern to handle operations, supporting undo and redo functionalities.

3. Class Structure and Inheritance - Polymorphism

The project follows Object-Oriented Programming (OOP) principles, and inheritance and polymorphism are used extensively to make the system flexible and maintainable.
Inheritance:
- The `ImageProcessor` class serves as the base class for all image processing operations. For example, `Rgb2GrayProcessor`, `Rgb2HsvProcessor`, and `MultiOtsuProcessor` all inherit from `ImageProcessor` and implement the `process` method in different ways.
- `BaseImageCommand` is the base class for all image processing commands that support undo functionality.

Polymorphism:

- The `process` method is implemented differently in subclasses such as `Rgb2GrayProcessor`, `Rgb2HsvProcessor`, and `MultiOtsuProcessor`, each performing different operations on the image, demonstrating polymorphism.
- In `commands.py`, various command classes such as `GrayscaleCommand`, `HsvCommand`, and `MultiOtsuCommand` inherit from the `BaseImageCommand` class and implement their own version of the `execute` method to perform the desired operation on the image.

4. Doxygen Compatibility

The code is written to be compatible with Doxygen documentation standards. Each class and method is commented with clear explanations of what the code does, what parameters it takes, and what it returns. This documentation style ensures that the code is easily readable, understandable, and maintainable for future developers and contributors.

5. Conclusion

This project follows object-oriented programming principles, which ensures that it is modular, maintainable, and easily extensible. The codebase is organized into separate files, with each file handling a specific functionality, making it easier to manage. The use of inheritance and polymorphism has made the code more flexible, allowing for easy modification and addition of new features. The project has been structured in such a way that it can be expanded in the future with minimal effort.