Summer Fellowship Report

On

Scilab Image Processing Toolbox Development

Submitted by

Sanika Wagle

Under the guidance of

Prof. Prabhu Ramchandran

Aerospace Engineering Department

IIT Bombay

Mentor

Mrs. Rashmi Patankar

July 2026

Acknowledgment

I would like to express my deepest gratitude to the FOSSEE Team, IIT Bombay, for offering me the invaluable opportunity to participate in the FOSSEE Summer Fellowship 2026. I am immensely grateful to Prof. Prabhu Ramchandran for spearheading this initiative, which plays a pivotal role in promoting free and open-source software in education.

A special note of thanks goes to my mentor, Mrs. Rashmi Patankar, for her unwavering guidance, valuable feedback, and encouragement throughout the fellowship. Her expertise significantly enriched my learning journey and was instrumental in the successful completion of my assigned tasks.

I am thankful to everyone who has been part of this journey, supporting and inspiring me along the way. Your unwavering belief in my capabilities has made this experience incredibly rewarding.

Contents

Chapter 1

Introduction

Scilab is a free and open-source, cross-platform numerical computational package and a high-level, numerically oriented programming language.

It can be used for signal processing, statistical analysis, image enhancement, fluid dynamics simulations, numerical optimization, modeling and simulation of explicit and implicit dynamical systems, and (if the corresponding toolbox is installed) symbolic manipulations.

Scilab is one of the two major open-source alternatives to MATLAB, the other being GNU Octave. Scilab puts less emphasis on syntactic compatibility with MATLAB than Octave does, but is similar enough to easily transfer skills between the two systems.

IIT Bombay is leading the effort to popularize Scilab in India, and the Scilab Image Processing Toolbox is one of its endeavors toward this cause. This effort is part of the Free and Open Source Software for Science and Engineering Education (FOSSEE) project, supported by the National Mission on Education through ICT of the Ministry of Education.

The FOSSEE Image Processing Toolbox is a comprehensive suite designed for the analysis, enhancement, manipulation, and visualization of digital images, developed and maintained by FOSSEE, IIT Bombay.

It offers a wide range of functions covering fundamental and advanced image processing techniques, including image filtering, geometric transformations, intensity adjustments, segmentation, morphological operations, feature detection, and image restoration.

Users can perform tasks such as image enhancement, noise reduction, edge detection, color space conversion, histogram analysis, and object detection. The toolbox also supports frequency-domain image processing using Fourier transforms, enabling efficient filtering and analysis of image content.

With its intuitive interface and extensive testing, the Image Processing Toolbox is a powerful tool for engineers, researchers, educators, and students working in fields such as computer vision, medical imaging, remote sensing, robotics, and digital image analysis.

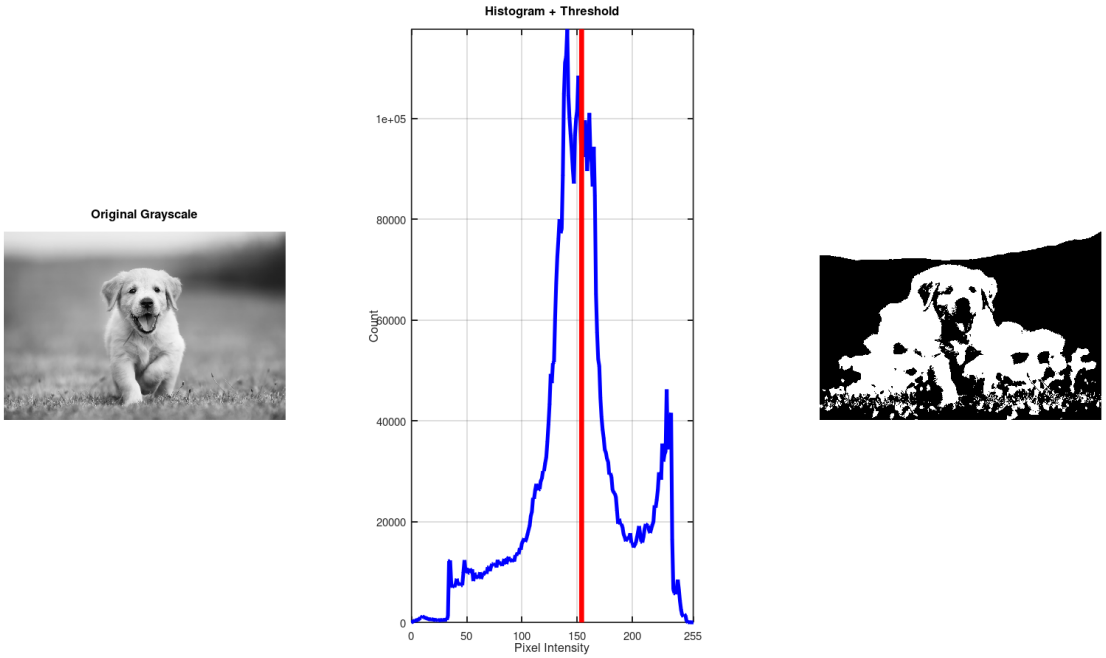

Figure 1.1: Histogram analysis and binary thresholding output generated by the original Octave implementation of `graythresh`.

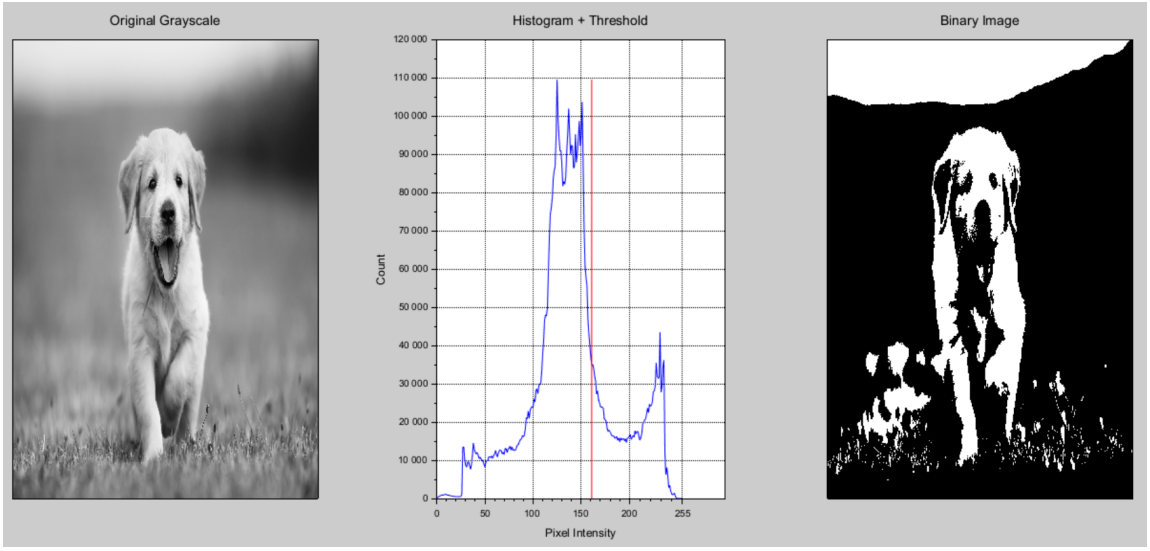

Figure 1.2: Histogram analysis and binary thresholding output generated by the translated native Scilab implementation of `graythresh`, confirming functional equivalence with the Octave output above.

Chapter 2

Image Processing Toolbox Development

2.1 Overview

The toolbox comprises an amalgamation of various functions (listed in the directory `FOSSEE-Image-Processing-Toolbox/macros/`) that fall into two main categories: functions that perform the required computations natively in Scilab, and functions that pass user inputs to Octave for processing.

Converting functions of the latter category into native Scilab implementations has been the primary objective of this internship project. The motivation for this is multifaceted, with the key reasons listed below:

- Calling Octave's image processing functions from Scilab requires the FOSSEE Scilab-Octave Toolbox as an intermediary. This introduces an unnecessary dependency on both the FOSSEE Scilab-Octave Toolbox and Octave for the Scilab Image Processing Toolbox.
- Native Scilab implementations offer significantly better performance than forwarding computations to Octave while using Scilab merely as an interface.
- The FOSSEE Scilab-Octave Toolbox is still under active development and currently has limitations. For example, functions involving logical values, structures, or image-based inputs and outputs may not be fully supported. Since image processing functions frequently operate on multidimensional image data and produce image outputs, these limitations considerably reduce the toolbox's effectiveness.

The process of rewriting these functions in native Scilab involves several stages, described in the following sections.

2.2 Development Workflow

The development workflow evolved significantly over the course of the internship as I gained a deeper understanding of Scilab and the Octave-to-Scilab translation process. Figure 2.1 summarizes the overall workflow.

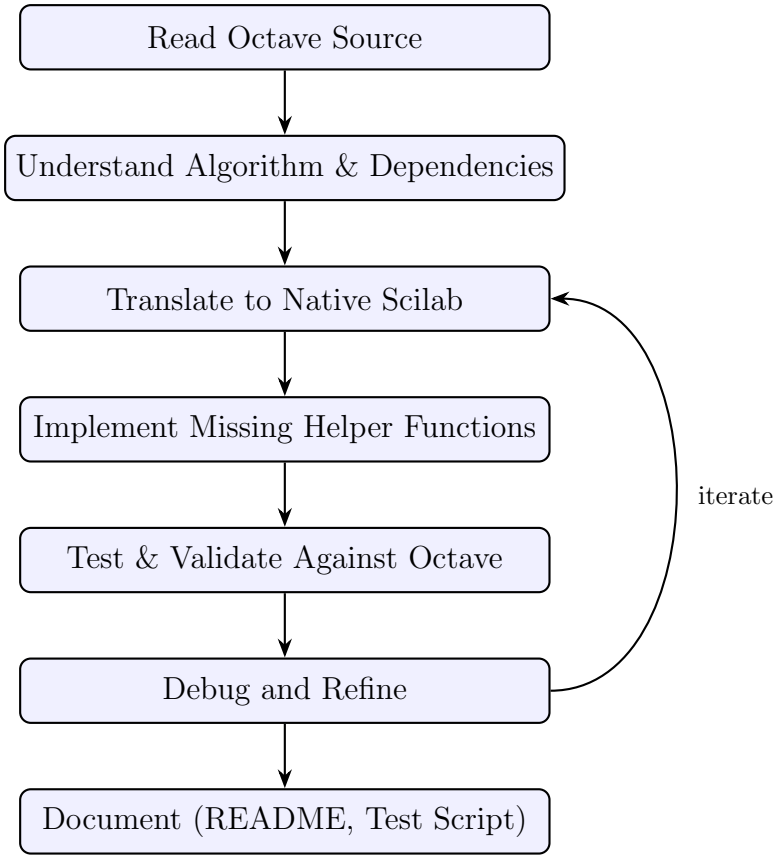

Figure 2.1: Overall development workflow followed for each translated function.

2.2.1 Reading the Octave Implementation

Analyzing the original Octave implementation is an essential first step in planning the translation of a function to native Scilab. During this stage, it is important to identify any helper functions or dependencies that may not have direct equivalents in Scilab, and to determine how they can be implemented.

To locate the source code of the desired function, first consult the Octave documentation to determine the package to which the function belongs. In most cases, functions are found in one of the following:

1. Octave's `image` package: the source code for image processing functions can be accessed at https://octave.sourceforge.io/image/. Most functions are implemented in pure Octave, while a few performance-critical routines may be written in C++.
2. Octave's core package: some image-related utility functions are part of Octave's core distribution. In such cases, download the source code of the latest Octave release and locate the required function within the source tree.

This analysis helps achieve the following objectives:

- Estimate the effort and time required for the translation.
- Identify dependent functions that are unavailable in Scilab and must be implemented separately.

- Assess the overall complexity of the translation process and anticipate potential challenges.

2.2.2 Line-by-Line Translation

Once the functionality of the Octave implementation has been understood, the next step is to translate the code into native Scilab, carefully converting each statement while preserving the original functionality. The primary focus is on handling the syntactic and semantic differences between Octave and Scilab.

Some important differences encountered during the translation process are listed below. Whenever necessary, the official documentation of both Octave and Scilab should be consulted.

- In Octave, control structures such as `if`, `for`, `while`, and `switch` are terminated using `endif`, `endfor`, `endwhile`, and `endswitch`, respectively. In Scilab, all of these constructs are terminated using the single keyword `end`.
- Several predefined constants have different names in the two languages. For example, Octave uses `pi`, `eps`, `i/j`, and `true/false`, whereas Scilab uses `%pi`, `%eps`, `%i`, and `%T/%F`, respectively.
- The Octave function `print_usage()` has no direct equivalent in Scilab. It is typically replaced with an appropriate `error("message")` statement that reports incorrect usage.
- Although the overall programming syntax of Octave and Scilab is similar, careful attention must be paid to differences in indexing, function behavior, built-in library support, and data structures to ensure that the translated function produces results consistent with the original implementation.

2.2.3 Comparing Built-in Functions and Implementing Missing Ones

After identifying the helper functions used by an image processing function and verifying their availability in Scilab, the next step is to compare their behavior across both platforms. Even when Scilab and Octave provide functions with the same name, their implementations, default parameters, or outputs may differ.

The recommended approach is to consult the official documentation for both Octave and Scilab and modify the translated implementation wherever necessary. If the behavioral differences are substantial, it is often preferable to write a wrapper function or a Scilab-compatible version that reproduces Octave's behavior.

For example, the `max` and `min` functions behave differently in the two environments. In Octave, these functions operate column-wise by default when applied to matrices, whereas Scilab computes the maximum or minimum over the entire matrix unless a dimension is explicitly specified. Such differences must be carefully handled to ensure that the translated function produces results identical to its Octave counterpart.

For helper functions that are unavailable in Scilab, a similar workflow is followed: study the Octave documentation, obtain the source code, understand the underlying algorithm,

and implement an equivalent version in native Scilab.

Occasionally, some functions are implemented in C++ for performance reasons rather than in Octave itself. In such cases, the algorithm can be extracted from the C++ source and reimplemented directly in Scilab. Alternatively, Scilab's C/C++ interface may be used to create a wrapper around the existing implementation and dynamically link it with Scilab. This approach is generally reserved for situations where a native Scilab implementation is impractical, as it requires a good understanding of both C++ and Scilab's external interface.

2.2.4 Common Challenges Encountered

Several recurring challenges were encountered while translating functions from Octave to Scilab:

- Differences in behavior between similarly named Octave and Scilab built-in functions (e.g., dimension handling in `max`/`min`).
- Helper functions available in Octave's `image` package with no direct Scilab equivalent, requiring a from-scratch implementation.
- Differences in image datatype handling and conversions (e.g., `uint8`, `uint16`, `double` representations).
- Indexing differences and edge cases at array boundaries.
- Inconsistent matrix dimension handling between the two environments.
- Differences in boundary/padding conventions used by functions such as `padarray`.
- Preserving exact Octave-compatible behavior while still following idiomatic Scilab conventions.

2.2.5 Testing, Validation, and Iteration

Testing is one of the most important stages of the development workflow, as it ensures that the translated Scilab implementation is functionally equivalent to its Octave counterpart. Although this stage can be time-consuming, a systematic testing strategy significantly improves the reliability of the implementation.

A good starting point is the set of test cases provided at the end of the corresponding Octave source file. These are executed whenever possible to verify the correctness of the translated function. Where certain test cases cannot be reproduced in Scilab due to language-specific differences or unavailable dependencies, additional test cases are designed to validate the same functionality.

Each translated function was validated using the following strategy:

- Running identical inputs through both the original Octave implementation and the translated Scilab implementation.
- Comparing numerical outputs for exact or near-exact agreement.
- Comparing generated images/matrices for consistency, where applicable.

- Testing error conditions and invalid inputs to confirm consistent error handling.
- Testing every supported calling sequence of the function.
- Covering boundary conditions and edge cases (empty inputs, single-pixel images, extreme parameter values, etc.).

(a) Octave Output (b) Translated Scilab Output

Figure 2.2: Validation showing identical output between Octave and the translated Scilab implementation.

Any discrepancies identified during testing were analyzed, the implementation refined accordingly, and the testing process repeated until the translated function consistently produced results equivalent to its Octave counterpart.

2.2.6 Git-Based Development Workflow

All development followed a standard collaborative Git workflow using GitHub:

- Fork the FOSSEE Image Processing Toolbox repository.
- Create a feature branch for each function or set of related functions.
- Implement the function, its test script, and its README.
- Commit changes with descriptive messages and push to the feature branch.
- Open a Pull Request against the upstream repository.
- Address reviewer comments and iterate until the Pull Request is merged.

2.3 Current Status

Following the development workflow described above, I successfully translated **17 image processing functions** from Octave to native Scilab, spanning several domains of image processing:

- Image quality assessment (`immse`, `psnr`)
- Histogram processing (`histeq`, `stretchlim`, `graythresh`)
- Binary image analysis (`bwarea`)
- Template matching (`normxcorr2`)
- Quadtree decomposition (`qtdecomp`, `qtgetblk`, `qtsetblk`)
- Color space conversion (`lab2uint8`, `lab2uint16`, `lab2double`, `lab2single`)
- Image generation and padding utilities (`phantom`, `padarray`, `getrangefromclass`)

2.3.1 Documentation Pattern

Since the objective is to preserve the behavior of the corresponding Octave image processing functions, the existing Octave documentation serves as the primary reference when documenting the Scilab implementations.

In addition, the corresponding MATLAB documentation was consulted wherever applicable, since differences or inconsistencies sometimes exist between Octave and MATLAB implementations. When unexpected behavior was observed during testing, comparing outputs against both Octave and MATLAB helped identify implementation issues.

The documentation for each function is placed immediately below its declaration as a single comment block, consisting of:

1. **Calling Sequence**: valid function signatures and supported calling sequences.
2. **Parameters**: input and output arguments, including data types, dimensions, valid ranges, and default values where applicable.
3. **Description**: the function's purpose, behavior, implementation details, default settings, expected inputs/outputs, dependencies, and important notes.
4. **Examples**: representative usage examples.

Since the translated Scilab functions closely match the behavior and interface of their Octave counterparts, only minor modifications to the original documentation were generally required.

All of my work is available in my GitHub repository: FOSSEE Image Processing Toolbox Development Repository.

2.3.2 Functions Completed

Table 2.1 groups the completed functions by category, and Table 2.2 lists each function along with its dependencies.

Category	Functions
Image Quality Assessment	`immse`, `psnr`
Padding & Utilities	`padarray`, `getrangefromclass`
Histogram Processing	`histeq`, `stretchlim`, `graythresh`
Binary Image Processing	`bwarea`
Image Generation	`phantom`
Quadtree Operations	`qtdecomp`, `qtgetblk`, `qtsetblk`
Template Matching	`normxcorr2`
Color Space Conversion	`lab2uint8`, `lab2uint16`, `lab2double`, `lab2single`

Table 2.1: Translated functions grouped by category.

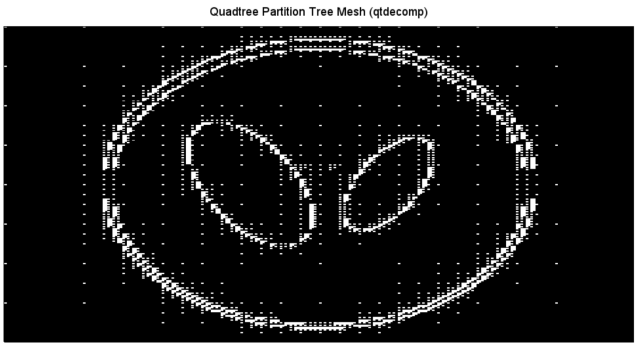

Figure 2.3: Variable-sized block grid generated by the `qtdecomp` function.

Table 2.2: Translated functions with their dependencies.

S.No	Function	Dependencies / Custom Functions
1	immse	
2	psnr	immse
3	padarray	
4	histeq	
5	bwarea	
6	getrangefromclass	intmin, intmax
7	graythresh	intmax
8	qtdecomp	
9	qtgetblk	
10	qtsetblk	
11	normxcorr2	postpad
12	stretchlim	postpad
13	phantom	
14	lab2uint8	lab2cls
15	lab2uint16	lab2cls
16	lab2double	lab2cls
17	lab2single	lab2cls

2.3.3 Performance Benefits

Replacing Octave-dependent implementations with native Scilab code provides several concrete benefits:

- Reduced dependence on Octave and the FOSSEE Scilab-Octave Toolbox as an intermediary.
- Faster execution, since computations no longer need to be forwarded to an external interpreter.
- Easier long-term maintenance, as the entire codebase resides in a single language.
- Better portability across systems where Octave may not be installed.
- Simpler deployment and packaging of the toolbox for end users.

2.3.4 Incomplete Implementations and FIXME Issues

All implemented functions have been completed, and no pending `FIXME` issues remain.

Chapter 3

Contributions

While Chapter 2 describes *how* the functions were developed, this chapter consolidates *what* was concretely delivered as part of the internship. Every function translated during the fellowship was shipped as a complete unit — implementation, test script, and documentation — rather than as standalone code, so that it could be directly reviewed and merged into the FOSSEE Image Processing Toolbox repository.

3.1 Deliverables Summary

Table 3.1 summarizes the concrete outputs of the internship.

Deliverable	**Details**
Translated functions	17 image processing functions translated from Octave to native Scilab.
Helper/custom functions	4 shared helper functions implemented: `lab2cls`, `postpad`, `intmin`, `intmax`.
Test scripts	17 dedicated test scripts covering normal, boundary, and edge cases.
README files	17 README files documenting calling sequence, parameters, description, and examples.
Validation	Every function's output cross-checked against its original Octave implementation.
Bug fixes	Implementation issues identified during iterative testing were diagnosed and resolved.
Pull Requests	All code, tests, and documentation submitted as reviewed Pull Requests to the FOSSEE Image Processing Toolbox repository.

Table 3.1: Summary of deliverables produced during the internship.

3.2 Repository Contributions

All contributions were made directly to the FOSSEE Image Processing Toolbox codebase following the Git-based workflow described in Section 2.2.6. This ensured that:

- Each function went through peer review by the mentor before being merged.
- The commit history provides full traceability of the translation and debugging process for every function.
- The toolbox's public repository reflects a growing, community-reviewed set of native Scilab implementations, reducing its dependence on Octave over time.

3.3 Quality Assurance

Beyond simply producing working code, each deliverable was held to the same quality bar:

- Consistent documentation format across all 17 functions (Calling Sequence, Parameters, Description, Examples), as detailed in Section 2.3.1.
- Test scripts exercising every supported calling sequence, not just the primary use case.
- Explicit comparison against Octave (and, where relevant, MATLAB) outputs rather than visual inspection alone.

Together, these deliverables represent a self-contained, reviewable, and maintainable addition to the Scilab Image Processing Toolbox.

Chapter 4

Scilab-Octave Toolbox Development

4.1 Overview

The FOSSEE Scilab-Octave Toolbox is an interoperability tool that enables Scilab to execute Octave functions directly within the Scilab environment. During the development of the Scilab Image Processing Toolbox, it was primarily used to execute functions from Octave's `image` package and validate the translated Scilab implementations by comparing their outputs with the original Octave functions.

Along with the Scilab-Octave Toolbox, the `IPCV` module available through the ATOMS (Automated Modules for Scilab) package manager was extensively used during the implementation process. The IPCV module provides several native image processing functions in Scilab that closely resemble those in Octave's `image` package. Wherever applicable, these functions were used as building blocks, while the remaining functionality was implemented directly in Scilab.

The Scilab-Octave Toolbox provides the following capabilities:

- Execute Octave functions directly from Scilab.
- Transfer supported data between Scilab and Octave.
- Support multiple input and output arguments.
- Load and use Octave packages, including the `image` package.
- Display Octave error messages within the Scilab environment.
- Facilitate testing and validation by comparing translated Scilab functions with their Octave counterparts.

Although the goal of the Image Processing Toolbox is to replace Octave-dependent implementations with native Scilab code, the Scilab-Octave Toolbox remains an essential tool for development, testing, and verification throughout the translation process.

4.2 Current Status

Throughout the internship, the FOSSEE Scilab-Octave Toolbox served as an important aid during the development of the Scilab Image Processing Toolbox. It was primarily used to execute the original Octave implementations, study their behavior, and validate the translated Scilab functions. Comparing outputs from both environments helped ensure that the native Scilab implementations were functionally equivalent to their Octave counterparts.

Key contributions of the Scilab-Octave Toolbox to the development workflow are summarized below:

- Executed Octave image processing functions directly from Scilab to understand their implementation and expected behavior.
- Verified translated Scilab functions by comparing their outputs with those produced by the corresponding Octave functions.
- Helped identify differences in the behavior of built-in functions, default parameter handling, and data representations between Scilab and Octave.
- Assisted in debugging translated functions by enabling incremental validation throughout the implementation process.
- Provided a reliable reference implementation, making it easier to ensure correctness before integrating functions into the Scilab Image Processing Toolbox.

The use of the Scilab-Octave Toolbox throughout the internship significantly simplified the translation and validation process, contributing to accurate and reliable native Scilab implementations.

Chapter 5

Learnings

The internship at FOSSEE, IIT Bombay, was a valuable learning experience that enhanced both my technical knowledge and professional skills. Some of the key takeaways from this internship are listed below.

1. **Open-Source Development**
    - Gained practical experience in contributing to an open-source software project.
    - Learned to write clean, maintainable code and use Git for version control and collaboration.
    - Understood the importance of documentation and code quality in software development.
2. **Technical Skills**
    - Improved my proficiency in Scilab and Octave while gaining a deeper understanding of image processing algorithms.
    - Strengthened my debugging, testing, and problem-solving skills through function translation and validation.
    - Learned to analyze existing implementations and adapt them to a different programming environment.
3. **Software Development Practices**
    - Learned the importance of systematic testing, edge-case analysis, and technical documentation.
    - Gained experience developing reliable and reusable software components following a structured workflow.
4. **Professional Growth**
    - Improved my ability to work independently, accept constructive feedback, and continuously refine my work.
    - Enhanced my technical communication skills through documentation and reporting.

Chapter 6

Future Scope

While the internship achieved its core objective of translating 17 functions, several avenues remain for future work on the Scilab Image Processing Toolbox:

- Translate the remaining Octave-dependent functions in the toolbox to native Scilab.
- Expand automated test coverage, including randomized and property-based test generation.
- Improve and standardize documentation across all functions in the toolbox.
- Further reduce reliance on the FOSSEE Scilab-Octave Toolbox where native alternatives exist.
- Optimize computationally intensive functions (e.g., `normxcorr2`, `qtdecomp`) for large images.
- Expand integration with the IPCV ATOMS module to reuse more native Scilab building blocks.

Chapter 7

Conclusion

My internship at FOSSEE, IIT Bombay, was a valuable learning experience that allowed me to contribute to the development of the Scilab Image Processing Toolbox while gaining practical exposure to open-source software development. Throughout the internship, I strengthened my understanding of image processing algorithms, Scilab programming, software testing, and technical documentation.

The primary focus of my work was translating image processing functions from Octave into native Scilab implementations. By replacing Octave-dependent functions with Scilab-native code, the toolbox becomes more efficient, maintainable, and independent of external dependencies. During the internship, I successfully translated 17 image processing functions, ensuring their correctness through comprehensive testing and validation against the corresponding Octave implementations.

In addition to code translation, I prepared documentation, test scripts, and extensive test cases for each function. This systematic approach helped ensure that the implementations were reliable, well-documented, and easy to maintain for future development.

I sincerely thank my mentor, Mrs. Rashmi Patankar, for her continuous guidance and encouragement throughout the internship. I am also grateful to Prof. Prabhu Ramchandran and the entire FOSSEE team for providing a collaborative environment that fostered learning and innovation.

Overall, this internship has significantly enhanced my technical skills, problem-solving abilities, and understanding of collaborative software development. The knowledge and experience gained during this period will serve as a strong foundation for my future academic and professional pursuits.

References

- FOSSEE Image Processing Toolbox Repository
- Octave Image Package Documentation
- Scilab Image Processing Toolbox (Scilab.in)
- Scilab Official Documentation
- Octave Official Documentation
- IPCV ATOMS Module Documentation
- MATLAB Image Processing Toolbox Documentation (used for verification)