Summer Fellowship Report

On

Scilab Image Processing Toolbox development

Submitted by

Zachariah P Jijo

Under the guidance of

Mentor

Ms. Rashmi Patankar

Acknowledgment

I am deeply grateful to my mentor Ms. Rashmi Patankar for her invaluable guidance, support, and encouragement throughout my internship with the FOSSEE Team at IIT Bombay. Her expertise and patience have been pivotal in my learning and professional growth during this period.

I remain dedicated to contributing to Scilab and other FOSSEE initiatives. I am enthusiastic about the opportunities ahead and eager to make meaningful contributions to these projects.

I am thankful to everyone who has been part of this journey, supporting and inspiring me along the way.

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, and modeling and simulation of explicit and implicit dynamical systems.

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 allow skills to transfer easily between the two systems.

IIT Bombay is leading the effort to popularise 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 Scilab Image Processing Toolbox is a comprehensive suite designed for the analysis, manipulation, and transformation of images, developed and maintained by FOSSEE, IIT Bombay. It offers a wide range of functions covering fundamental and advanced image processing techniques, including morphological operations, feature detection, image type conversion, and transform-based methods such as the Hough transform.

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

Chapter 2

Image Processing Toolbox Development

2.1 Overview

The toolbox comprises an amalgamation of various functions located in the toolbox macros directory. The primary goal of this internship project was to translate Octave image package functions into native Scilab code, ensuring functional equivalence with the original Octave implementations.

The motivation for this is multifaceted:

- Using Scilab-native code for computations eliminates the dependency on an intermediary toolbox (the FOSSEE Scilab-Octave Toolbox) and on Octave itself, making the toolbox self-contained.
- Native Scilab implementations are more performant, as they do not incur the overhead of cross-platform function calls.
- The Scilab-Octave Toolbox has limitations with certain data types (boolean values, structs, images) as inputs or outputs. Image processing functions frequently deal with such types, making native ports essential.

The porting methodology followed strict functional equivalence to the Octave originals: same variable names where possible, no added functionality beyond what the source provides, and three deliverables per function — a `.sci` implementation, a `.sce` test suite, and a Markdown README.

2.2 Development Workflow

The workflow followed a consistent four-step process throughout the internship:

- Reading the Octave implementation
- Translating line by line
- Comparing built-in functions and writing missing equivalents
- Testing and iterating

2.2.1 Reading Octave Implementation

This analysis is crucial for planning the code translation process from Octave to Scilab. Functions in the Octave `image` package are available at https://octave.sourceforge.

io/image/. Most functions are written in pure Octave, though some may be implemented in C++.

The outcomes of this step include:

- Estimation of translation complexity and time requirements.
- Identification of sub-functions not directly available in Scilab.
- Assessment of behavioral differences between Octave and Scilab built-ins of the same name.

2.2.2 Line By Line Translation

This task requires meticulous line-by-line translation. The focus is primarily on syntax and semantic differences between Octave and Scilab. Key differences encountered include:

- Block-closing keywords: Octave uses `endif`, `endfor`, `endwhile`; Scilab uses a unified `end`.
- Constants: Octave's `pi`, `true`/`false`, `eps` become Scilab's `%pi`, `%T`/`%F`, `%eps`.
- The `end` keyword used as a last-index operator in Octave becomes `$` in Scilab.
- Scilab has no compound assignment operators (`+=`, `-=`, etc.).
- `print_usage` in Octave is replaced with `error()` in Scilab.

2.2.3 Comparing Built-in Functions and Writing Missing Ones

Even when Scilab and Octave share a function name, their behaviors may differ. Consulting documentation for both platforms and adjusting the implementation accordingly is essential. Key substitutions made during this project include:

Octave	**Scilab equivalent**
`any()` / `all()`	`or()` / `and()`
`isnumeric()`, `issparse()`, `islogical()`	`type()` code comparisons
`strcmpi()`	`convstr(s, "l")`
`sort()`	`gsort()`
`reshape()`	`matrix()`
`end` (index)	`$`
`ones(vector)` / `zeros(vector)`	`matrix(ones(total,1), sz)`
`argn()` order	Returns `(nargout, nargin)` in Scilab

For sub-functions unavailable in Scilab, the same workflow was applied: consult Octave's documentation, obtain the source code, and translate it to Scilab.

2.2.4 Test and Iterate

Test cases were derived primarily from the Octave source files themselves, as many Octave functions include embedded `%!test` blocks. These were adapted into Scilab `.sce` test scripts. Outputs were compared against Octave's reference results. Test suites were

written to be comprehensive, covering different calling sequences and edge cases, with at least one example per calling form.

2.3 Current Status

Following the workflow outlined above, 15 functions have been successfully ported from the Octave `image` package to native Scilab code.

Each function is accompanied by inline documentation and a standalone README file containing the calling sequence, parameter descriptions, a function description, and examples.

2.3.1 Documentation Pattern

Since the functions are intended to mirror their Octave counterparts, Octave's documentation serves as the primary reference. The documentation for each function consists of four components:

1. **Calling Sequence**: The valid forms in which the function can be called.
2. **Parameters**: Expected types and acceptable values for each parameter.
3. **Description**: Comprehensive description of the function's behavior, defaults, and interpretation of inputs and outputs.
4. **Examples**: Demonstrations of correct usage.

2.3.2 Functions Completed

S.No	Function	Dependencies / Custom Functions
1	`colorangle`	—
2	`hough_line`	—
3	`hough`	`hough_line`
4	`hough_circle`	`bwmorph`
5	`houghtf`	`hough_line`, `hough_circle`
6	`houghpeaks`	`isimage`, `isnumeric`, `hough_line`, `hough`
7	`houghlines`	`isimage`, `isnumeric`, `hough_line`, `hough`, `houghpeaks`
8	`isind`	`isimage`
9	`imcast`	`isimage`, `isind`
10	`im2single`	`isimage`, `isind`, `imcast`
11	`conndef`	—
12	`imreconstruct`	`conndef`
13	`imhmax`	`conndef`, `imreconstruct`
14	`imcomplement`	—
15	`imhmin`	`conndef`, `imreconstruct`, `imcomplement`

2.3.3 Incomplete Implementations and FIXME Issues

No issues.

Chapter 3

Scilab-Octave Incompatibilities Encountered

3.1 Overview

Porting functions from the Octave `image` package to Scilab requires navigating a number of systematic incompatibilities between the two languages. This chapter documents the key issues encountered during this project.

3.2 Common Errors Encountered

- `argn()` **argument order**: In Scilab, `argn()` returns `(nargout, nargin)` — the opposite order from what Octave's `nargin`/`nargout` convention might suggest. This caused subtle bugs in optional-argument handling across multiple functions.
- `ones(vector)` / `zeros(vector)` **sizing behavior**: In Octave, passing a size vector to `ones()` or `zeros()` produces an array of those dimensions. Scilab treats the argument differently. The fix required using `matrix(ones(total, 1), sz)` to explicitly reshape.
- **Missing** `any()` / `all()` **equivalents**: Scilab does not provide `any()` or `all()` as direct built-ins for logical reduction. These were replaced with `or()` and `and()` respectively.
- **Type-checking functions**: Octave's `isnumeric()`, `issparse()`, and `islogical()` have no direct equivalents in Scilab. These were replaced with `type()` code comparisons.
- `strcmpi()` **unavailable**: Case-insensitive string comparison using `strcmpi()` in Octave was replaced with `convstr(s, "l")`
- `sort()` **vs** `gsort()`: Octave's `sort()` is replaced by Scilab's `gsort()`, which has a different argument interface and default sort direction.
- `reshape()` **vs** `matrix()`: Octave's `reshape()` is replaced by Scilab's `matrix()`, which provides equivalent functionality with a different syntax.
- **No cell array dynamic index expansion**: Scilab does not support dynamic cell array index expansion as Octave does. Explicit helper patterns were required to handle cases that relied on this behavior.
- **No compound assignment operators**: Scilab does not support `+=`, `-=`, `*=`, or similar operators. All such expressions were expanded into explicit assignment statements.

Chapter 4

Learnings

This internship provided a wide range of technical and professional experiences:

1. **Technical Skills Enhancement**
    - Gained proficiency in Scilab, Git, and GitHub workflows.
    - Developed a deep understanding of image processing algorithms including morphological reconstruction, the Hough transform, and image type casting.
    - Strengthened the ability to read, analyze, and faithfully port foreign codebases with minimal deviation from source behavior.
    - Developed advanced coding practices: systematic testing, structured documentation, and iterative debugging.
2. **Feedback and Continuous Improvement**
    - Learned to receive and act on constructive feedback effectively.
    - Developed the habit of cross-referencing Octave, MATLAB, and Scilab documentation to resolve behavioral discrepancies.
    - Gained an appreciation for rigorous test suite design as a tool for validating functional equivalence.
3. **Professionalism and Work Ethic**
    - Developed a strong sense of professionalism in an open-source contribution setting.
    - Learned the importance of consistent code style, documentation standards, and reproducible deliverables.
4. **Adaptability and Flexibility**
    - Learned to adapt translation strategies when direct syntactic equivalents did not exist.
    - Developed resilience in debugging subtle cross-language behavioral differences that were not immediately apparent from documentation alone.

Chapter 5

Conclusion

In conclusion, my internship experience at FOSSEE, IIT Bombay, working on the development of the Scilab Image Processing Toolbox has been immensely rewarding and educational.

The primary focus of this work was translating Octave `image` package functions into native Scilab code. Through meticulous workflow planning, careful line-by-line translation, rigorous testing, and iterative improvement, 15 functions have been successfully ported to Scilab, each accompanied by comprehensive documentation and test suites.

These implementations cover a range of image processing capabilities — morphological operations (`imreconstruct`, `imhmax`, `imhmin`), Hough transform pipelines (`hough`, `houghpeaks`, `hough_line`, `houghlines`, `houghtf`, `hough_circle`), image type utilities (`imcast`, `im2single`, `imcomplement`), and supporting functions (`conndef`, `colorangle`, `isind`).

I am deeply grateful to my mentor Ms. Rashmi Patankar for her unwavering support and guidance throughout this internship. Her expertise and encouragement have been instrumental in my professional development.

This internship has not only strengthened my technical skills but also reinforced my commitment to contributing to the advancement of open-source software for scientific and engineering education.

References

- https://octave.sourceforge.io/image/
- https://scilab.in/fossee-scilab-toolbox/image-processing-toolbox
- https://github.com/FOSSEE/fossee-scilab-octave-toolbox