Summer Fellowship Report

On

Scilab Image Processing Toolbox Development

Submitted by

Arjun E Naik
Computer Science Student
Indian Institute of Information Technology, Nagpur

Under the guidance of

Prof. Prabhu Ramachandran
Aerospace Engineering Department
IIT Bombay

Mentor

Ms. Rashmi Patankar

July 17, 2026

Acknowledgment

I want to express my heartfelt thanks to the FOSSEE team at IIT Bombay for giving me the opportunity to be part of the FOSSEE Summer Fellowship. This fellowship gave me a platform to study Octave's image processing implementations closely and port them into native, working Scilab code.

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

I also sincerely thank Prof. Prabhu Ramachandran for his insightful guidance, which has significantly shaped my understanding of open-source systems and how they are built and maintained.

I am thankful to everyone who has been part of this journey, supporting and encouraging me along the way. Their belief in my capabilities made this experience genuinely rewarding, and I look forward to continuing to contribute to FOSSEE's open-source efforts.

Contents

Chapter 1

Introduction

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

It is used for signal processing, statistical analysis, image enhancement, fluid dynamics simulation, numerical optimization and modeling, and the simulation of explicit and implicit dynamical systems, and, where the corresponding toolbox is installed, symbolic manipulation.

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 that skills transfer fairly easily between the two systems.

IIT Bombay leads 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 Education (FOSSEE) project, supported by the National Mission on Education through ICT (NMEICT), Ministry of Education, Government of India.

The FOSSEE Scilab Image Processing Toolbox is a comprehensive suite designed for the analysis, filtering, morphological processing, and quantitative evaluation of images, developed and maintained by FOSSEE, IIT Bombay.

It offers functions covering statistical image measures, local neighborhood filtering, lookup-table-based operations, binary image topology, synthetic test-pattern generation, and region-of-interest based color selection.

Users can compute the Shannon entropy of an image, apply local-neighborhood statistical filters, perform lookup-table transformations on binary neighborhoods, compute Euler numbers for binary images, generate LUT test patterns, and extract regions of interest based on hue. With its faithful port from Octave's image package and its extensive documentation, this toolbox is a valuable resource for students, researchers, and educators working in image analysis and computer vision.

Chapter 2

Image Processing Toolbox Development

2.1 Overview

The toolbox is organized as a collection of independent function directories at the root of the repository (e.g. `entropy/`), each containing the Scilab implementation, its test suite, and a dedicated README. This is in contrast to housing all functions inside a single shared source directory, which is how several other Scilab toolbox repositories are structured. Keeping each function self-contained and independently documented makes the toolbox easier to extend as new functions are ported by different contributors.

The repository is available at:
https://github.com/Arjun-E-Naik/scilab-image-processing-toolbox

Broadly, the functions in the repository fall into two categories: functions that perform the required computation natively in Scilab, and functions that were originally intended to pass input through to Octave (via the FOSSEE Scilab-Octave Toolbox) for processing.

Converting functions of the latter type into fully native Scilab implementations has been the primary goal of this internship project. The motivation for this is multifaceted, but a few reasons are listed below:

- Calling Octave's image processing functions from Scilab requires an intermediary toolbox, the FOSSEE Scilab-Octave Toolbox.
- Native Scilab computation is significantly more performant than routing calculations through Octave and using Scilab merely as an interface.
- The FOSSEE Scilab-Octave Toolbox is still under active development and is somewhat limited in scope. Functions that operate on boolean/logical arrays, structs, or images with non-trivial shape behavior cannot be reliably passed through the toolbox, which significantly diminishes its usefulness for image processing functions that frequently rely on logical masks and structuring elements.

Within this internship, my own contributions cover the functions listed below in Section 2.2.6. Each of these was ported line-by-line from Octave's image package, tested against Octave's own output for correctness, and documented with a comprehensive README following a consistent template described in Section 2.2.5.

This process of rewriting functions in native Scilab code involves several steps, which are explained from here onwards.

2.2 Development Workflow

The workflow was refined as I gained experience during this internship. Here is a summary of the workflow:

- Reading the Octave implementation
- Translating line by line
- Comparing inbuilt functions and writing missing ones
- Testing and iterating
- Writing a comprehensive README for each function

2.2.1 Reading Octave Implementation

This analysis is crucial for planning the code translation process from Octave to Scilab. It is important to note that certain sub-functions and MEX/oct-file implementations in Octave may not have direct equivalents in Scilab.

To find the Octave documentation for a desired function, I referred to the Octave Forge image package source (https://octave.sourceforge.io/image/) and cross-checked behavior against MATLAB documentation where Octave's own documentation was ambiguous.

The outcomes of this process include:

- Estimation of time constraints.
- Identification of sub-functions and built-ins not available in Scilab.
- Assessment of the complexity involved in the translation process, particularly around type handling (logical vs. double) and memory layout (row-major vs. column-major operations).

2.2.2 Line By Line Translation

This task required meticulous line-by-line translation. The focus was primarily on syntax and semantic differences between Octave and Scilab. Some key differences encountered:

- Octave's `nargin`/`nargout` were replaced with Scilab's `argn(2)`/`argn(1)` for argument introspection; in certain legacy contexts, `argn(0)` was used in place of `nargin`.
- Scilab has no native `logical` array type. Wherever Octave returned a `logical` array, the Scilab port returns a `double` array, using `bool2s()` for explicit boolean-to-double conversion at the boundary.
- Octave's `endif`, `endfor`, and `endwhile` are all simply `end` in Scilab.
- Constants differ: Octave's `true`/`false` vs. Scilab's `%T`/`%F`; `pi`, `eps`, `j` vs. `%pi`, `%eps`, `%i`.
- `print_usage()` is not available in Scilab; replaced with `error("message")`.

2.2.3 Comparing Inbuilt Functions And Writing Missing Ones

Once the sub-functions used by the Octave implementation were identified, checking their availability in Scilab still left room for subtle behavioral mismatches, since Scilab and Octave functions sharing a name do not always behave identically.

A number of built-ins used routinely by the Octave image package have no Scilab equivalent and required custom helper implementations, including:

- `im2uint8`
- `imhist`
- `immse`
- `getrangefromclass`
- `isvector` and `isscalar` (in certain contexts)

These helper functions were deliberately written as narrow shims to support only the parent function being ported, rather than as general-purpose Octave-compatible reimplementations. This scoping decision is explicitly documented in each function's README to avoid the false impression that, e.g., a bundled `im2uint8` helper is a full drop-in replacement for Octave's.

A similar narrow-shim approach was used for `padarray_constant`, a helper required while porting `integralImage3` to correctly zero-pad a 3-D volume before computing its cumulative sum. It was extended to accept a `direction` argument (`"pre"`, `"post"`, `"both"`) as a faithful line-by-line translation of Octave's N-D constant-padding logic. Since Scilab has no equivalent of Octave's cell-expansion indexing syntax `A(idx{:})` for indexing along a dimension chosen at runtime, this was implemented using `select` blocks combined with `execstr`-based dynamic construction of the indexing expression. The specific bugs this helper triggered, and how they were diagnosed, are discussed in Chapter 3.

One particularly subtle issue arose while porting `applylut`: Scilab's `matrix()` reshaping operates in column-major order, and when combined with `conv2`'s automatic kernel-flipping behavior, weight matrices had to be pre-rotated by 180° to reproduce Octave's output. Additionally, Octave encodes neighborhood bit patterns in left-to-right, top-to-bottom order, while the natural column-major approach in Scilab produces a right-to-left, bottom-to-top ordering; this only becomes visible with asymmetric neighborhoods – symmetric test patterns mask the discrepancy entirely, which made this bug especially difficult to catch during initial testing.

Another engine-level difference surfaced while handling edge cases with empty arrays: Scilab's `zeros()` collapses any dimension that is zero down to a canonical 0×0 array, whereas Octave/MATLAB preserve the original shape (e.g., a 0×12 array remains 0×12). This is a limitation of the Scilab engine itself rather than a translation bug, and is documented as such rather than "fixed."

2.2.4 Test And Iterate

Wherever available, I used the test cases embedded at the bottom of the Octave source files as a first pass, and supplemented these with hand-constructed edge cases (empty

inputs, asymmetric neighborhoods, boundary conditions, multi-frame and volumetric inputs) where the original tests were insufficient.

Wherever feasible, I ran the actual Scilab and Octave code side-by-side in a sandboxed environment (installing `octave-cli` alongside Scilab) rather than computing expected values by hand. This practice surfaced several real bugs and at least one genuine engine-level limitation (the empty-array shape-collapsing behavior described above) that would have been very difficult to catch by inspection alone.

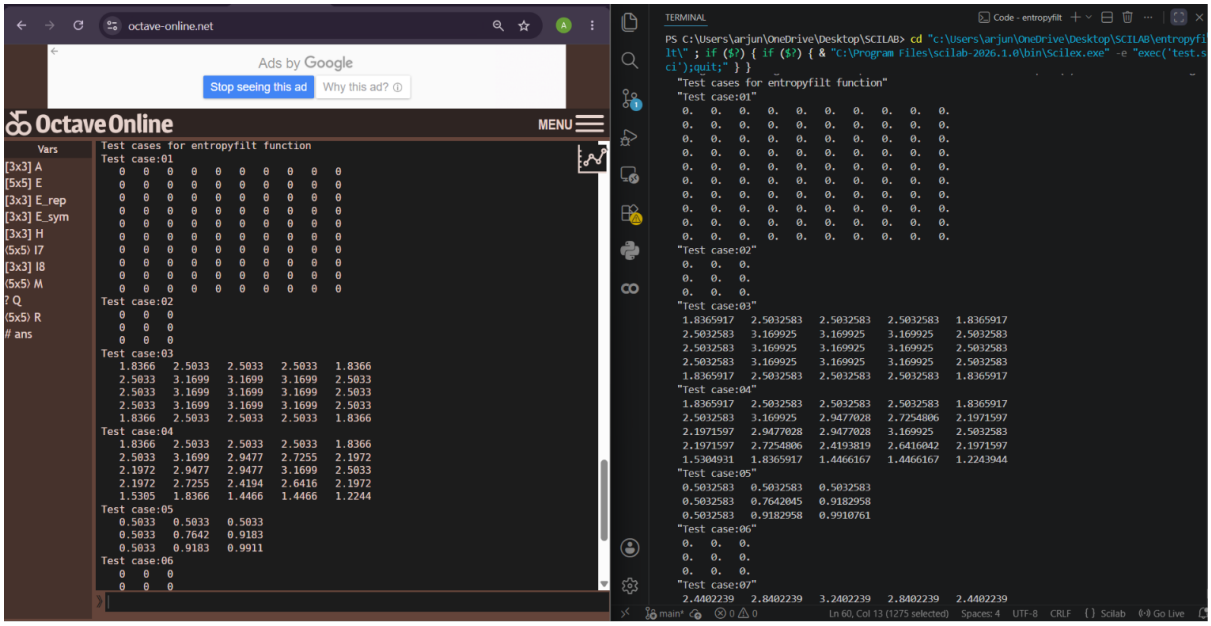

Figure 2.1: Cross-platform test validation: Scilab vs. Octave output

Following the workflow outlined above, I have successfully translated and documented the functions listed in Section 2.2.6 from Octave's image package into native Scilab code.

2.2.5 Documentation pattern

Each function is intended to faithfully mirror its Octave counterpart's behavior (while flagging deliberate or unavoidable deviations). I adopted a single, consistent README template across all functions, anchored initially to the `entropy.sci` README and applied identically to every function ported afterward. Each README consists of:

1. **Overview table** summarizing the function's purpose at a glance.
2. **Calling Sequence** usage snippet.
3. **Parameter reference**: parameter and return-value tables.
4. **Full worked examples** for every test case, with expected outputs shown explicitly.

Any inaccuracies discovered in the original Octave in-code comments (for example, a mislabeled shading description encountered while studying `stdfilt`) were flagged and corrected in the corresponding Scilab documentation rather than silently carried over.

2.2.6 Functions Completed

S.No	Function	Notes / Dependencies
1	`entropy`	Reference README style; Shannon entropy of image histogram
2	`makelut`	Creation of a lookup table from a sub-image predicate function
3	`entropyfilt`	Local entropy filtering over sliding neighborhoods
4	`rangefilt`	Local range (max−min) filtering
5	`stdfilt`	Local standard deviation filtering; cross-validated with Octave
6	`applylut`	LUT-based neighborhood transform; convolution kernel pre-rotation
7	`bweuler`	Euler number computation for binary images
8	`intlut`	Substitutes integer pixel values using a mapped lookup table (LUT)
9	`roicolor`	Region-of-interest extraction via hue/color range
10	`grayslice`	Partitions the intensity range of a grayscale image into a set of intervals
11	`fftconv2`	Performs 2-D convolution using the Fast Fourier Transform
12	`otsuthresh`	Computes the global threshold of a histogram using Otsu's method
13	`wavelength2rgb`	Converts visible light wavelengths into an RGB color representation
14	`integralImage`	Computes the 2-D cumulative-sum image of an input image
15	`integralImage3`	Extends the integral-image concept to a 3-D volume

Chapter 3

Cross-Platform Behavioral Differences: Scilab vs. Octave

3.1 Overview

Beyond simple syntax translation, a substantial part of this internship involved understanding *why* certain Octave functions behaved differently once ported to Scilab, even when the translated code appeared line-for-line equivalent. This chapter consolidates the recurring categories of divergence encountered while porting the image processing toolbox.

3.2 Key Differences Encountered

- **Type system:** The absence of a native `logical` type in Scilab meant every Octave function returning a logical mask had to be re-specified as returning `double`, with `bool2s()` used at explicit conversion points. This is a recurring documentation item across nearly every function's Octave-compatibility table.
- **Argument introspection:** `nargin`/`nargout` → `argn(2)`/`argn(1)`, with `argn(0)` used in a small number of legacy call-pattern contexts.
- **Memory layout and convolution:** Scilab's column-major `matrix()` reshaping, combined with `conv2`'s automatic kernel flipping, required deliberate 180° pre-rotation of weight matrices in `applylut` to reproduce Octave's numerical output exactly.
- **Bit-encoding order:** Also specific to `applylut`, Octave encodes neighborhood bit patterns left-to-right, top-to-bottom, while the natural Scilab port produces a right-to-left, bottom-to-top ordering. Symmetric test patterns mask this difference entirely, so asymmetric neighborhoods were needed to expose and confirm the discrepancy.
- **Reduction-function dimension defaults:** Scilab's bare `cumsum(I)`, when given a matrix or hypermatrix, flattens the input in column-major order and computes a single cumulative sum over that flattened sequence, rather than summing along the first non-singleton dimension the way Octave's `cumsum` does implicitly. This surfaced while porting `integralImage3` and was resolved by explicitly detecting

the first non-singleton dimension of the input, d, and calling `cumsum(I, d)` with that dimension supplied.

- **Hypermatrix display suppression:** Scilab's `disp()` silently omits any all-zero page (slice) when printing a 3-D array (hypermatrix), which can make a legitimately correct but sparse result look like a failed computation. This was encountered while validating `integralImage3` test output and was addressed with a small custom helper, `dispImage3`, that iterates over the third dimension explicitly and prints every slice, including all-zero ones.

- **Dynamic N-D indexing:** Octave's cell-expansion indexing syntax, `A(idx{:})`, used to index into an array along a dimension chosen at runtime, has no direct Scilab equivalent. While extending the `padarray_constant` helper (used by `integralImage3`) to support a `direction` argument, this was reproduced using `select`-based branching combined with `execstr`-based construction and evaluation of the indexing expression at runtime.

- **Missing built-ins:** `im2uint8`, `imhist`, `immse`, `getrangefromclass`, and context-dependent absences of `isvector`/`isscalar` all required custom, narrowly-scoped helper functions.

- **Empty array behavior:** Scilab's `zeros()` collapses any zero-length dimension to a canonical 0×0 array, while Octave/MATLAB preserve the original shape (e.g. 0×12). This is documented as an engine-level limitation rather than something correctable within the port.

- **Naming collisions:** Studying the function implementation surfaced a Scilab warning about redefining an existing function (`cell2mat`), due to Scilab's flatter macro namespace compared to Octave's package scoping.

- **Functions in .cc files** Studying the `rangefilt` implementation,observed some inbilt functions are in .cc files. Because these .cc extensions run faster than .m .

- **Helper function scope:** Bundled helper functions are intentionally narrow shims supporting only their parent function, not general Octave-compatible reimplementations – a distinction explicitly called out in each README to prevent misuse.

3.3 Common Errors And Fixes

- **Undefined variable / function errors from missing built-ins:** Resolved by writing narrowly-scoped local helper functions (`im2uint8`, `imhist`, `immse`, `getrangefromclass`) rather than assuming Scilab equivalents existed.

- **Mismatched LUT indices in `applylut`:** Traced to the combined effects of column-major reshaping and `conv2`'s kernel flipping; fixed via explicit 180° kernel pre-rotation.

- **Cumulative sums computed along the wrong axis in `integralImage3`:** Traced to Scilab's bare `cumsum` flattening the input in column-major order instead of operating along the first non-singleton dimension; fixed by detecting that dimension explicitly and passing it to `cumsum(I, d)`.

- **Silent shape mismatches on empty-array edge cases:** Identified as a genuine Scilab engine limitation (canonical 0×0 collapsing) rather than a translation bug; documented rather than worked around.

Chapter 4

Learnings

I have had a lot of valuable experiences from this internship, some of which are enumerated below.

1. **Technical Skills Enhancement:**
    - Gained proficiency in Scilab, Octave, and the subtle behavioral differences between the two despite their syntactic similarity.
    - Learned to reason about memory layout (row-major vs. column-major) and how it silently affects operations like convolution and reshaping.
    - Learned to debug N-dimensional array (hypermatrix) behavior in Scilab, including reduction-function axis defaults and display quirks that do not surface with ordinary 2-D matrices.
    - Developed strong documentation habits, including writing structured, reproducible READMEs for every function.
    - Improved debugging skills by cross-validating results against a second computational engine rather than relying on manual verification.
2. **Feedback and Continuous Improvement:**
    - Learned to receive and act on constructive feedback from my mentor.
    - Gained an understanding of the importance of continuous learning and iterative refinement of both code and documentation.
3. **Professionalism and Work Ethic:**
    - Developed a strong sense of professionalism in a remote, open-source work setting.
    - Learned the importance of accurate, honest documentation – including flagging engine-level limitations rather than hiding them.
4. **Adaptability and Flexibility:**
    - Learned to adapt to unexpected behavioral mismatches between platforms that were not documented anywhere and had to be discovered empirically.
    - Developed resilience while tracking down subtle bugs (e.g. the `applylut` bit-ordering issue and the `integralImage3` axis and padding bugs) that only manifested under specific test conditions.

Chapter 5

Conclusion

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

Throughout this internship, I contributed to the open-source community, particularly by porting core image processing functions from Octave's image package into well-documented, native Scilab implementations.

The primary focus of my work was ensuring each translated function behaved identically to its Octave counterpart wherever possible, and clearly documenting the cases where it could not – whether due to Scilab's lack of a native logical type, its column-major memory layout, its differing reduction-function defaults over hypermatrices, or genuine engine-level limitations such as empty-array shape collapsing. Through workflow planning, careful function translation, cross-platform testing, and iterative improvement, I converted and documented 15 functions with accompanying test suites and comprehensive READMEs, culminating in `integralImage3`, whose 3-D cumulative-sum and zero-padding logic surfaced some of the most subtle engine-level bugs encountered during the entire internship.

Looking ahead, I am committed to continuing my contributions to FOSSEE projects, particularly in further expanding the Image Processing Toolbox with additional morphological and filtering functions, and in continuing to refine the documentation so it serves as a reliable reference for future contributors.

I am deeply grateful to my mentor, Ms. Rashmi Patankar, and to Prof. Prabhu Ramachandran, for their support and guidance throughout this internship. Their 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. I look forward to applying the knowledge and experience gained here in my future endeavors.

References

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