

Original Software Publication

SHINIER: An open-source Python package for controlling low-level image properties

Mathias Salvas-Hébert^{a,*}, Nicolas Dupuis-Roy^{b,1}, Catherine Landry^a, Ian Charest^a,
Frédéric Gosselin^a

^a Cerebrum, Département de psychologie, Université de Montréal, Canada

^b Elephant Scientific Consulting, Canada

ARTICLE INFO

Keywords:

Vision science

Image processing

Luminance contrast

Histogram equalization

Fourier spectrum normalization

ABSTRACT

SHINIER (Spectrum, Histogram, and Intensity Normalization, Equalization, and Refinement), written in Python, is an open-source package that replicates and extends the functionality of the popular SHINE toolbox written in MATLAB (Willenbockel et al., 2010). Like SHINE, it includes functions for normalizing and scaling mean luminance and contrast, for specifying either the full Fourier amplitude spectrum or its rotational average, and for exact histogram specification. SHINIER further introduces a CIE xyY-based color-processing strategy, and image-dithering algorithms for improved tonal rendering. It also improves existing workflows through enhanced exact histogram-specification procedures, more efficient memory management, and a modular architecture designed for scalability and future extension.

1. Metadata

Nr	Code metadata description	Metadata
C1	Current code version	v0.2.1
C2	Permanent link to code/ repository used for this code version	https://github.com/Charestlab/shinier
C3	Permanent link to reproducible capsule	-
C4	Legal code license	BSD 3-Clause
C5	Code versioning system used	git
C6	Software code languages, tools and services used	Python (with C/C++ extensions via CPython)
C7	Compilation requirements, operating environments and dependencies	Python (≥3.9, <3.13), numpy (≥1.22, <2.1), Pillow (≥9.0), tqdm (≥4.60), matplotlib (≥3.9), pydantic (≥2.12), cython (≥3.0); optional C/C++ compiler
C8	If available, link to developer documentation/manual	https://shinier.readthedocs.io/
C9	Support email for questions	mathias.salvas-hebert@umontreal.ca

2. Motivation and significance

2.1. The importance of low-level image control

When conducting experiments with humans, animals, or machines, the choice of stimuli is critical. We usually intend observers to rely on features that genuinely support recognition in real life. However, experimental image sets—necessarily small subsets of the virtually infinite possible scenes and viewing conditions—often contain accidental features that can be exploited instead. For example, in a dog–cat categorization task, observers might succeed not because they attend to diagnostic shape or texture cues, but because the dog images (often taken outdoors in bright sunlight) have luminance histograms with higher means and greater variance than cat images (typically photographed indoors under dim lighting). These luminance differences are artifacts of illumination, not reliable distinguishing properties of dogs versus cats in the real world.

One way to avoid such confounds is to use artificially generated stimuli with fully controlled low-level properties. Another is to rely on very large naturalistic image databases, such as the Natural Scenes Dataset [1], where idiosyncratic correlations tend to average out. When working with natural images, if such large-scale resources are

* Corresponding author.

E-mail address: mathias.salvas-hebert@umontreal.ca (M. Salvas-Hébert).

¹ Nicolas Dupuis-Roy is a shared first author.

unavailable—or impractical due to time constraints with human or animal participants—normalizing and adjusting low-level image properties, as permitted by the SHINIER package, becomes essential.

2.2. Scientific impact and community adoption

For more than fifteen years, the SHINE toolbox [2], of which SHINIER is a Python implementation, has served as the primary solution for controlling low-level image properties. According to Google Scholar, it has been cited >1450 times—an average of roughly 100 citations per year—highlighting its widespread adoption and importance in vision science.

SHINIER implements in Python all functionalities of the original MATLAB-based SHINE toolbox while extending its capabilities through improved features, new functionality motivated by community needs, increased accessibility, and a restructured codebase designed for optimization and scalability. Among these extensions is a CIE xyY-based color-processing pipeline, distinct from the HSV- and CIELAB-based approaches previously implemented in SHINE_color [3], that provides more direct control of colorimetric luminance while preserving chromaticity for controlled experimental applications.

2.3. Workflow and usage

The package is openly accessible via `pip install shinier`, with its source code hosted on CharestLab's GitHub (<https://github.com/Charestlab/shinier>). A typical workflow begins with a set of grayscale or color images (e.g., faces, scenes, objects) that the user seeks to match with respect to specific low-level image properties. Through either the

Python API or the interactive CLI (shell command: `shinier`), the user provides an input folder of images and, optionally, a binary mask to restrict processing to a region of interest (e.g., the face area). The user then selects a processing mode—for instance, histogram matching (Mode 2; see Fig. 1) or full Fourier spectrum matching (Mode 4). The processed images are saved to an output folder along with a log file.

2.4. Related algorithms and software

SHINIER relies on several algorithms, standards, and software frameworks. Histogram-related methods include Exact Histogram Specification (EHS; [4]) and Exact Global Histogram Specification (EGHS) optimized for structural similarity [5]. The package also provides classical global Histogram Equalization (HE) and several recent low-contrast and low-light enhancement methods: Tripartite Image Decomposition-Based Histogram Equalization (TIDHE; [6]), Recursive Dualistic Fuzzy Histogram Equalization (RDFHE; [7]), Nonlinear Fuzzification–Linear Defuzzification-Based Image Contrast Enhancement (NFLDICE; [8]), Bi-Entropy Curve Equalization (BETCE; [9]), and Sakaguchi-Type Function-Based Cost-Effective Filtering (SFCEF; [10]). Structural similarity is assessed using the Structural Similarity Index (SSIM; [11]). Dithering algorithms include Floyd–Steinberg error-diffusion dithering [12] and noisy-bit dithering [13]. Color-space conversion and gamut-control procedures rely on CIE colorimetry standards [14,15], ITU-R color encoding standards [16–18], and the sRGB standard [19]. Color-conversion validation was performed using the Colour software framework [20]. SHINIER also implements MATLAB-compatible numerical operators and processing behaviors to reproduce the original SHINE workflows while maintaining backward

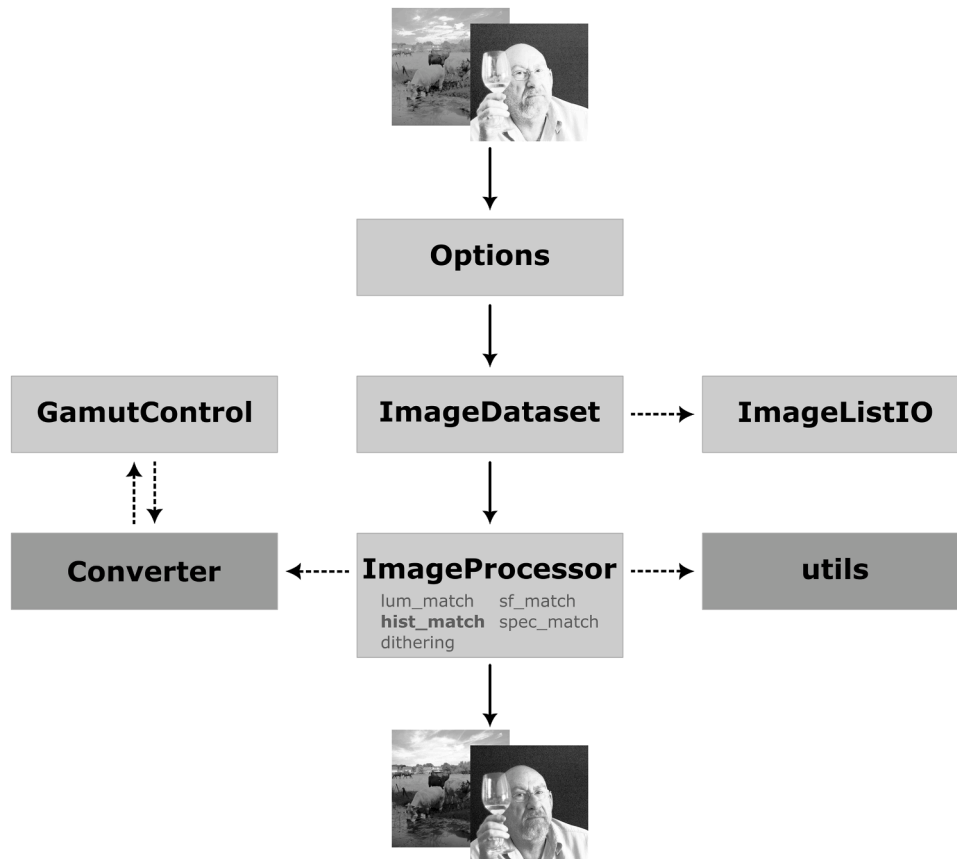


Fig. 1. Illustration of the SHINIER architecture showing a typical workflow from input to output. The image folder and processing parameters are defined in `Options`, images are managed in `ImageDataset`, and processing is performed by `ImageProcessor` with calls to utility modules (e.g., `Converter`, `utils`). Solid arrows denote the main processing flow, whereas dashed arrows indicate supporting module interactions. Bright boxes indicate modules that contain a single class, whereas dark boxes indicate modules that contain multiple classes. Fig. 2 presents a detailed example of this histogram-matching workflow.

compatibility with MATLAB-based implementations [21].

3. Software description

3.1. Software architecture

SHINIER adopts an object-oriented programming (OOP) architecture that reorganizes the image-processing workflow into three core classes (Fig. 1): (1) `Options`, a user-facing class that defines the input images and associated processing parameters; (2) `ImageDataset`, which manages image storage and associated buffers; and (3) `ImageProcessor`, which applies the processing pipeline in a modular fashion. This modular organization improves maintainability and facilitates extension.

The use of large datasets whose memory requirements exceed available random-access memory (RAM) is now the norm in computer vision and increasingly common in vision science. To ensure SHINIER's scalability, we introduced `ImageListIO`, a dedicated subclass that can handle large datasets. This class leverages fast solid-state drive (SSD) read-and-write operations to circumvent RAM limitations. Specifically, when the `conserve_memory` option is enabled, images are read from and written disk on the fly rather than stored in RAM. Although this new approach incurs lower real-time performance, in practice, the overhead is negligible compared to the core image-processing operations of the package. This approach also provides opportunities for future development, such as video processing.

SHINIER was deliberately designed with minimal dependencies to simplify long-term maintainability and with a lightweight architecture to increase accessibility. These design choices required additional effort to preserve low-level performance through optimized implementations (e.g., Cython-based fast convolution) and to integrate external algorithms, including a dedicated color subpackage for color-space support (`Converter` and `GamutControl`).

A detailed overview of the package architecture and available functionality is provided in the continuously updated SHINIER documentation that is available on read the docs at <https://shinier.readthedocs.io/>. The supplementary material provides three additional reference tables: Table S1, "SHINIER public API overview"; Table S2, "Mapping of SHINE and SHINE_color functions to their SHINIER equivalents"; and Table S3, "Selected MATLAB built-in and toolbox functions and their SHINIER equivalents".

3.2. Software functionalities

3.2.1. Core SHINE-compatible processing modes

SHINIER replicates the four image processing modes of SHINE for controlling low-level image properties across stimulus sets: (1) `lum_match`, in which images are equalized in terms of mean luminance and contrast (standard deviation), preserving the overall shape of their luminance distribution; (2) `hist_match`, in which images' luminance histograms are matched exactly; (3) `sf_match`, in which the rotational average of the Fourier spectrum is matched across images, thereby matching energy at each spatial frequency while preserving orientation-specific structure and phase information; and (4) `spec_match`, in which the full amplitude of the Fourier spectrum is matched across images while preserving phase information. Targets for mean luminance and luminance contrast, histogram, or Fourier properties can be explicitly defined or derived from the average of the input image set. All modes can be applied globally or restricted to specific image regions (e.g., foreground vs. background) using masks. SHINIER also provides a `legacy_mode` that fully replicates SHINE's operations while, by default, using improved Python-based numerical handling.

3.2.2. Color processing, grayscale conversion, and luminance control

SHINE was limited to grayscale images, whereas SHINE_color added RGB-, HSV-, and CIELAB-based processing for color images [3]. SHINIER

supports both grayscale and color processing and includes independent RGB-channel processing, generally recommended for spatial-frequency-based matching, as well as a CIE xyY-based pipeline, generally recommended for pixel-based matching. Unlike HSV value (V) and CIELAB lightness (L*), CIE xyY separates chromaticity (x, y) from linear relative luminance (Y) following appropriate RGB linearization. This enables direct luminance manipulation while initially preserving chromaticity. Accurate stimulus presentation nevertheless requires appropriate monitor calibration, as discussed by Willenbockel et al. [2].

Operationally, linear luminance determines whether SHINIER processes the RGB channels directly or converts the image to CIE xyY, whereas `as_gray` determines whether the output remains in color or becomes single-channel. In direct RGB mode, the channels are processed independently or combined using an equal-weight mean; in CIE xyY mode, only Y is processed and is either retained as grayscale or recombined with the preserved chromaticity coordinates (x, y) to reconstruct the color image. This reconstruction can produce out-of-gamut or distorted colors. Most color conversion packages naively clip invalid RGB values. In addition to this approach, SHINIER provides four gamut-control options: reducing luminance either (1) per image or (2) across the dataset, and desaturating chroma either (3) per image or (4) across the dataset. As an exception to these processing pathways, enabling `legacy_mode` produces a single-channel intensity image using MATLAB- and SHINE-compatible grayscale conversion based on the luma weighting shared by NTSC and ITU-R Rec. 601.

3.2.3. Low-light and low-contrast image enhancement

SHINIER also provides a dedicated enhancement pipeline for low-contrast and low-light grayscale images. In addition to classical CDF-based histogram equalization, it implements several recent methods: TIDHE, RDFHE, NFLDICE, BETCE, and SFCEF [8,9,6,7,10]. These deterministic, non-learning-based methods process each image independently and require neither training data nor pretrained models, although their performance may vary across images and they may amplify noise or alter local contrast.

3.2.4. Exact histogram specification

Furthermore, the exact histogram workflow, which enforces identical pixel-intensity distributions across images, was substantially revised in SHINIER. Two additional exact histogram specification algorithms were implemented: the method proposed by Coltuc et al. [4] and a Gaussian-kernel variant. New specification options were also introduced to handle special cases, including images with no identical pixel values and situations in which a single algorithm is insufficient. In such cases, a hybrid strategy applies Gaussian ordering.

3.2.5. Dithering

Finally, the Floyd-Steinberg dithering algorithm [12] and the noisy-bit dithering algorithm [13] were added to the package. In both cases, the effective tonal resolution of the processed images is increased by exploiting spatial pooling mechanisms in human and animal visual systems.

3.3. Sample usage example

```
from shinier import (
    Options, ImageDataset, ImageProcessor
)
opts = Options(
    input_folder    = "data/input",
    output_folder   = "data/output",
    mode            = 2,          # hist_match
    # Collapse RGB grayscale to one channel
    as_gray         = True,
    linear_luminance = True,
)
dataset = ImageDataset(options=opts)
processor = ImageProcessor(dataset=dataset)
```

This code example applies exact histogram specification to the input images and saves the processed results to the output folder. Figs. 1 and 2 use the input and output images produced by this example. By default, as in this case, the target histogram is the average histogram of the input image set, and exact histogram specification is performed using a Gaussian-based variant inspired by Coltuc, Bolon and Chassery [4], with a noise fallback when isoluminant pixels persist.

4. Impact

SHINIER has been developed in response to the scientific community's shift toward Python-based workflows in computer vision, artificial intelligence, and modern vision science. Python has become one of the dominant languages in scientific computing [22], yet no Python package provides the specialized functionality for low-level image control that SHINE and SHINE_color offer to vision scientists. General-purpose Python libraries such as scikit-image and OpenCV are not designed for this purpose. As a result, many laboratories rely on heterogeneous in-house implementations that differ in algorithmic and numerical details (e.g., FFT implementations, dithering procedures, numerical precision, or uint8 versus float64 representations), potentially introducing methodological variability across studies. SHINIER builds on the conceptual and methodological foundations of SHINE. It provides a unified, transparent, and reproducible framework that consolidates specialized image-processing procedures within a single open-source Python package. It further extends this framework through histogram equalization, low-light and low-contrast enhancement, and dithering for improved tonal rendering. Because low-level image statistics can strongly influence perceptual and neural responses, standardizing these procedures facilitates comparisons across studies and promotes methodological consistency across laboratories.

Beyond reproducing the core functionality of SHINE, SHINIER

introduces several improvements aimed at increasing the robustness of image-processing operations. For example, image precision is tracked across processing stages and, when possible, maintained in 64-bit floating-point format until final conversion to 8-bit unsigned integer representations. Similarly, the revised exact histogram specification procedure reduces reliance on arbitrary noise-based tie-breaking in favour of more deterministic ordering strategies, particularly when identical pixel intensities are rare or absent.

By bringing these methods into the Python ecosystem, SHINIER integrates naturally with contemporary scientific software, including pandas, scikit-learn, and PyTorch. This facilitates the construction of automated analysis pipelines, machine-learning workflows, large-scale image-processing applications, and high-performance computing deployments without requiring users to re-implement or validate low-level numerical procedures. As an open-source Python package, SHINIER also reduces dependence on proprietary MATLAB licenses, thereby lowering financial and technical barriers to adoption across research environments. SHINIER, or its preliminary versions, has already been used for stimulus preparation and psychophysical control in several studies conducted in our laboratories as well as those of our colleagues.

We hope that SHINIER will contribute to the broader adoption of standardized, transparent, and reproducible image-processing practices across vision science and related fields.

5. Conclusions

In this paper, we present SHINIER, an open-source, lightweight Python package for controlling low-level image properties in experimental stimuli used with humans, animals, or machines. In addition to reproducing the core functionality of the widely used SHINE MATLAB toolbox, SHINIER introduces several methodological and numerical improvements while maintaining backward compatibility when

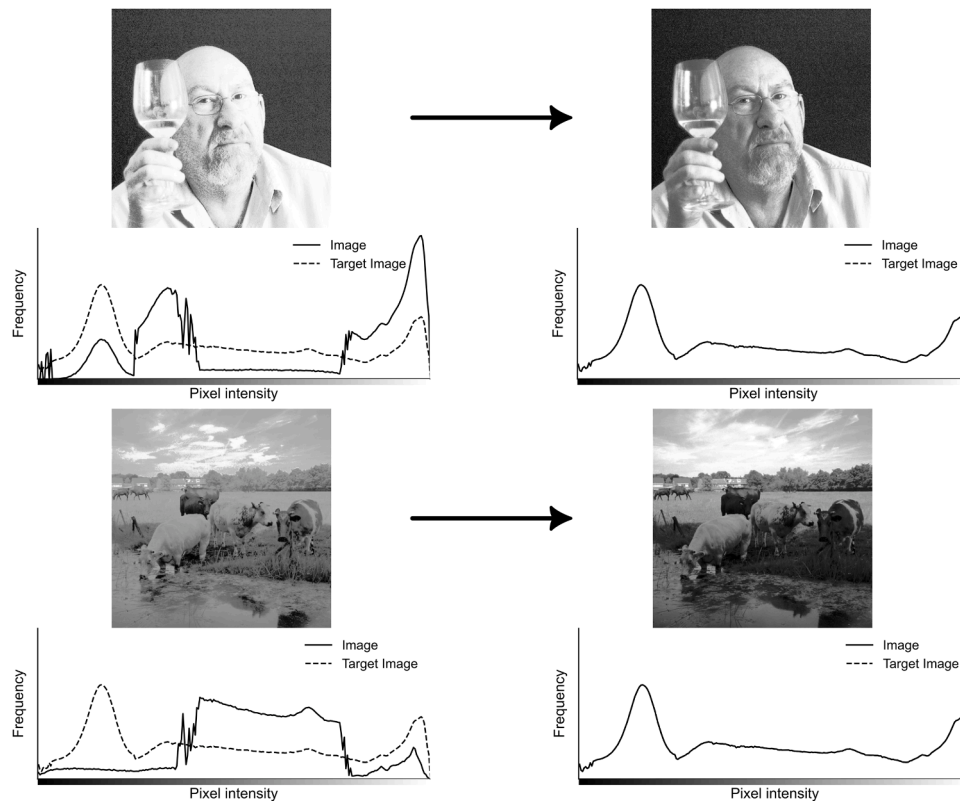


Fig. 2. Images and their luminance histograms before and after exact histogram specification using Gaussian-based tie-breaking with noise fallback. The target histogram corresponds to the average histogram of the input images. Initial Total Variation Distance (TVD) values of 0.4542 and 0.4409 decreased to zero after processing (TVD = 0 against the pixel-constrained target histogram).

required. The package supports both programmatic use and a command-line interface, making these methods accessible to users with varying levels of coding experience.

SHINIER was designed with modularity, extensibility, and reproducibility in mind, enabling future extensions such as video processing and additional methods for controlling low-level image properties. By integrating naturally within the scientific Python ecosystem and providing an open alternative to proprietary low-level image-processing workflows, SHINIER may improve methodological consistency and reproducibility across studies through the standardization of common image-manipulation procedures.

AI usage disclosure

Several Large Language Models (LLMs)—mainly ChatGPT, Claude, and Gemini—were used to assist in the following tasks: implementing the official style guide for Python code (PEP8); assisting in improving the OOP architecture; writing boilerplate code for classes and functions; providing insights for optimization; writing docstrings and API documentation; writing unit and validation tests. All proposed ideas and code were thoroughly examined, tested, and validated by the authors.

CRedit authorship contribution statement

Mathias Salvas-Hébert: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Conceptualization. **Nicolas Dupuis-Roy:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Conceptualization. **Catherine Landry:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Ian Charest:** Writing – review & editing, Writing – original draft, Supervision, Resources, Methodology, Conceptualization. **Frédéric Gosselin:** Writing – review & editing, Writing – original draft, Supervision, Software, Resources, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank the original contributors of the SHINE toolbox.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at [doi:10.1016/j.softx.2026.102884](https://doi.org/10.1016/j.softx.2026.102884).

References

- [1] Allen EJ, St-Yves G, Wu Y, Breedlove JL, Prince JS, Dowdle LT, Nau M, Caron B, Pestilli F, Charest I. A massive 7T fMRI dataset to bridge cognitive neuroscience and artificial intelligence. *Nat Neurosci* 2022;25(1):116–26. <https://doi.org/10.1038/s41593-021-00962-x>.
- [2] Willenbockel V, Sadr J, Fiset D, Horne GO, Gosselin F, Tanaka JW. Controlling low-level image properties: the SHINE toolbox. *Behav Res Methods* 2010;42(3):671–84. <https://doi.org/10.3758/BRM.42.3.671>.
- [3] Dal Ben R. SHINE_color: controlling low-level properties of colorful images. *MethodsX* 2023;11:102377. <https://doi.org/10.1016/j.mex.2023.102377>.
- [4] Coltuc D, Bolon P, Chassery J-M. Exact histogram specification. *IEEE Trans Image Proc* 2006;15(5):1143–52. <https://doi.org/10.1109/TIP.2005.864170>.
- [5] Avanaki AN. Exact global histogram specification optimized for structural similarity. *Opt Rev* 2009;16:613–21. <https://doi.org/10.1007/s10043-009-0119-z>.
- [6] Rahman H, Shimamura T. Tripartite image decomposition-based histogram equalization to enhance slightly low-contrast and low-contrast images. *ICIC Express Lett* 2026;20(3):321–32. <https://doi.org/10.24507/icicel.20.03.321>.
- [7] Rahman H, Mostofa S, Akter T, Rashedunnabi AHM. Efficient enhancement of images using recursive dualistic fuzzy histogram equalization. In: 2026 IEEE 2nd International Conference on Quantum Photonics, Artificial Intelligence, and Networking (QPAIN); 2026. p. 1–6. <https://doi.org/10.1109/QPAIN69676.2026.11546014>.
- [8] Rahman H. A time-efficient and effective image contrast enhancement technique using fuzzification and defuzzification. In: Mahmud M, Kaiser MS, Bandyopadhyay A, Ray K, Mamun SA, editors. *Proceedings of trends in electronics and health informatics* (Lecture notes in networks and systems, vol. 1034, pp. 45–58). Springer Nature; 2025. https://doi.org/10.1007/978-981-97-3937-0_4.
- [9] Rahman H. Bi-entropy curve equalization for enhancement of images. In: 2025 7th International Conference on Electrical Information and Communication Technology (EICT). IEEE; 2025. p. 1–6. <https://doi.org/10.1109/EICT68394.2025.11355632>.
- [10] Rahman H, Sugitara Y, Shimamura T. Enhancement of low-light images using Sakaguchi-type function-based cost-effective filtering. *Pattern Anal Appl* 2025;28:193. <https://doi.org/10.1007/s10044-025-01578-8>.
- [11] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. *IEEE Trans Image Process* 2004;13(4):600–12. <https://doi.org/10.1109/TIP.2003.819861>.
- [12] Floyd RW, Steinberg L. An adaptive algorithm for spatial grayscale. *Proc Soc Inf Disp* 1976;17(2):75–7.
- [13] Allard R, Faubert J. The noisy-bit method for digital displays: converting a 256 luminance resolution into a continuous resolution. *Behav Res Methods* 2008;40(3):735–43. <https://doi.org/10.3758/BRM.40.3.735>.
- [14] CIE. Commission internationale de l'Éclairage proceedings, 1931. Cambridge University Press; 1932.
- [15] CIE. CIE publication no. 015:2004: colorimetry. Central Bureau of the CIE; 2004.
- [16] International Telecommunication Union. Recommendation itu-r BT.601-7: studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. ITU; 2011.
- [17] International Telecommunication Union. Recommendation itu-r BT.709-6: parameter values for the hdtv standards for production and international programme exchange. ITU; 2015.
- [18] International Telecommunication Union. Recommendation itu-r BT.2020-2: parameter values for ultra-high definition television systems for production and international programme exchange. ITU; 2015.
- [19] International Electrotechnical Commission. Multimedia systems and equipment—Colour measurement and management—Part 2-1: colour management—Default rgb colour space—sRGB (IEC 61966-2-1:1999). IEC; 1999.
- [20] Mansencal T, Mauderer M, Parsons M, Shaw N, Wheatley K, Cooper S, Vandenberg JD, Canavan L, Crowson K, Lev O, Leinweber K, Sharma S, Sobotka TJ, Moritz D, Pppp M, Rane C, Eswaramoorthy P, Mertic J, Pearlstine B, Schmidt L. Colour (Version 0.4.7) [Software]. Zenodo; 2025. <https://doi.org/10.5281/zenodo.17837391>.
- [21] The MathWorks Inc. MATLAB (Version R2025a) [Computer software], <https://www.mathworks.com>; 2025.
- [22] Srinath K. Python—the fastest growing programming language. *Int Res J Eng Technol* 2017;4(12):354–7.