Performance Evaluation of Digital Image Processing by Using Scilab

Rudi Heriansyah[1], Wahyu Mulyo Utomo[2]

[1]*Computer Engineering Technology Section, Malaysian Institute of Information Technology, Universiti Kuala Lumpur, Malaysia*

[2]*Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn, Johor, Malaysia*

[1]rudi@unikl.edu.my, [2]wahyu@uthm.edu.my

Abstract - Scilab is an open-source, cross-platform computational environment software available for academic and research purposes as a free of charge alternative to the matured computational copyrighted software such as MATLAB. One of important library available for Scilab is image processing toolbox dedicated solely for image and video processing. There are three major toolboxes for this purpose: Scilab image processing toolbox (SIP), Scilab image and video processing toolbox (SIVP) and recently image processing design toolbox (IPD). The target discussion in this paper is SIVP due to its vast use out there and its capability to handle streaming video file as well (note that IPD also supports video processing). Highlight on the difference between SIVP and IPD will also be discussed. From testing, it is found that in term of looping test, Octave and FreeMat are faster than Scilab. However, when converting RGB image to grayscale image, Scilab outperform Octave and FreeMat.

Keywords: Scilab, image processing, video processing, computational free software

I. INTRODUCTION

Scilab was created in 1990 by researchers from INRIA and ENPC as a free platform for academia to support their daily research activities [1]. Since May 20013, in order to broaden its contributions and promoting Scilab as worldwide reference software in academia and industry, the Scilab Consortium has been established. Since July 2012, Scilab is developed and published by Scilab Enterprises which was created by the Consortium for developing and marketing directly through an international network or affiliated service providers. Scilab Enterprises also has partnership with Equalis since September 2010 in providing Scilab Online Support (SOS) Services.

Scilab was written in C/C++, Java and Fortran that works under GNU/Linux, Windows, Mac OS X and BSD operating systems. The discussion in this paper is based on the release is 5.4.1. The software is also available in several foreign languages rather than English, such as German, Spanish, Russian, Chinese, and so on. Scilab resembles MATLAB [2] in syntax and using matrices as the main data types. Xcos is a free package of Scilab for modeling and simulation of explicit and implicit dynamical systems that compared to Simulink in MATLAB. Scilab also includes a translator for the code conversion from MATLAB to Scilab.

External developers have contributed to many functionalities modules/toolboxes for Scilab under the name ATOMS (AuTomatic Modules Management for Scilab). This ATOMS console manages for installing, uninstalling and updating purposes. These modules cover many aspects of applications, for instance aerospace, data analysis and statistics, mathematics and optimization, graphics, image and signal processing, control, physics, and real-time libraries.

Image processing module in Scilab comes from three different developers. The first recognized module is Scilab Image Processing (SIP) toolbox developed by Ricardo Fabbri [3]. It has a very extensive 74 built-in functions to manipulate image, color image processing [4], filtering, edge detection, segmentation, transformation, morphological operations, shape analysis and contrast manipulation [5]. However, it cannot handle streaming images or video file. It was also fully targeted to work under GNU/Linux operating systems. Hence up to its Linux SIP 0.5.6 version, there is no stable release of the toolbox that work well under Windows operating system.

In order to cover the deficiency of SIP in handling video file, the Scilab Image and Video Processing (SIVP) toolbox has been introduced. Although SIVP has less functionality compared to SIP, but it can handle streaming images or video file and also work well under Windows operating system [6]. Recently, alternative for image processing working under Scilab is Image Processing Design (IPD) toolbox [7]. SIVP and IPD almost have similar functionalities and IPD also works well under Windows. This paper puts attention more on

SIVP due to its large use in academic research. The discussion will be based on SIVP version 0.5.3.1-2.

Background for this writing is to explore Scilab as a free computational tool and will focus on image processing modules as conducted by [8], [9], [10]. This study is important to provide users on the free alternative when they do not have access to the paid licensed software. Comparative study also will be carried out with others free tools, such as Octave and Freemat. Next section discusses many aspects of SIVP, including its available built-in functions and categorization of these functions according to their usage. Practical uses of these functions discuss in Section 3 in which shall be concluded in Section 4. List of references are given at the end of the paper.

II. METHOD

SIVP aims to give some free and fully functional libraries for image and video processing under Scilab environment. It has been ported to work under any Unix-like operating systems and also run smoothly under Microsoft Windows. It was written using C and TCL/TK programming languages. It was developed based on OpenCV (Open Source Computer Vision Library), a free library from Intel [11].

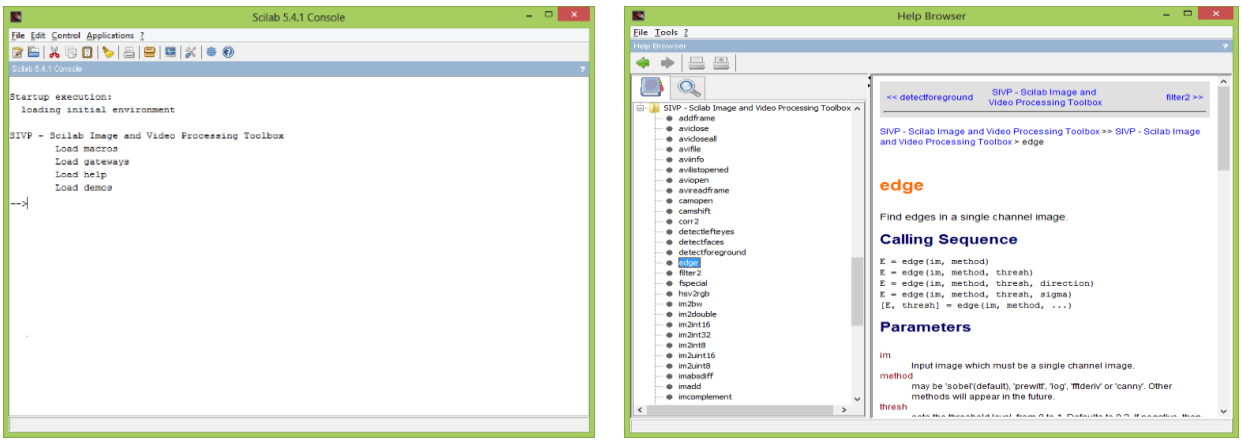
Basically, it has around 55 built-in functions for the various use of processing image and video. It can be then divided into few main categories: image I/O (3 functions), image data type conversions (6 functions), image color conversions (7 functions), spatial transformation (4 functions), image analysis and statistics (16 functions), image arithmetic (6 functions), linear filtering (3 functions) and 11 functions for video processing.

The easiest way to install SIVP is through Scilab interface. From Applications menu, click sub-menu Module manager – ATOMS. A new window will launch that lists all available modules. Click Image Processing module, it will list two modules Image Processing Design Toolbox (IPD) and Scilab Image and Video Processing Toolbox (SIVP). Click at the suitable SIVP module and click button [Install] to start installing process of SIVP. Wait until Scilab finish in installing SIVP module and after that quit from Scilab environment. Launch Scilab again and it should show SIVP module loaded at the Scilab console as in Fig. 1(a). Also note that the icon of SIVP module in ATOMS window will change to green color once it has been successfully installed. Another method is by installing the package independently from outside Scilab environment. To see SIVP functions, just click Scilab Help menu and then go down to SIVP installed module and expand the tree to see all available functions as shown in Fig. 1(b).

III. RESULTS AND DISCUSSION

To start our image processing journey in Scilab using SIVP, first load any image into computer memory. SIVP read image function support various type of images to read, such as Windows bitmaps (BMP, DIB), JPEG files (JPEG, JPG, JPE), Portable Network Graphics (PNG), Portable image format (PBM, PGM, PPM), Sun rasters (SR, RAS) and TIFF images (TIFF, TIF). The read image is shown in Fig. 2. This window enables the user to zoom-in and zoom-out the image.

To observe the active folder can use the pwd command in the console window. To know SIVP folder, can execute this getSIVPpath() function. There are few functions to manipulate colors, basically to convert RGB image into grayscale, HSV, NTSC, YCbCr image and vice-versa.



(a) **(b)**

Fig. 1 Installing SIVP: (a) SIVP module loaded; (b) SIVP module loaded

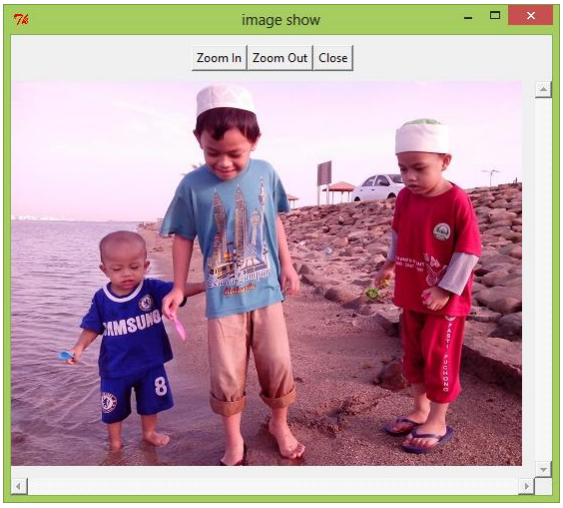


Fig. 2 Load an image

Fig. 3 shows various color image types after executing rgb2gray(), rgb2hsv(), rgb2ntsc() and rgb2ycbcr() functions respectively. Any of these color types can be converted back into RGB by using available built-in functions, for example to convert HSV image into RGB image, just employ hsv2rgb() function.

To convert a graysscale image into binary image [12], im2bw() function can be used as well. Grayscale image is basically an image with few levels of gray shade [13]. For a 8-bit system, there are 256 gray levels, in which the lowest level is black color at 0 intensity and the highest level is 255 indicate a white color. In contrary, a binary image has only two gray levels or two colors only, either black or white, 0 intensity is black and 1 is white. To convert from grayscale image to binary image, a threshold value should be determined. For example when threshold value is decided to be 0.5, then any intensities below 127 will convert to 0 or black color and any intensities higher or equal to 128 will be convert to 1 or white color in binary image. Fig. 4(a) shows a binary image by setting threshold value equal to 0.5 by employing im2bw() function.

Image complement is an image that has an opposite intensity from the original image. In the case of binary image, black and white pixel will become white and black pixel respectively for the complement image. Observe image in Fig. 4(b) which is the complement image of Fig. 4(a). It is obvious from the figure that it has an opposite intensities for every pixels of the original binary image. The function imcomplement() can be used for this purpose. In the case of intensity image, complement image obtained by subtracting each pixel with the maximum pixel value of the original image.

It is also worthy to note that image complement also simply can be obtained by put the operator "~" in front of original image, for instance this expression ~img will convert the original image into a complement image. Fig. 4(c) shows this complement image from a RGB image.

Another useful features to coloring image is by using built-in colormap in Scilab. There are at least 15 built-in colormaps, such as bonecolormap(), hotcolormap(), jetcolormap() and so on. To use this colomap, the function ind2rgb() can be used in which it requires two parameters the indexed image and the designated colormap. Hence, any color images should be then converted first into an indexed image (grayscale also considered as an indexed image). For example, this code indImg = ind2rgb(img, cmap) in which cmap = jetcolormap(256) will map the original image into a jet color map by executing imshow(indImg) as shown in Fig. 4(d).

A. Image Filtering & Detection

There are at least three built-in functions to perform filtering on image using SIVP. The functions imfilter() and filter2() will perform filtering based on input parameters. These functions requires two input parameters: input image and any filter that want to be applied. Another function fspecial() will create some 2D special filters, such as sobel, prewitt, gaussian, laplacian, log, average and unsharp filters [14]. Note that the only difference of imfilter() and filter2() is the output of filter2() is double matrix and the output of imfilter() has the same type as input and the elements in the output matrix that exceed the range of the integer type will be truncated.

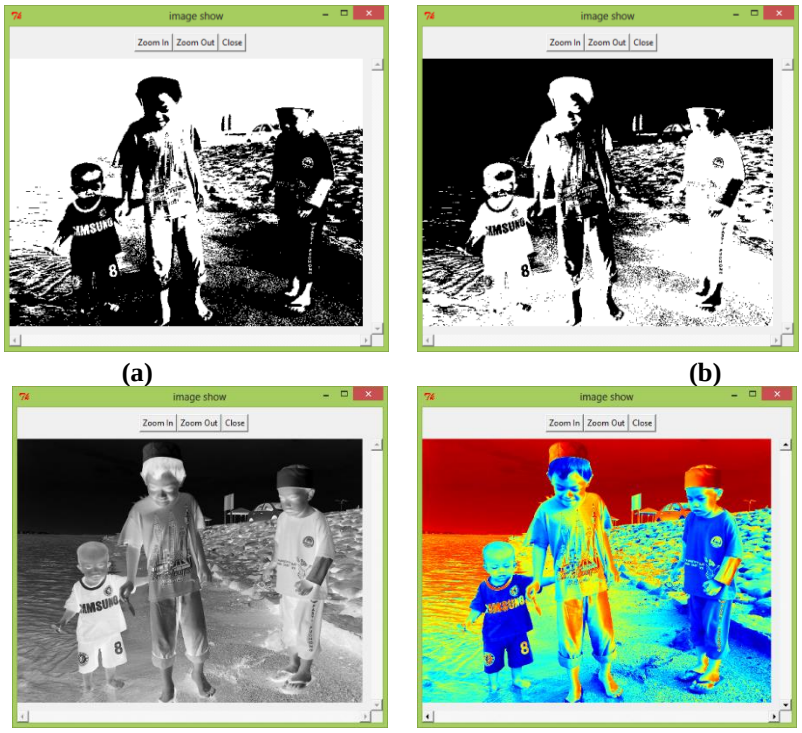
The use of the filter function is straight forward. First step is to determine or create the filter that want to be used. After that pass the created filter to either one available imfilter() or filter2() functions. The image filter with sobel shown in Fig. 5. By default, all these filter operators are working in RGB images directly.

As for edge detection, the source image must be a single channel or grayscale image otherwise an error will be occured if the input image is multi-channel or color image. There are five edge detectors available: sobel, prewitt, laplacian of gaussian [15], FFT gradient and canny edge detector. The built-in function edge() requires four input parameters: source image, detector type, threshold value and direction of the edge detector. The first two is compulsory and the last two if not provided, the default values will be used then.

(a) **(b)**

(c) **(d)**

Fig. 3 Various types of color images: (a) Grayscale image; (b) HSV image; (c) NTSC image; (d) YCbCr image



(a) **(b)**

(c) **(d)**

Fig. 4 Various types of intensity images: (a) Binary image; (b) Image complement of (a); (c) Complement of grayscale image; (d) Indexed image with jet colormap

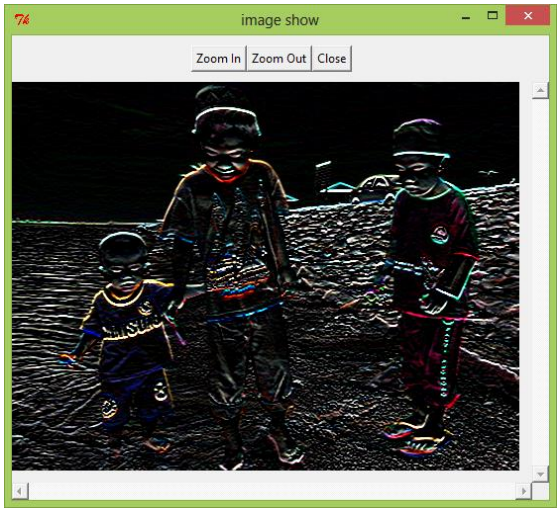


Fig. 5 Image filtering with Sobel filter

```
img = imread('siblings.jpg');
imgray = rgb2gray(img);
edgeS = edge(imgray, 'prewitt');
imshow(edgeS);
```

Fig. 6 Code for edge detection operation

Fig. 7(a) shows an implementation of the edge detection by using prewitt detector [16] by using the following code in Fig. 6. The canny detector [17] implementation is shown in Fig. 7(b). To clarify the detected edges, one can invert any edge image by applying imcomplement() function or just simply negate the edge image using "~" operator as shown in Fig. 7(c).

(a) (b) (c)

Fig. 7 Edge detection: (a) Prewitt edge detection; (b) Canny edge detection; (c) Complement of (c)

B. Image Arithmetic

Another interesting built-in function in SIVP is functions to do image arithmetic. Using these functions, it is possible to do some mathematical based operation to any images, for example to do some addition, subtraction, division, multiplication and also to calculate absolute difference between two images.

Each function basically will work at pixel level or point operation level in which two input parameters are required. The first parameter usually an input image and the second parameter can be also an image or double scalar. If both input are images, then the size of both images must be similar and color channels must be the same as well. Hence, it is not possible to work if the first image is RGB and the second image is grayscale. Rather than image as the second input parameter, applying double scalar or numerical value is also possible. For example in the case of image addition using imadd() function, adding scalar will either darker or brighter pixels of the output image. Fig. 9 shows two images for image addition operation. The first image is in Fig. 9(a) and the second image is in Fig. 9(b). The output image is shown in Fig. 9(c) which now becomes combination of both images. Code implementation of image addition is shown below in Fig. 8.

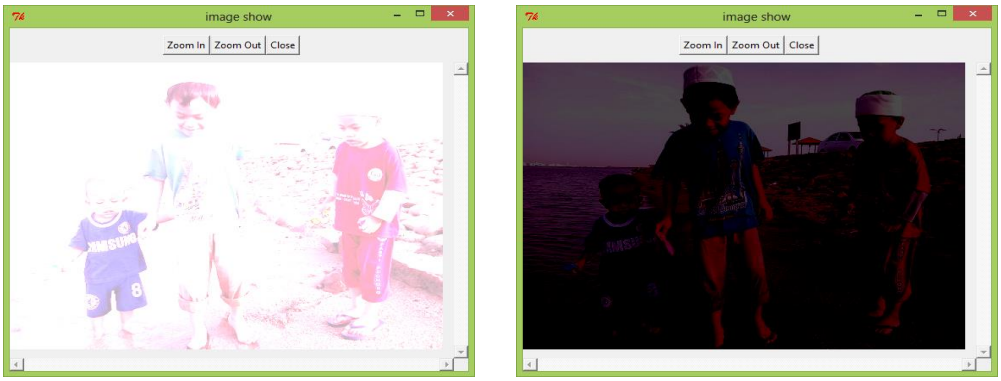
Fig. 10 shows image addition operation by involving scalar. The original image is as in Fig. 9(b). The code is still similar as in Fig. 9(a), however now `img2` is replaced with a numerical value. Fig. 10(a) and (b) are the output images when adding the input image with 180 and -180 respectively. It is obvious then adding positive constant will make the output image brighter and adding by negative number will make the output image darker. Therefore, adding 0 to the input image will give no effect at all.

```
img1 = imread('siblings.jpg');
img2 = imread('redsea.jpg');
ims = imadd(img1, img2);
imshow(ims)
```

Fig. 8 Code for image addition operation

(a) **(b)** **(c)**

Fig. 9 Image addition of two images: (a) first image; (b) second image; (c) output image



(a) **(b)**

Fig. 10 Image addition by scalar to original image: (a) Adding 180; (b) Adding −180

C. Video Processing

The power of SIVP is laid on its capability to handle video stream. There are few built-in functions to handle these streaming images. In this concern, at least there are three important functions for this purpose: `aviopen()`, `avreadframe()`, `avclose()`. Anytime when an open event is not employed anymore by the program, it then should be closed properly to save the usage of computer memory.

D. SIVP versus IPD Toolbox

SIVP and IPD although almost have the same functions, however there few functions in which they are different and this actually complement each other. There are few functions in IPD which are not available in SIVP, for example the functions for Otsu thresholding [18], morphological operations, texture energy, wavelet and watershed based segmentation. Also note that installing IPD is straight forward and basically just similar as installing SIVP as already described in Fig. 1.

Fig. 11(a) is a binary image after thresholding Fig. 9(a) with famous Otsu method [19] using CalculateOtsuThreshold() function. Fig. 11(b), (c), (d) are images after applying erosion, dilation and bottom hat filter to the original image. These are among important functions in analyzing binary image properties or behaviours. One may note that in showing image, IPD uses default Scilab window that allow to open few windows simultaneously in which SVIP with its Tk-based window cannot do. This multiple windows show is very useful in case one need to analyze many images at the same time and really make life of the user easier.

However, SIVP uses only one function imshow() to show image for whatever type (grayscale or color), in which IPD uses two function separately ShowImage() and ShowColorImage() respectively for this purpose. By this respect, SIVP seems more flexible rather than IPD.

E. Performance Evaluation

Previous sections give some clear ideas in how to work with SIVP in Scilab development environment. Various examples in processing images have been shown that indicates the effectiveness of the library and the benefit of working in Scilab environment. Few highlights also have been shown on the differences between two image processing toolboxes: SIVP and IPD.

The user can decide then in which toolbox that more suitable for their daily image processing works.

This section conducts further studies to explore performance of several freeware for image processing. Freeware interest is important since it will give all access

to image processing libraries at no cost and promptly. So then, all level of communities can access and get benefits from such software. Another two image processing freeware will be compared with Scilab, i. e. Octave and Freemat due to its popularity among the communities.

Octave is well-known due to its MATLAB-like interface, syntax and internal commands [20]. Freemat although is a new competitor in this area, but it has a quite comprehensive library [21]. Performance evaluation of these image processing freeware is carried out in term of speed processing, since it would be a fairer comparison to understand the effectiveness of the internal routines of the freeware in processing the data internally. The Intel® Core i7 2.4GHz with 8GB DDR3 RAM are used to serve this purpose.

The tests are conducted by using some built-in functions with the same purposes. There are four tests that have been performed. The first test is by applying a looping test in which each software need to loop for one million times. This is a very basic test to give some general idea on the speed performance of each software broadly. The second test is an extension of the first test in which now by printing simple text at the console window of the application. The last test is dealing with image processing conversion from RGB to grayscale. The size of the image is 1536×2048 in height and width respectively. Pixel intensity averaging is used for conversion. In which each R, G, B pixels will be divided by number of 3.

Table I shows the results for this performance evaluation. For the first two tests, indicates that the Scilab around 50% slower than the other two comparators. These tests are simply straightforward looping operation in which the interpreter need to do loop for one million times.

However, for the last test, when converting RGB image into grayscale image, it is obvious that the Scilab outperforms the other two. Although this test is not comparing apple-to-apple of the built-in image processing functions in all software, such as for image reading and displaying, but it gives a clear idea on the capability of Scilab to process arrays at the real time, which is the main function in processing images.

It only takes 111.392s for Scilab to do this averaging operation, in which it was 145.69s and 621.399s for Octave and Freemat respectively. At this sense, Scilab 30% faster than Octave and 556% or five times faster than Freemat.

Fig. 11 Some operations using IPD toolbox: (a) Otsu thresholding; (b) Eroded image; (c) Dilated image; (d) Bottom hat filtered image

TABLE I
PERFORMANCE EVALUATION

Test Set	Speed (seconds)		
	Scilab	Octave	Freemat
Looping one million times.	0.303	0.150	0.157
Looping one million times and write some text onto the hard disk.	106.501	57.791	52.051
Converting RGB into grayscale image (1536×2048)	111.392	145.69	621.399

IV. CONCLUSION

The existence of free image processing toolbox under Scilab environment, really helps so many researchers to do their research with peace of mind from copyright issues. Although, not each single toolbox can handle everything but they can synergize if one installs them all in his machine. SIVP and IPD work well under Windows and may give all their functionalities if installed together in Scilab environment. SIP provide more libraries (although does not support video processing) that will equip researchers with all necessaries for their daily research activities and also a really good option for Linux-based users. At the performance evaluation part as shown in Table I, it is obvious that Scilab has a very good performance in dealing with processing numerical two-dimensional arrays which is basically the main point in any image processing algorithms. Therefore, Scilab is one of the best alternative for image processing applications with not cost at all.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the Computer Engineering Technology Section, Malaysian Institute of Information Technology, Universiti Kuala Lumpur, Malaysia and Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia, Batu Pahat, Johor, Malaysia for providing any supporting facilities and conducive environment that makes this research and writing a smooth and joyful processes.

REFERENCES

[1] M. Affouf, *Scilab by Example*. CreateSpace Independent Publishing Platform, 2012.

[2] B. R. Hunt, R. L. Lipsman, J. M. Rosenberg, K. R. Coombes, J. E. Osborn, and G. J. Stuck, *A Guide to MATLAB: For Beginners and Experienced Users*. Cambridge University Press, 2006.

[3] R. Fabbri, O. M. Bruno, and L. da F. Costa, "Scilab and SIP for Image Processing," Mar. 2012.

[4] J. Q. Odeh, F. Ahmad, M. Othman, and R. Johari, "Image Retrieval System Based on Density Slicing of Colour Histogram of Images Subareas and Colour Pair Segmentation," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 196–202, 2004.

[5] J. Druel, "A SIP User Manual for SIP version 0.3 (rev. 1)," 2004.

[6] S. Yu and S. Shang, "SIVP – Scilab Image and Video Processing Toolbox," 2006.

[7] H. Galda, "Image Processing with Scilab and Image Processing Design Toolbox," 2011.

[8] J. S. Sohal, "Improvement of artificial neural network based character recognition system, using SciLab," *Optik (Stuttg).*, vol. 127, no. 22, pp. 10510–10518, 2016.

[9] R. Senthilkumar and R. K. Gnanamurhty, "Improvement and solution to the problems arise in the implementation of facial image recognition algorithms using open source software scilab," *World Appl. Sci. J.*, vol. 34, no. 12, pp. 1754–1761, 2016.

[10] S. Chopparapu and B. Seventline, "Object detection using Matlab, Scilab and Python," *Technology*, vol. 11, no. 6, pp. 101–108, 2020.

[11] A. Kaehler and G. Bradski, *Learning OpenCV 3*. O'Reilly Media, Inc., 2016.

[12] J. M. Kinser, *Image Operators: Image Processing in Python*. CRC Press, 2019.

[13] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[14] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A Practical Approach with Examples in MATLAB*. Wiley-Blackwell, 2012.

[15] S. L. Tanimoto, *An Interdisciplinary Introduction to Image Processing: Pixels, Numbers, and Programs.* Massachusetts Institute of Technology, 2012.

[16] P. Selvakumar and S. Hariganesh, "The performance analysis of edge detection algorithms for image processing," in *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, 2016, pp. 1–5.

[17] Z. Xu, X. Baojie, and W. Guoxin, "Canny edge detection based on Open CV," in *2017 13th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, 2017, pp. 53–56.

[18] A. McAndrew, *A Computational Introduction to Digital Image Processing*, 2nd ed. Taylor & Francis Group, LLC, 2016.

[19] N. Li, X. Lv, B. Li, and S. Xu, "An Improved Otsu Method Based on Uniformity Measurement for Segmentation of Water Surface Images," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications*

(GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2019, pp. 675–681.

[20] J. S. Hansen, *GNU Octave Beginner's Guide*. Packt Publishing, 2011.

[21] G. Schafer and T. Cyders, "The Freemat 4.0 Primer," 2011.