Computers in Biology and Medicine 84 (2017) 189–194

Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: www.elsevier.com/locate/compbiomed

IJ-OpenCV: Combining ImageJ and OpenCV for processing images in biomedicine

César Domínguez, Jónathan Heras*, Vico Pascual

Department of Mathematics and Computer Science, University of La Rioja, Logroño, Spain

ARTICLE INFO

Keywords:
ImageJ
OpenCV
Interoperability
Image processing
Computer vision
Machine learning
Biomedicine

ABSTRACT

Background and Objective. The effective processing of biomedical images usually requires the interoperability of diverse software tools that have different aims but are complementary. The goal of this work is to develop a bridge to connect two of those tools: ImageJ, a program for image analysis in life sciences, and OpenCV, a computer vision and machine learning library.

Methods. Based on a thorough analysis of ImageJ and OpenCV, we detected the features of these systems that could be enhanced, and developed a library to combine both tools, taking advantage of the strengths of each system. The library was implemented on top of the SciJava converter framework. We also provide a methodology to use this library.

Results. We have developed the publicly available library *IJ-OpenCV* that can be employed to create applications combining features from both ImageJ and OpenCV. From the perspective of ImageJ developers, they can use IJ-OpenCV to easily create plugins that use any functionality provided by the OpenCV library and explore different alternatives. From the perspective of OpenCV developers, this library provides a link to the ImageJ graphical user interface and all its features to handle regions of interest.

Conclusions. The IJ-OpenCV library bridges the gap between ImageJ and OpenCV, allowing the connection and the cooperation of these two systems.

1. Introduction

The analysis of images is instrumental in many life science fields and, in particular, in biomedicine [1]. Currently, the datasets of bioimages are growing exponentially, and due to the huge size of such datasets, visual inspection and manual measurement of the images is a time-consuming task that lacks sensitivity, accuracy, objectivity and reproducibility [2]. Hence, researchers need to rely on automatic or semi-automatic imaging techniques, provided by several software tools [3].

Since there is not a unique tool that can tackle every single aspect (acquisition, analysis, visualisation, and so on) of biomedical image processing, different programs are required [3]. Due to this fact, *interoperability* is emerging as an important issue [1,3–6], and the bioimaging community is working on making feasible the collaboration of some of the most popular packages [7,8]. Achieving such a collaboration might be challenging, due to the diversity of the tools, and requires a deep understanding and study of the involved technologies, and both a research and development effort. In this context, the SciJava project [8] plays a key role in bringing together projects like CellProfiler, ImageJ, Icy, KNIME, MiToBo, OMERO or MATLAB. The work presented in this paper uses SciJava as a frame to construct a bridge connecting two well-known free and open-source tools employed in bioimaging: ImageJ and OpenCV.

ImageJ [5] is an image-analysis tool that has been successfully employed to deal with many problems in life sciences [9–12]. There are several reasons for the success of this software: its easy-to-use interface, the ability to easily extend its functionality by means of plugins, the availability of plugins to solve a great variety of problems, and the macro system that captures the users' interactions allowing them to automate and reproduce their workflows. ImageJ mainly features image processing algorithms; however, it lacks other instrumental tools in image processing such as computer vision and machine learning methods. Hence, the development of ImageJ plugins that require the latter kind of algorithms needs either the from-scratch implementation of those methods or the connection with different external libraries.

OpenCV [13] is a widespread computer vision and machine learning library applied in a great variety of contexts, including life sciences [14–16]. The power of OpenCV relies on the huge amount

* Corresponding author.
E-mail addresses: jonathan.heras@unirioja.es (J. Heras), vico.pascual@unirioja.es (V. Pascual).

http://dx.doi.org/10.1016/j.compbiomed.2017.03.027
Received 29 November 2016; Received in revised form 28 March 2017; Accepted 28 March 2017

(more than 2500) of both classic and state-of-the-art computer vision algorithms provided by this library. OpenCV supplies algorithms for: image processing, feature detection, object detection, machine-learning, and video analysis. The major difficulties for employing OpenCV in life sciences are its usability and interactivity: OpenCV neither provides a by-default graphical interface or the functionality to interact with regions of interest (ROIs). This means that it is necessary to code the interaction with OpenCV, and this may be a problem for life scientists.

The aforementioned drawbacks of ImageJ and OpenCV could be tackled by combining these two systems. Recently, two libraries [17,18] have been developed allowing ImageJ users to take advantage of some OpenCV features, but not the other way around. In this work, we have developed a new free and open-source Java library called *IJ-OpenCV*[1] that allows the communication of ImageJ and OpenCV in both directions. This library brings to the table several benefits for both communities:

- ImageJ users can employ the wide variety of computer vision and machine learning algorithms available in OpenCV.
- ImageJ does not need to be connected with several third-party tools to explore different alternatives; instead, OpenCV common interfaces can be used to easily explore different algorithms.
- OpenCV developers do not need to implement the functionality to handle ROIs, but they can employ ImageJ's features for ROI management.
- ImageJ simple-to-use interface can be employed to interact with OpenCV programs, overcoming the lack of a by-default interface in OpenCV.
- OpenCV programs can be distributed as ImageJ plugins making its use and dissemination easier.

Therefore, it is our belief, that this bridge between systems can have a positive impact in both ImageJ and OpenCV communities, and avoid unnecessary duplications of efforts.

2. Methods

2.1. Implementation

IJ-OpenCV is a Java library that is built on top of the SciJava project [8]. On a technical level, the SciJava core components are a set of standard Java libraries for managing extensible applications. Socially, the SciJava initiative aims to achieve the cooperation of organisations, reuse code, and synergise wherever possible [8]. The SciJava Common Library is the ground floor of the ImageJ software [19].

SciJava provides a unified mechanism for defining plugins — extensions that add new features or behaviour to the software [19]. In the interoperability context, we can highlight SciJava's *converter plugins* that provide a general way of transforming data from one type to another. New converter plugins can be developed to extend SciJava's conversion capabilities, allowing objects of one type to be used as module inputs of a different type, in cases where the two types are conceptually analogous. An example of these converter plugins is provided by the ImageJ-Matlab library [6] that allows the conversion from ImageJ datasets to MATLAB matrices and viceversa. Similarly, an image converter between ImageJ and the Insight ToolKit (ITK) [20] greatly streamlines the use of ITK-based algorithms within ImageJ [21].

The foundations of the IJ-OpenCV library are also SciJava's converter plugins. Namely, we have defined two kinds of SciJava converter plugins: image converters and ROI converters.

The image converter plugins provided by IJ-OpenCV enable the conversion from ImageJ images and stacks of images, objects of the `ImagePlus` class, to OpenCV images and arrays of OpenCV images, implemented by the classes `Mat` and `MatVector` respectively, and viceversa. These conversions use the third-party JavaCV library [22].

The ROI converter plugins allow the conversion between ImageJ ROIs and OpenCV ROIs (the interested reader can consult the complete list of converters in the project webpage). Namely, we have defined plugins to convert from both a unique ROI and a list of ROIs of the same type. The IJ-OpenCV library supports all the kinds of ROIs available in OpenCV and the ones that are mostly used in ImageJ: rectangles, circles, polygons, lines, ellipses, and points. It is also worth mentioning that OpenCV sometimes uses different encodings to represent the same kind of ROI. For instance, a point can be represented as an object of the class `Point`, but also as an object of the class `Mat`; therefore, we have defined different converters for those cases. Another issue handled by these converters is whether the conversion is actually possible; for instance, in ImageJ, circular ROIs are represented by means of the class `OvalRoi` that allows the encoding of ovals in general, and circles in particular; however, OpenCV only works with circles; therefore, the converter in charge of the conversion from `OvalRoi` to `Circle` includes the functionality to check whether the instance of the `OvalRoi` class is an actual circle.

In the combining process achieved thanks to the converter plugins of IJ-OpenCV, the major contribution is a set of tools that allow ImageJ users (and respectively OpenCV users) to employ objects and results obtained with OpenCV (and respectively ImageJ) using a representation that is well-known for them — since it is ImageJ's (or OpenCV's) own representation. The main problems solved in this process were: handling different encodings to define the converters, fixing the necessary restrictions to avoid conversion problems, facilitating the extensibility and usability of the library (solved by using the `Converter` interface of SciJava), and making the library stable to changes in ImageJ and OpenCV (solved by employing the APIs provided by them).

2.2. Quality of the IJ-OpenCV library

In the development of the IJ-OpenCV library, we have taken into account the different quality criteria of the ISO/IEC 25010 [23] for software quality. These criteria include the following characteristics.

- *Functionality*. The IJ-OpenCV library provides the converters between ImageJ and OpenCV images and ROIs; this is the primary requirement to achieve the communication between these two systems.
- *Reliability*. IJ-OpenCV relies on three libraries (ImageJ, OpenCV, and SciJava Common) that have been employed and tested by large communities. Moreover, we have increased the reliability of IJ-OpenCV using the Java version of QuickCheck [24] to thoroughly test the IJ-OpenCV library. In the future, we plan to apply formal methods, as the ones applied in [25], to verify the complete correctness of the library.
- *Usability*. The IJ-OpenCV library has been documented, and we provide several examples explaining how to use it. Moreover, thanks to the use of the interface for converter plugins provided by SciJava, the user can easily learn how to employ IJ-OpenCV – all the converters are used in the same way; and, therefore, it is easy to infer from the examples and documentation how to use any of them. Moreover, anyone coding a SciJava module [8] can annotate `@Parameter` fields of the OpenCV types, and ImageJ images will be auto-converted.
- *Efficiency*. The time complexity of the ROI converters is either constant (in the case of individual ROIs) or lineal (in the case of lists of ROIs); and, the time complexity of the image converters is lineal.
- *Maintainability*. Thanks to the use of converter plugins of the SciJava Common library, the IJ-OpenCV library can be easily

[1] Available at https://github.com/joheras/IJ-OpenCV.

changed and extended with new converters. Regarding the stability of IJ-OpenCV, it has been implemented using the APIs provided by ImageJ and OpenCV; hence, as long as those APIs are not modified, IJ-OpenCV will keep working even if changes are made in the internal representations employed in ImageJ and OpenCV.

- *Portability*. IJ-OpenCV can be easily installed by using the Maven Central binary repository. Since, IJ-OpenCV is implemented in Java, it is a platform independent library.

2.3. A methodology to use IJ-OpenCV

We finish this section by presenting a methodology to use the IJ-OpenCV library in order to combine the features of OpenCV and ImageJ — a suite of ImageJ plugins has been developed employing this methodology, see the supplementary materials, and it can be easily downloaded and installed in ImageJ using the ImageJ Updater [19]. We will particularise this methodology to a concrete example in the next section.

Let us start by considering a common approach followed when analysing images in life sciences [26–28]. That approach consists of three steps: (1) the ROIs of the image are automatically detected applying an image processing or computer vision algorithm; (2) the ROIs are manually adjusted by the user; and finally, (3) measurements are performed over those ROIs. In some cases, ImageJ will not be able to carry out the first and third steps, and the same happens with OpenCV for the second step. However, we could use IJ-OpenCV to create a plugin, or more generally a program, that connects ImageJ and OpenCV, and has the desired functionality. The workflow of such a program could be summarised as follows:

1. The program takes as input an image loaded in the GUI of ImageJ.
2. The ImageJ image is transformed to an OpenCV image using the IJ-OpenCV library.
3. Using OpenCV algorithms, the ROIs are detected on the OpenCV image.
4. The OpenCV ROIs are transformed to ImageJ ROIs using the IJ-OpenCV library, and added to the original ImageJ image.
5. The user adjusts the selected ImageJ ROIs using the ImageJ features for managing ROIs.
6. The modified ImageJ ROIs are converted back to OpenCV ROIs using the IJ-OpenCV library.
7. OpenCV algorithms are used to perform measurements over the OpenCV ROIs.
8. The results are sent to ImageJ, and they are shown to the user using its GUI.

3. Results and discussion

In this section, we present how the IJ-OpenCV library and the methodology presented previously have been implemented in the development of AntibiogramJ: a tool for analysing images from disk diffusion tests [29]. Using this software as a running example, we illustrate the benefits provided by IJ-OpenCV.

Disk diffusion testing, known as antibiogram, is widely applied in microbiology to determine the antimicrobial susceptibility of microorganisms [30]. The measurement of the diameter of the zone of growth inhibition of microorganisms around the antimicrobial disks (see Fig. 1) in the antibiogram is frequently performed manually by specialists using a ruler. The diameters of the inhibition zones are then used to categorise the bacterial isolate as susceptible, intermediate or resistant to each antimicrobial drug tested according to the clinical breakpoints established by an international committee — there is also the category “Not Available” when the clinical breakpoints do not include the tested antibiogram. AntibiogramJ is a Java application, built on top of ImageJ, that semi-automatises this reading process from plate images by combining several features of ImageJ and OpenCV thanks to the IJ-OpenCV library.

The process to read plate images in AntibiogramJ can be split into the following phases: (P.0) load an image; (P.1) detect and crop the plate from the image; (P.2) adjust brightness and contrast; (P.3) detect the antimicrobial disks of the plate; (P.4) read the drug code written in the antimicrobial disks; (P.5) detect and measure the diameter of the zone of growth inhibition of microorganisms around the antimicrobial disks; and (P.6) categorise the bacterial isolate as susceptible, intermediate or resistant to each antimicrobial drug. It is worth mentioning that the plate images in AntibiogramJ can be captured using any camera device; hence, it is not sensible to use a fully automated process (even if it could be applied) — due to the huge variability of the captured images — and a semi-automated approach is more adequate.

In the aforementioned process, Phases (P.0), (P.2) and (P.5) have been implemented directly using features of ImageJ; and Phase (P.6) does not require image processing techniques; so, they will not be longer discussed here. On the contrary, Phases (P.1), (P.3) and (P.4) have required the combination of ImageJ and OpenCV.

Cropping the plate from the image, i.e. Phase (P.1), is a key pre-processing step since it allows the user, and the algorithms, to focus on the important part of the image. This functionality has been implemented following the methodology presented in Section 2.3 as follows. Steps **S.1**. and **S.2**. are common to all the programs following the methodology; so, let us focus on the other steps. The plate of the image (Step **S.3**.) is a circle (see Fig. 1) that can be detected using several algorithms implemented in OpenCV like Hough circles [31] or active contours [32]; in particular, we use Hough circles. However, the precision of the detected ROI may not be sufficient, and the user might need to manually correct such a region. Therefore, the OpenCV circle is transformed to an `OvalRoi` object of ImageJ and added to the original ImageJ image (Step **S.4**.). The `OvalRoi` object can be altered using the ImageJ interface (Step **S.5**), and once the region has been fixed, the user can crop the image using the ImageJ cropping functionality. This workflow is depicted in Fig. 2.

This process to crop the plate from an image shows some of the advantages of using IJ-OpenCV instead of only using OpenCV. The OpenCV library is designed to carry out tasks fully automatically (e.g. robotics [33] or quality control [34]); therefore, when OpenCV detects ROIs in an image, those ROIs are fixed and cannot be modified by the user. However, when analysing images in life science fields, it is common to manually adjust the detected ROIs as in the above example. Hence, if we implement Phase (P.1) only using OpenCV, we will need to implement the management of ROIs in OpenCV, an arduous task. Instead, we have combined OpenCV with ImageJ and used the excellent ROI management features provided by the latter tool.

In addition, the implementation of this Phase (P.1) also exhibits one of the benefits of using IJ-OpenCV instead of only using ImageJ. The functionality to detect circles in an image using the Hough transform for circles is not implemented in ImageJ; hence, the ImageJ developer would need to implement such an algorithm. In this case, the situation is not too problematic since there is an ImageJ plugin [35] that implements the desired algorithm; unfortunately, this is not always the case as we will show with the implementation of Phases (P.3) and (P.4) of AntibiogramJ.

The implementation of both Phases (P.3) and (P.4) of AntibiogramJ can be considered together, since they are closely related, and both follow the methodology presented in Section 2.3. Since the image is already available in AntibiogramJ from the previous steps, Steps **S.1**. and **S.2**. of the methodology can be skipped. The antimicrobial disk of the image (Step **S.3**.) are the white circles inside the plate (see Fig. 1), and they are detected in AntibiogramJ thanks to the combination of several OpenCV algorithms (including Gaussian blurring, change of color space from RGB to HSV, and

Fig. 1. Screenshot of AntibiogramJ. On the left, the image of a plate is shown with the diameter and category (**I**ntermediate, **S**usceptible, **R**esistant, **N**ot Available) associated with each antimicrobial disk.

adaptive thresholding). However, depending on the quality of the image, some false positive and false negative antimicrobial disk might occur; hence, the user might need to add or remove some antimicrobial disks. To achieve this functionality, the list of circles detected by the OpenCV algorithm is converted to a list of `OvalRois` and added to the ImageJ image (Step **S.4**.). The list of `OvalRois` can be altered using ImageJ's `ROI manager` (Step **S.5**). Once the list of antimicrobial disks have been fixed, AntibiogramJ invokes OpenCV to read the code written in the disks. This requires a conversion (Step **S.6**) from the list of `OvalRois` to a list of OpenCV circles. Subsequently, the codes are read by using a matching procedure that compares the disk against a database of disks using the ORB binary descriptor implemented in OpenCV (Step **S.7**). Finally, the results are shown to the user (Step **S.8**). The workflow of this process is provided in the supplementary materials.

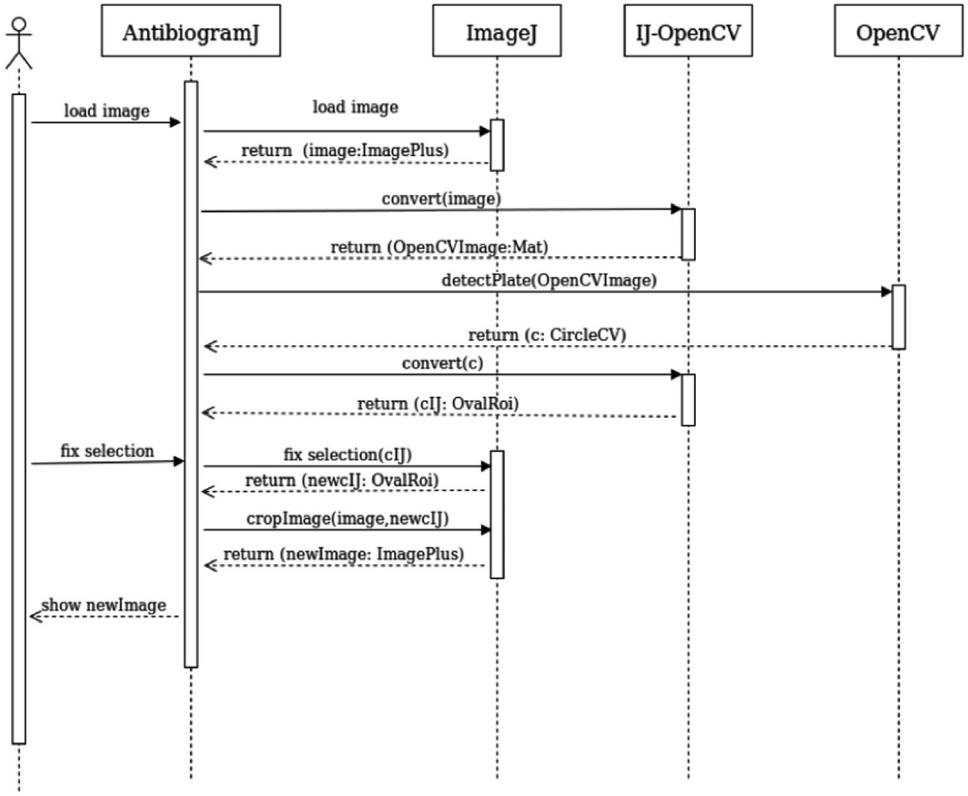

Fig. 2. Workflow to crop the plate from an image.

This example shows several benefits of using the IJ-OpenCV library instead of working only with ImageJ or OpenCV. First of all, some of the algorithms employed in the procedure (e.g. the ORB binary descriptor or the matching algorithm) are not available neither as a standalone feature of ImageJ or as a third-party ImageJ plugin. In this scenario, common in bioimaging problems that require computer vision and machine learning techniques (see, for instance, [36–39]), the ImageJ developer has two options: implement the computer vision and machine learning methods, or manually exporting data from ImageJ and importing them in a different program. Neither approach is fully satisfactory. In the former, the developer might, sometimes, "reinvent the wheel"; and, in the later, manually exporting and importing data across programs is a tedious and error-prone task. A more sensible alternative consists in connecting ImageJ with a library that supplies the desired functionality, as we have done here; and OpenCV is a good candidate since it provides a complete computer-vision and machine-learning suite, avoiding the implementation of standard algorithms and the manual exportation/importation of data across programs.

Another improvement that is introduced in ImageJ by connecting it with OpenCV, and that is shown with our running example, is related to the exploration of different alternatives. For instance, there are several algorithms for describing images using keypoint detectors (e.g. FAST, Harris, GFTT, MSER, ORB, SIFT, SURF [40]), but they are not directly implemented in ImageJ. Then, if an ImageJ user wants to analyse the results obtained using different detectors, ImageJ should be connected with several libraries and particularised for each one of them. On the contrary, OpenCV provides a common interface for keypoint detectors, and different algorithms can be tried by just changing the name of the algorithm to use. Therefore, the connection of ImageJ and OpenCV allows the developer to easily test different alternatives since OpenCV offers a wide variety of classical and state-of-the-art algorithms that share common representations across them. Hence, the task of trying different alternatives is reduced to change a few lines of code; and, in some cases, to just change a parameter of a method.

Due to the exponential growth, both in size and complexity, of datasets of images, reproducing results on those datasets might be difficult; especially, if it requires human corrections. However, manual correction and analysis are sometimes necessary and inescapable. For instance, adding and removing detected ROIs from an image is a common issue in several problems of life sciences (e.g. in our running example adding and removing antimicrobial disks, adding and removing bands from a DNA fingerprint image [26], or adding and removing synapses [27]). However, as we have previously commented in the case of cropping, OpenCV does not provide a by-default method to deal with that problem. Hence, instead of reinventing the wheel several times, OpenCV developers can use the IJ-OpenCV library and take advantage of ImageJ's `ROI manager`, that provides a simple way to deal with this problem.

Another place where OpenCV is improved thanks to the connection with ImageJ is its usability. As we have previously mentioned, OpenCV does not provide a by-default GUI, and developers are in charge of creating special-purpose interfaces for final users. Developing good GUIs is almost an art [41] because they must be easy-to-use and also easy-to-learn — the creation of such interfaces is not a simple task at all. The combination of OpenCV with ImageJ would solve this problem since OpenCV developers could develop their programs as ImageJ plugins and, then, use the GUI of this system (a well-known interface for life scientists) without implementing a new interface from scratch. Additionally, more special purpose interfaces can be built on top of ImageJ once the connection with OpenCV is achieved — this is the approach followed by AntibiogramJ.

As we have indicated in the Introduction, two other projects — called CVForge [17] and IJToolsUsingOpenCV [18] — connecting ImageJ and OpenCV were released almost at the same time that IJ-OpenCV. IJToolsUsingOpenCV provides a suite of ImageJ plugins that connects ImageJ with some of the algorithms implemented in OpenCV; however, the connection between ImageJ and OpenCV is achieved individually for each plugin, and there is not a generalisation (as in the case of IJ-OpenCV converters) that allows developers to easily extrapolate the approach followed by IJToolsUsingOpenCV. CVForge is an ImageJ plugin implementing a simple interface that gives access to all the methods of OpenCV. As IJ-OpenCV, the CVForge project provides image converters, but, in contrast to IJ-OpenCV, it does not implement ROI converters; hence, regions of interest detected by OpenCV algorithms cannot be handled using the ROI management features of ImageJ.

CVForge and IJToolsUsingOpenCV are focused on allowing ImageJ users to employ some of the features of OpenCV, but not the other way around. On the contrary, the IJ-OpenCV library provides the connection in both directions bringing several benefits to ImageJ and OpenCV communities. Therefore, it is our belief that IJ-OpenCV is the most complete library to connect these two systems.

4. Conclusions

IJ-OpenCV is a free and open-source library that allows the collaboration of ImageJ and OpenCV. This library has been successfully employed to construct a standalone application and several ImageJ plugins proving the benefits of connecting ImageJ and OpenCV. From our point of view, thanks to the communication provided by the IJ-OpenCV library, and the methodology presented in this paper to achieve it, the development efforts can be greatly reduced in both ImageJ and OpenCV communities when developing new tools for bioimaging.

As always when developing software, there is room for improvement. As further work, we are planning to introduce several features of the SciJava library in a future version of IJ-OpenCV. Namely, the suite of plugins provided in the supplementary materials will be refactorized as SciJava's Command plugins. Moreover, to achieve a greater extensibility and interoperability with other systems, a future version of the IJ-OpenCV library will employ the SciJava Convert Service to perform the conversions.

Availability and requirements.

- Project name: IJ-OpenCV.
- Project home page: https://github.com/joheras/IJ-OpenCV
- Operating system(s): Platform independent.
- Programming language: Java.
- License: GNU GPL 3.0.
- Any restrictions to use by non-academics: None.

The project home page contains the installation instructions and several plugins that have been developed using the IJ-OpenCV library.

Conflict of interest

All named authors hereby declare that they have no conflicts of interest to disclose.

Acknowledgements

This work was partially supported by the Ministerio de Economía y Competitividad (MTM2014-54151-P).

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.compbiomed.2017.03.027.

References

[1] C. Dietz, et al., Integrative open-source software enables image analysis in biological sciences, Photon. Int. 3 (2012) 35–38.
[2] E. Meijering, G. van Cappellen, Imaging cellular and molecular biological functions, Springer, Ch. Quantitative Biological Image Analysis, 2007, pp. 45–70.
[3] K.W. Eliceiri, et al., Biological imaging software tools, Nat. Methods 9 (7) (2012) 697–710.
[4] A.E. Carpenter, et al., A call for bioimaging software usability, Nat. Methods 9 (7) (2012) 666–670.
[5] J. Schindelin, et al., The ImageJ Ecosystem: an open platform for biomedical image analysis, Mol. Reprod. Dev. 82 (2015) 518–529.
[6] M.C. Hiner, C.T. Rueden, K.W. Eliceiri, ImageJ-MATLAB: a bidirectional framework for scientific image analysis interoperability, BioInformatics http://dx.doi.org/10.1093/bioinformatics/btw681.
[7] T. Pietzsch, et al., ImgLib2 — generic image processing in Java, Bioinformatics 28 (22) (2012) 3009–3011.
[8] C. Rueden, J. Schindelin, M. Hiner, K. Eliceiri, SciJava Common (Software), 2016 ⟨http://scijava.org/⟩.
[9] T. Abdulla, et al., Epithelial to mesenchymal transitionthe roles of cell morphology, labile adhesion and junctional coupling, Comput. Methods Prog. Biomed. 111 (2) (2013) 435–446.
[10] F. Ghasemian, et al., An efficient method for automatic morphological abnormality detection from human sperm images, Comput. Methods Prog. Biomed. 122 (3) (2015) 409–420.
[11] F. Piccinini, et al., Cancer multicellular spheroids: volume assessment from a single 2d projection, Comput. Methods Prog. Biomed. 118 (2) (2014) 95–106.
[12] E. Berry, A Practical Approach to Medical Image Processing, Series in Medical Physics and Biomedical Engineering, Taylor & Francis, 2007.
[13] A. Kaehler, G. Bradski, Learning OpenCV 3, O'Reilly Media, 2015.
[14] Y. Wang, et al., Polyp-alert: near real-time feedback during colonoscopy, Comput. Methods Prog. Biomed. 120 (3) (2015) 164–179.
[15] J. Sheng, S. Xu, X. Luo, Categorizing biomedicine images using novel image features and sparse coding representation, BMC Med. Genom. 6 (2013) S8.
[16] G. Islam, K. Kahol, Application of computer vision algorithm in surgical skill assessment, in: Proceedings of the IEEE 6th International Conference on Broadband Communications & Biomedical Applications (IB2Com'11), 2011, pp. 108–111.
[17] J. Martens, CVForge (Software), ⟨https://github.com/m4dguy/CVForge⟩, 2016.
[18] T. Nishida, IJToolsUsingOpenCV (Software), ⟨https://github.com/WAKU-TAKE-A/IJToolsUsingOpenCV⟩, 2016.
[19] C.T. Rueden, et al., ImageJ2: ImageJ for the next generation of scientific image data, CoRR arXiv:1701.05940, ⟨http://arxiv.org/abs/1701.05940⟩.
[20] T. Yoo, et al., Engineering and algorithm design for an image processing API: a technical report on ITK-the insight toolkit, Stud. Health Technol. Inform. 85 (2002) 586–592.
[21] C. T. Rueden, et al., ImageJ-ITK (Software), ⟨https://imagej.net/ITK⟩, 2016.
[22] S. Audet, et al., JavaCV (Software), ⟨https://github.com/bytedeco/javacv⟩, 2015.
[23] ISO, International software quality standard, ISO/IEC 25010, Systems and Software Engineering-Systems and software Quality Requirements and Evaluation (SQuaRE), 2011.
[24] T. Jung, QuickCheck for Java, ⟨https://bitbucket.org/blob79/quickcheck⟩, 2015.
[25] J. Heras, et al., Verifying a platform for digital imaging: a multi-tool strategy, in: Proceedings of Conferences on Intelligent Computer Mathematics 2013 (Calculemus track), vol. 7961 of Lecture Notes in Computer Science, Springer, 2013, pp. 66–81.
[26] J. Heras, et al., GelJ – a Tool for Analyzing DNA Fingerprint Gel Images, BMC Bioinformatics 16 (270).
[27] G. Mata, et al., SynapCountJ: A tool for analyzing synaptic densities in neurons, in: Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BioImaging'16), vol. 2, ScitePress, 2016, pp. 25–31.
[28] E. Meijering, et al., Design and validation of a tool for neurite tracing and analysis in fluorescence microscopy images, Cytom. Part A 58 (2) (2004) 167–176.
[29] C. Domínguez, et al., AntibiogramJ: a tool for analysing images from disk diffusion tests, ⟨https://sourceforge.net/projects/antibiogramj/⟩, 2016.
[30] S. Jenkins, A.N. Schuetz, Current concepts in laboratory testing to guide antimicrobial therapy, Mayo Clin. Proc. 87 (3) (2012) 290–675.
[31] H.K. Yuen, et al., Comparative study of hough transform methods for circle finding, Image Vision. Comput. 8 (1) (1990) 71–77.
[32] M. Kass, A. Witkin, D. Terzopoulos, Snakes: active contour models, Int. J. Comput. Vision. 1 (4) (1988) 321–331.
[33] A. Velayudhan, T. Gireeshkumar, An autonomous obstacle avoiding and target recognition robotic system using kinect, in: Proceedings of International Computing, Communication and Devices (ICCD'14), IEEE, vol. 1 of Advances in Intelligent Systems and Computing, Springer, 2015, pp. 643–649.
[34] M.M.G. Ramirez, J.C.V. Rincon, J.F.L. Parada, Liquid level control of coca-cola bottles using an automated system, in: Proceedings of IEEE International Conference on Electronics, Communications and Computers (CONIELECOMP'14), 2014, pp. 148–154.
[35] H. Pistori, E. Rocha, Hough Circles Plugin for ImageJ, ⟨https://imagej.nih.gov/ij/plugins/hough-circles.html⟩, 2009.
[36] G. Mata, et al., Automatic detection of neurons in high-content microscope images using machine learning approaches, in: Proceedings of the 13th IEEE International Symposium on Biomedical Imaging (ISBI'16), IEEE, 2016, pp. 330–333, http://dx.doi.org/10.1109/ISBI.2016.7493276.
[37] V. Kayniga, B. Fischerb, E. Müllerc, J.M. Buhmann, Fully automatic stitching and distortion correction of transmission electron microscope images, J. Struct. Biol. 171 (2) (2010) 163–173.
[38] S. Preibisch, S. Saalfeld, P. Tomancak, Globally optimal stitching of tiled 3D microscopic image acquisitions, Bioinformatics 25 (2009) 1463–1465.
[39] V. Kaynig, T. Fuchs, J.M. Buhmann, Neuron geometry extraction by perceptual grouping in sstem images, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'10), 2010, pp. 2902–2909.
[40] T. Tuytelaars, K. Mikolajczyk, Local Invariant Feature Detectors: A Survey, Now Publishers Inc., 2008.
[41] J. Anderson, et al., Effective UI: The Art of Building Great User Experience in Software, O'Reilly Media, 2010.