Cell Image Analyzer - A visual scripting interface for ImageJ and its usage at the microscopy facility Montpellier RIO Imaging

Volker Baecker[a] and Pierre Travo[a]

[a]Montpellier RIO Imaging, CNRS, 1919, route de Mende, 34293 MONTPELLIER CEDEX 5, France

ABSTRACT

A rapid image analysis application development framework called "Cell Image Analyzer" based on ImageJ is presented. It adds a visual scripting interface to ImageJ's capabilities, which allows creating applications from existing operations by drag and drop. It provides support to create batch applications as well as interactive applications. "Cell Image Analyzer" is used at the microscopy facility Montpellier RIO Imaging to create custom image analysis applications and to solve automation tasks. The applications include the topics "DNA combing", "quantification of stained proteins in cells", "comparison of intensity ratios between nuclei and cytoplasm" and "counting nuclei stained in different channels".

Keywords: ImageJ, microscopy, image analysis, visual programming, rapid prototyping, automation

1. INTRODUCTION

Modern techniques in microscopy allow biologists to acquire large amounts of data. The analysis of the data often remains a time consuming task. When done manually results might be involuntary biased and not reproducible. The analysis needed on the one hand and the image qualities on the other hand vary widely between experiments and research groups. The knowledge of the biologist can often facilitate the analysis of the images. In cases where a full automatic treatment is not possible with the desired accuracy for the time being, partial automation can help to work more efficiently. Standard image analysis applications are often not apt for the automation of specific tasks. They are not flexible enough to take experiment specific knowledge into account easily or to adapt the workflow in the desired way.

At Montpellier RIO Imaging we use the following approach: Specific solutions for analysis and automation problems are developed on demand in close collaboration with the biologists. The development is based on a rapid prototyping framework for image analysis applications. If necessary for the realization of a project and only then, the framework is expanded, in a modular way. This way, the framework grows iteratively with each project and all development efforts serve an immediate purpose.

To provide the framework we developed a software called "Cell Image Analyzer" that is based on ImageJ.[1] The basic addition to ImageJ is a visual scripting interface[2] that allows creating applications from existing operations by using drag and drop. New basic operations can be added to the framework on the programming level either by wrapping existing ImageJ operations or from scratch. It will be possible to use the visual scripting as a plugin in a standard ImageJ installation, as well. A number of applications has been successfully developed with "Cell Image Analyzer" by now.

2. MATERIALS AND METHODS

2.1. Using the framework

The visual scripting system is organized under the menus "Operations" and "Applications". Operations and applications are displayed in a tile representation. Applications have a second representation as a box, containing the list of operations. An example of a simple application is shown in Fig. 1. A new application is created with

Send correspondence to volker.baecker@mri.cnrs.fr

Figure 1. The tile representation of a simple application and its box representation. The applications is created by adding operations per drag and drop and by connecting them.

an empty list of operations. Operations are then added per drag and drop from the available operations. An operation can have one or more input slots and one or more output slots that are typed. To create a working application the input slots of all operations except for the first one have to be connected to output slots of foregoing operations with the same type. Furthermore operations can have options that can be changed in an options dialog. Each operation and each application has a help text that can be opened from the tile representation.

Applications saved in the default folder appear automatically in the applications menu. If the application is saved into another folder, the folder can be installed in the application menu afterwards.

2.2. Operation collections

Operations are ordered in collections. Users can create and save their own collections. Besides processing and analysis operations other kinds of operations are needed to create an application.
"Control operations" are operations that control the execution of the application. Two kinds of loops, "foreach image do" and "foreach object do are available. There is a limited support for branching in the form that a number of operations can be skipped when a condition is met. A third kind of control operation changes the program execution according to a user input.
"Input/output operations" show and hide or load and save images and analysis results.
"Reporting operations" write analysis results to spreadsheet files. Besides the generic "report measurements operation" reporting operations for specific applications exist.
The handling of selections (regions of interest) constitutes another class of operations. Operations in this class are for example: "inverse selection", "select all objects" or "objects to point selection".

2.3. Batch processing

Batch applications are created by using the "foreach image do" loop. When the "foreach image do" operation is executed, the user is asked for a list of image files. The application will iterate over all selected image files. If the user chooses a folder, the application will iterate over all images in the folder and in all of its subfolders.

The "open image" operation can be connected to the filename from the "foreach image do" operation. In the "open image" operations options a replace rule for the filename or parts of it can be defined. This can be used to load multiple dependent files in the same time, like for example multiple channels of an image.

If the loop contains a reporting operation, the filename for the result spreadsheet file will be asked only once before the loop is started.

The "save image" operation has the full path as a parameter that can be connected either to a "foreach image do" or to an "open image" operation. In the options of the "save image" operation, the output folder and additions to the output filename can be specified. The folder can be absolute or relative to the input folder and is created when it doesn't exist.

2.4. Interactive applications

Semi-automatic applications that require some user input can be created using the operations "accept or skip or exit" and "wait for user". The user interaction consists usually either in a decision to accept a result or in modifying a selection (region of interest) or an image, for example a mask, before the execution of the application continues.

If the execution reaches one of these operations a window with buttons is shown and the script execution is suspended until the user presses one of the buttons. If he is expected to modify an image or a selection he can use the standard ImageJ tools to do so. We created a toolbox that assembles the most important tools in one place.

"Cell Image Analyzer" adds some other interactive tools to ImageJ's palette, which are either used for experimentation, or in semi-automatic applications. The "slide show control" opens one image after the other in a folder. It can be configured to close the last opened image or all opened images before the next image is shown. The next image is shown in the position and with the zoom that the previous image had when it was closed. It can be configured to automatically apply brightness / contrast adjustment and a lookup table to the image. The "lookup table dialog" lists the available lookup tables, shows a preview and lets the user apply a lookup table to images. The "pixel spy" follows the mouse movement over the current image and shows a magnified view of the image region under the mouse cursor. The "channel mixer" lets the user adjust the minimum and maximum values of the RGB channels of an image.

2.5. Extending the framework

To add a new operation to the framework a new subclass of the class `Operation` has to be implemented. The methods that need to be implemented are: the `standard constructor`, `initialize` and `doIt`. In the `initialize` method the parameter and result classes and names of are declared. If the operation has options, the option names should be declared in `initialize` as well. For parameters and results other than `inputImage` and `result` getters and setters have to be defined in the class. The action of the operation is implemented in `doIt`. Normally an operation should not modify the input image, but work on a copy of it. If the operation works on a stack, slice by slice, the method `doItForSlice` should be implemented and the method `doIt` should call `processSlices`.

The method `setupOptions` must be implemented when non-numerical options are used. Classes for boolean, choice, list and matrix options with a predefined graphical user-interface exist. In the method `connectOptions`, options can be mapped to instance variables. If the operation has parameters other than the input image the method `cleanUpInput` should be implemented. The interpreter of the visual script will call this method at the moment the input of the operation is not needed anymore to free memory. To make the operation available to the user, it has to be declared in the method `setupOperationClasses` of the class `Operation`.

2.6. Accessing ImageJ methods

While an operation is running, ImageJ is switched to batch mode except for operations whose purpose is to show results. No image or measurements windows are opened in batch mode. An application, or an operation when not used within an application, is executed in its own thread.The standard ImageJ interface to run a command is `doCommand` in the class `IJ`. This cannot be used in an operation. The called command might open a dialog to ask for parameters and it would work on the current image managed by ImageJ. But the parameters and the image to work on should be set from within the operation.

The public interface of plugins consists of the two methods `setup(String arg, ImagePlus imp)` and `run(ImageProcessor ip)`. Most of the internal image processing code implements this interface as well. In the simplest case, when no parameters need to be set, ImageJ commands can be accessed by using this interface. In the case where protected accessors are available, the ImageJ code can be accessed by creating and using a subclass. However much of the ImageJ code has the default, package wide accessibility. To solve this problem a class `Proxy`[3] has been implemented that allows access to all methods and fields whatever their access level is. It does this by using java's reflection api.[4] It provides the method `execute` to access methods and the methods `setField` and `getField` to access instance or class variables that do not have accessors. So in general

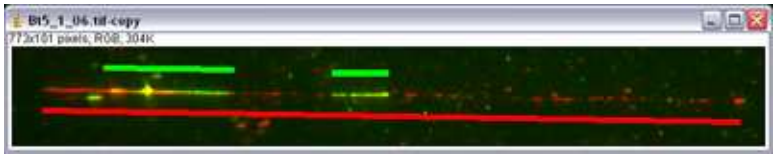

Figure 2. The DNA molecule is stained in red and sites at which replication takes place are stained in green. Above and below the molecule the results of the tracings are shown.

the following pattern is used, when access to restricted code is needed: A subclass of the class from ImageJ is created and used, giving access to all members that are at least protected. A proxy for the class to be accessed is build as a subclass of the class `Proxy`. The methods and getters and setters for fields that should be made available are implemented as calls to the superclass api. The subclass of the class to be accessed delegates calls to restricted code to the proxy.

2.7. The implementation

An application stores the list of operations and the mapping of parameters. The application can be written to a text file and read in again. The java reflection api is used to create objects from the operations' names when an application is read from a text file.

The visual script interpreter manages a program counter that indexes the current operation. It iterates over the list of operations. In each iteration the current operation is executed. With the help of the mapping of parameters, the following operations that use the results of the current operation are found. The parameter values are set to the result values using the java reflection api to get the getter and setter methods and to call them. In each iteration the result values that will not be needed any more are identified, set to null and a garbage collection is run to free memory. If an operation that signifies the end of a loop is executed, it sets the program counter to the index of the according start of loop operation if the loop hasn't reached the end yet and to the operation following the end of the loop, otherwise.

3. RESULTS

3.1. DNA Combing

The DNA combing application has been created together with Dr. Etienne Schwob's research group*. Images contain combed, i.e. stretched out DNA molecules that are stained with a fluorescent dye and appear red in the images. Sites within the molecule at which replication takes place are stained with another fluorescent dye and appear green in the images. The image analysis task is to measure the lengths of the DNA molecules and within each DNA molecule the lengths of the replication sites and the distances between the centers of the replication sites. See figure 2 for an example.

The task is made more difficult by the fact that the images contain a high level of biochemical noise. In addition the stretched out molecules are not perfect line segments, but contain small curvatures and large gaps can be present in a segment.

The application is implemented as a visual script. Two special operations to find the ends of the green and red line segments and one operation to filter out long objects have been added to the framework. Since the green channel contains less noise, the green segments are detected first. The green segments are traced with a method similar to the one used to trace neurites that is described in.[5] To find start points for the tracings a Hessian derivative operation that contains a smoothing filter is used on the green channel. An auto-threshold is applied to the result. Objects are found using an operation that wraps the `Particle Analyzer` from ImageJ. The width and height and the circularity measurements are used to remove objects that clearly do not represent line segments. Shortest paths in the neighborhood of the remaining object centers are computed in the Hessian

*Institut de Gntique Molculaire de Montpellier, UMR5535, CNRS

derivative as described in.[5] The tracing of the green segments in the shortest paths image allows for gaps of a configurable size.

Since the quality of the red channel images is much lower, a different approach is used to trace the red segments. The centers of the green segments are used as starting points to trace the red segments. A small line segment is placed into the current point. The following point of the tracing is computed by turning it around the current point and measuring the average intensity under the segment. The tracing is continued in the direction that yielded the highest value. The tracing stops when the best measured value is not higher as the intensity measured in the perpendicular direction.

To get data to evaluate the automatic tracing, an application for manual tracing has been implemented. The red and green segments are selected with the polygon selection tool. A special DNA combing tool is used to manage the tracings. Tracings can be added and removed. When a tracing is selected, it is shown in the image. The sets of tracings can be saved to a file. The tracings can be measured and the results are written to a spreadsheet file in the same way as in the automatic application.

3.2. Measuring intensity ratios between nuclei and cytoplasm

This application has been developed together with Dr. Olivier Coux[†]. A protein has been stained with a fluorescent dye and images of cells containing the protein have been taken. The assumption is that the proportion of the protein present in the cytoplasm and in the nuclei is changed by the experiment. To identify the nuclei epifluorecent DAPI images have been taken. The image analysis task is to measure the proportion of intensity in the nuclei and in the cytoplasm for each image.

An operation to find and subtract the background intensity has been implemented for this application. It searches the maximum intensity value in the neighborhood of intensity minima. This operation is applied to the input image. A threshold is applied to the DAPI image and the particle analyzer is used to identify the nuclei. A selection is created from the resulting mask and transferred to the input image. The integrated intensity within the selection is measured. The selection is inverted and the integrated intensity is measured again. The proportion is computed and the result is written to a spreadsheet file. To have a control of the results, an image showing the identified background and the identified nuclei is saved for each input image.

3.3. Measuring fluorescent stained proteins

The application has been created together with Dr. Philippe Fort's research group[‡]. The image analysis task is to measure number and size of fluorescent stained proteins that appear as small plaques in the cell images. The cell cytoplasm contains some intensity as well that sometimes lies within the same greyscale range as the intensity in the plaques. The application applies a Gaussian filter with a large radius to the input image and subtracts the result from it. In this way most of the cytoplasm intensity is removed while the plaques that have sharp borders are kept. The plaques are then measured and counted in the standard way, using a threshold and the `Particle Analyzer`. A semi-automatic version has been added. It allows drawing a selection around cells that shall be taken into account, before the application continues. An overlay of the input image and the outlines of the found objects is saved for each image.

3.4. Counting nuclei

Multiple applications to count nuclei or cells have been implemented together with several research groups: Pr Jean-Jacques Guilhou[§], Philippe Pasero[¶] and Marc Piechaczyk[‖]. In the general case the nuclei might be stained with multiple fluophores revealing the presence of different proteins in the nuclei. The image analysis task is to count for each image the number of nuclei stained with each fluophore. The main problem is the separation of nuclei touching each other in the image. Watershed[6] approaches often suffer from over-segmentation. An

[†]Centre de Recherches en Biochimie Macromolculaire, FRE 2593, CNRS
[‡]Centre de Recherches en Biochimie Macromolculaire, FRE 2593, CNRS
[§]JE 1950, Institut Universitaire de Recherche Clinique
[¶]Institut de Gntique Humaine, CNRS, UPR 1142
[‖]Institut de Gntique Molculaire de Montpellier, UMR5535, CNRS

approach based on image derivates has been used. A full automatic and an interactive version are implemented. In the interactive version the images are opened and the found nuclei are marked by point selections. The user can correct the selections before the results are exported.

3.5. Automation of simple image proccesing tasks

Another field of application, besides image analysis, is the automation of simple image processing tasks. Example applications are batch image type conversion, the creation of compressed movies from very large data sets and the creation of overlays of fluorescence images and phase contrast images for large time series. To create compressed movies from very large data sets the virtual stack opener plugin is used in combination with the QuickTime Stack Writer plugin.

4. CONCLUSIONS

ImageJ with the Cell Image Analyzer addition can be used to rapidly create custom image analysis applications and to automate image analysis tasks. The visual scripting interface can be helpful in the collaboration of application developers and scientists. The framework supports experimentation to find solutions and allows turning a solution into a batch application quickly. Semi-automatic applications in which the user takes decisions or corrects preliminary results can be created. On the one hand they can be used to let the user work more efficiently in cases where full automatic procedures are not reliable enough yet, on the other hand they can be used to acquire the data necessary to evaluate an automatic solution. The implementation of new operations profits from the infrastructure provided by the framework. The framework contains a mechanism to use ImageJ code with restricted access.

At the facility Montpellier RIO Imaging this framework has been successfully used to create image analysis applications on demand in close collaboration with the biologists. It was thus possible to provide solutions that could not be obtained with the available standard image analysis packages.

In the current state the visual scripting interface is useful to create applications up to a certain size and complexity. Although the complexity can be moved from the scripting level into the operations on the programming level, this would hurt modularization. This restriction could be overcome in the future by allowing hierarchical applications, i.e. by treating applications as operations that can be used in other applications.

ACKNOWLEDGMENTS

I want to thank all scientists who participated in developing image analysis applications.

REFERENCES

1. S. J. R. M. D. Abramoff, P. J. Magelhaes, "Image processing with imagej," *Biophotonics International* **11**, pp. 36–42, 2004.
2. M. D. M. Boshernitsan, *Visual Programming Languages: A Survey. Technical Report CSD-04-1368*, University of California, Berkeley, Berkeley, 2004.
3. R. J. E. Gamma, R. Helm and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
4. I. Forman and N. Forman, *Java Reflection in Action*, Manning Publications, 2004.
5. J.-C. F. S. P. S. H. H. E. Meijering, M. Jacob and M. Unser, "Design and validation of a tool for neurite tracing and analysis in fluorescence microscopy images," *Cytometry* **58A**, pp. 167–176, 2004.
6. S. Beucher, "The watershed transformation applied to image segmentation," *Scanning Microscopy International* **6**, pp. 299–314, 1992.