JavaVis: open source code for computer vision subjects *

J.M. Perez, B. Bonev, P. Suau, D. Viejo, and M. Cazorla

Dept. de Ciencias de la Computación e Inteligencia Artificial
University of Alicante, P.O. Box 99, E-03080 Alicante
{jmperez,boyan,pablo,dviejo,miguel}@dccia.ua.es

Abstract. In this paper we present the open source project JavaVis, oriented to Computer Vision teaching. It consists of a framework with several features that make it useful for that purpose. Some of them are: a) image format, supporting frames and bands for sequence processing, b) code with many algorithms available, it contains implementations of some of the most widely used algorithms in computer vision, c) 3D data support, d) a visual desktop for visualizing partial results. It was designed to be easy to use: the user does not have to deal with internal structures, and adding a new algorithm is a simple task. J
We have developed three different modules, based on three different needs we have noticed in our subjects. The first one is a basic library for image processing. Besides the previously commented features, it supports geometrical data (edges, segments, points, etc.). The second module is based on the same working schema as the first one, but applied to 3D data. These two modules are enough for testing many well-known algorithms. They also suit the programming needs of students and teachers, as they can easily develop their own algorithms for the JavaVis framework. All JavaVis functions can be launched both from command line, as well as with the JavaVis Graphical User Interface. Finally we have extended JavaVis with a third module consisting of a visual desktop where different Computer Vision functions can be easily placed and connected. Its purpose is to visualize intermediate results in processes involving several functions, helping their better understanding.
JavaVis, as its name suggests, is implemented in Java.

1 Introduction

Computer vision is an important subject in computer science degrees. For laboratory lectures, we need a tool that is complete and easy to use. In this work we present a Java library which is oriented to teaching. This means that we have designed and built the library thinking in readability and understanding instead of efficiency.

We have developed three different modules, based on three different needs we have discovered in our vision related subjects. The first one is a basic library to

* This work has been partially supported by project GV06/134 from Generalitat Valenciana (Spain).

process images. It is the oldest one and it follows a special file and image format enabling easy sequence processing; it incorporates geometrical data (edges, segments, points, etc.) and it has implemented several well-known computer vision algorithms. The students can, easily, develop their own algorithms. The second module is based on the same working schema than the first one, but applied to 3D data. Finally, we have developed a visual desktop in order to visualize how the different algorithms work and the results of combining them.

Several libraries and systems have been developed for computer vision research and education. First of all, we will describe libraries written in any programming language, and we will conclude with Java libraries. Khoros library [7] is available for Unix platforms, Windows and MacOs. It incorporates a visual programming environment where programs are created by placing toolboxes: rectangular icons that represent operators, which are simply stand-alone programs written in C, C++, Java or a script language. Each operator performs on an input image or dataset, producing an output image. Connections that represent data flow between the toolboxes are created by clicking the mouse. To complete the visual programming capabilities, there are advanced programming language constructs such as loops, procedures and control structures. This library is currently not freeware nor open source. Open Computer Vision Library (OpenCV) [8] is available for Windows and Linux. It is distributed under Intel's licence for both commercial and non-commercial (research and teaching) purposes. The library includes over 300 image analysis and processing methods from morphology, geometry, image treatment, etc., up to the recently added methods for computing stereoscopic correspondence, face recognition or 3D tracking. Although OpenCV is one of the most complete and efficient computer vision libraries, it is not portable enough for other operating systems. Also, OpenCV does not incorporate a Graphic User Interface (GUI) to visualize and evaluate results. VIGRA [9] is a novel computer vision library which focuses on customizable algorithms and data structures. The library was built using generic programming as indicated by the Standard Template Library (STL). By writing a few adapters (image iterators) it is possible to use VIGRA's algorithms in computer vision applications. Nevertheless, the library does not have enough implemented algorithms, nor does it have a GUI.

On the other hand, there are a few libraries written in Java. Java Imaging and Graphics Library (JIGL) [4] was developed at Brigham Young University to make programming both course-level and research-level image-handling algorithms as easy as possible. JIGL extends standard Java image-handling capabilities. It is based on the Java Advanced Imaging (JAI) [3] development by Sun Microsystems. JAI is a base library which focuses primarily on web-based applications. Nevertheless, JAI is not useful for computer vision applications. This is why JIGL was developed. It is built around a set of image classes that support individual pixel access, image-wide operations and image-image operations. Nevertheless, the number of included computer vision algorithms is low. Image Processing in Java (IPJ) [10] is another JAI-based library. The IPJ goal is to expand JAI's functionalities with computer vision algorithms. The methods

included in the library are spatial filters, convolutions, compression, morphological filtering, boundary processing and chromatic light. Nevertheless, some of the more elaborated methods begin in lack as, for example, sequence processing, object tracking, 3D stereo vision, etc. Furthermore, IPJ does not have templates for adding new algorithms to the library, nor does it have a GUI. Java Vision Toolkit (JVT) [5] is based on the JAI library, adding computer vision algorithms for 2D and 3D images. JVT provides implementations for image-handling filtering, edge detection, segmentation, Hough transform, morphology and colour analysis. It also includes a GUI application, which makes JVT easy to use for end-users and developers alike. JVT is designed for students using the image-computation template provided to implement new algorithms. The main problem with JVT is the lack of sequential image handlers. ImageJ library [2] is based on three fundamental features. First, the use of macros, which allow users to automate tasks and create custom tools; second, it is possible to extend the capabilities of ImageJ by developing plug-ins; and finally, we can process an entire stack of related images using a single command. The library core includes several image handlers. All the related computer vision functions are incorporated as plug-ins. Also, ImageJ contains a GUI which allows end-users to launch the library algorithms and evaluate the results. This library is widely accepted in scientific areas, although the lack of templates for the development of new algorithms makes it less appropriate for educational purposes.

The rest of the paper is organized as follows: In Section 2, we describe the main features of JavaVis. Section 3 explains how the 3D GUI works and Section 4 explains the JavaVisDesktop. Finally, some conclusions are drawn in Section 5.

2 JavaVis

JavaVis was designed as an open source tool for computer vision teaching. This section will introduce its main features. We began to develop JavaVis following the same philosophy of work than Vista [1]. Vista was a computer vision library written in standard C which tried to simulate the object-oriented programming. The features of Vista were very attractive for developing algorithms in vision and, in fact, many researchers used it, but its creator left the project. Our objective was to create a similar library with a totally OO language. Most of the features of JavaVis are based on the operation of Vista.

In JavaVis we have defined both a file and an image format that allow to work with image sequences. Some computer vision algorithms have to manage image sequences. For example, optical flow algorithms use several images to estimate movement and stereo computation needs two or three images to work. Furthermore, images are usually composed by different bands, for example, a RGB color image can be stored in three different bands. In this way, JavaVis handles its own image format that enables it to manage image sequences and bands.

Each image in JavaVis is composed of one or several frames. A frame represents an image that can be of two types: bitmap or geometric. A bitmap frame

is an image represented as a matrix in which each element is a pixel (picture cell). JavaVis has five types of bitmap frames: BIT, BYTE, WORD, REAL and COLOR. A brief explanation of frame types is showed in Table 1. In addition, each bitmap frame can be formed by one or several bands.

Table 1. Bitmap frame types in JavaVis

Type	Description
BIT	Defines a binary image where a pixel value may be 0 or 1
BYTE	It represents the classic gray scale image, with pixel values in the range [0, 255]
WORD	Uses a range of [0, 65525]
REAL	It is the only format which allows negative values, useful in convolution and gradient computations. It is represented using the float type of Java
COLOR	Color image (R, G, B) formed by a three bands image, each band being of *BYTE* type

On the other hand, a geometric type frame manages information in a different way. For example, the *SEGMENT* frame type just stores the coordinates (x and y) of the initial and end defining points of a segment. A *SEGMENT* type frame may have several segment objects stored not into a pixel matrix but into a segment object list. So the representation of geometric frames is more compact and computations are faster. The available geometric types are showed in Table 2. This kind of frames are useful in several computer vision algorithms.

Table 2. Geometric frame types in JavaVis

Type	Description
POINT	The simplest type, a 2 dimensional point.
SEGMENT	It represents segments. A segment consists of an initial and an end point.
EDGES	It represents a set of points forming an edge. They keep an adjacent relation, i.e. the first point is adjacent to the second, the second to the third an so on. Adjacency between the last point and the first one is not necessary.
POLY	It contains a set of points forming a polygon. The points form a circular sequence.

A sequence can be o rganized in two ways: several frames in a sequence, or several bands per frame, which can also form a part of a sequence. However, in

order to insert several bands in the same frame, every band must have the same size and be of the same type whereas different frames can have different size, type and number of bands. In Fig. 1 we show an example of an image in JavaVis. Finally, we have developed a special file format to store all information from our image format. It's called JIP format or JIPZ for compressed files in order to save disk space. JavaVis can read both JIP and other file formats such as JPEG or GIF, converting them into its own image format to work with them. For storing source images and computer vision algorithm results the JIP or JIPZ formats can be used, an exporting to JPEG is also allowed.

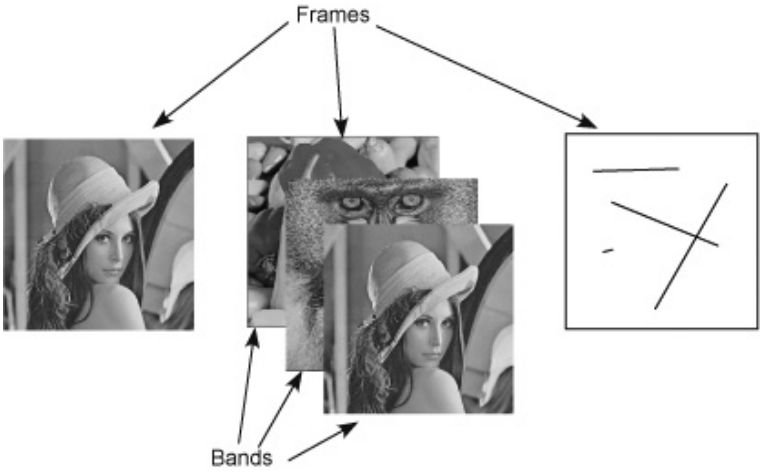

Fig. 1. A sequence in JavaVis having three frames. The second one is formed by several bands and the third one is geometric.

An important point in JavaVis is the organization of its functions (implementation of algorithms), as it is important for other users to be able to easily implement their algorithms in a standarized way. A function in JavaVis is a Java class which implements an algorithm or method. A function inherits from an abstract class JIPFunction. In order to implement an algorithm, only the function code must be developed and input and output parameters specified. When a function is executed, neither the user nor the programmer has to check the parameters; JavaVis does it automatically.

There are three ways to execute a function. The first one is from command line: the Launch class can execute a fucntion of the library, after the function name and its parameters are specified. This class checks the values entered (name, type and range of the parameters) and returns an error if something is wrong. Otherwise, the function is executed. The execution takes an input file and generates an output file. The output file has the same sequence as the input file, where every frame has been processed with the function. The second way to execute a function is from the graphical user interface (see Fig. 2). A func-

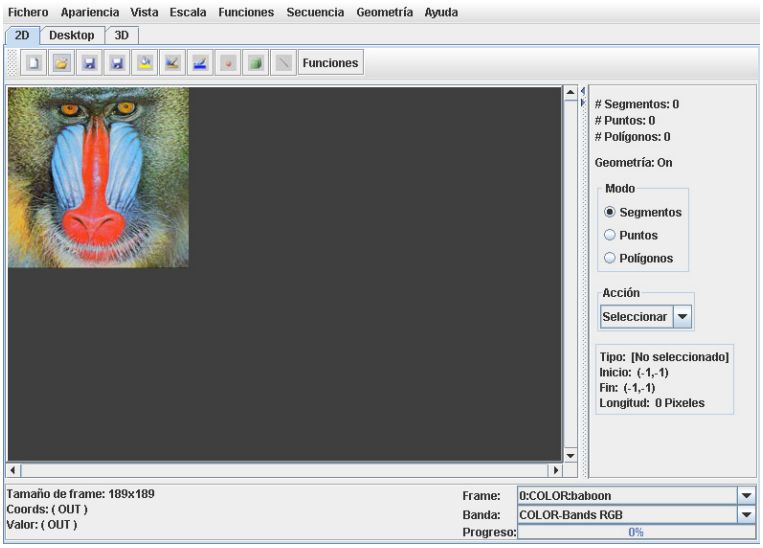

Fig. 2. Graphic user interface used to visualize images and to apply functions.

tion can be executed by introducing the input parameters in a window and the parameters are checked too. Furthermore, the function can be applied to all the sequence or just to the current frame. The final way to use a function is from another function. Each function can use any existing function defined in the library. Note that despite the possibility of executing a function in three different ways, it is only defined once, and there is no need of additional configuration of the JavaVis environment but just adding the class for the function in a directory.

Another important issue regarding functions is the definition of parameters. A standard is defined for both input and output parameters for JIPFunctions. This is done by means of a class JIPParameter. Thus, the GUI can set and get parameters for any implemented JIPFunction.

Finally, regarding function implementation, it must be noted that Matlab linking is possible for intermediate results, as functions for transforming an image to a Matlab matrix are provided. JavaVis is mainly used in academic and research areas, so linking with other mathematical and Computer Vision related programs is a very usual situation.

Table 3 lists some of the functions already included in the distribution of JavaVis. Many of them are implemented by undergraduate students who used JavaVis for their projects and needed to implement new functions. The maintainers of JavaVis have revised their work before including it into the JavaVis distribution.

Table 3. JavaVis JIPFunctions

Group	Functions
Transform	They allow image format conversion: FColorToGray , FGrayToGray , FGrayToColor , FRGBToHSB , FRGBToHSI , FRGBToYCbCr , FBinarize .
Adjustments	Image adjustment: FBrightness , FContrast , FGamma , FEqualize , FSharpen , FSharpenMore .
Smoothness	Image smoothness, reducing noise: FSmoothAverage , FSmoothMedian , FSmoothGaussian .
Convolution	Implement several ways to convolution: FUser3x3 , FUser5x5 , FConvolveImage , FConvolveAscii , FGabor .
Manipulation	Manipulate the image, rotating it, scaling it, and so on: FOpenWindow , FSkew , FFlip , FFuzzyKmeans , FKmeans , FMirror , FScale , FRotate , FCrop , FWaveHoriz , FWaveVert , FNegate , FNoise , FMaximum , FMinimum , FPixelate .
Geometry	Apply functions to the geometric data: FAddPoint , FAddSegment , FRandomPoint , FRandomSegment , FInterSegment , FGeoToGray , FHoughCirc , FHoughLine .
Edges	Calculate and operate with edges: FCanny , FLink , FSegEdges , FNitzberg , FSusan , FGrad , FMag , FPhase .
Math Morph	Implement morphological operators: FErode , FDilate , FClousure , FOpening .
Applications	Functions which use other functions and implement complex application: FCountCoins .
ImageBD	Makes image Data Base searching: FCalcHistoColor , FCalcHistoBD , FSearchImage , FSOM .
Ring Projection	Manages omnidirectional images: FRectifyOmnidir , FCleanOmnidir , FRingCreateMask
Other	Functions non included in previous groups: FHistogram , FSkeleton , FOp , FSegmentHSB , FBlobs , FManhattan , FCapture , FAnnealing, FPca , FPcaRecon, FFlow .

3 JavaVis3D

3D data is a special data type formed by a set of points where each point is described by 3 coordinates (usually X, Y and Z) and, sometimes, a gray or color level associated to this point. These data come from a stereo camera or a 3D laser. We have realized that others computer vision libraries do not manage 3D data and we think it could be useful to incorporate a tool for managing them.

Java provides a 3D API: Java3D. We have used this API in order to incorporate a GUI for showing 3D data (see Fig. 3). The GUI is built on a typical

3D scene in Java3D. Data is shown as points and they can be rotated, zoomed in and out, and translated. We can also change several properties like aliasing, color and point size.

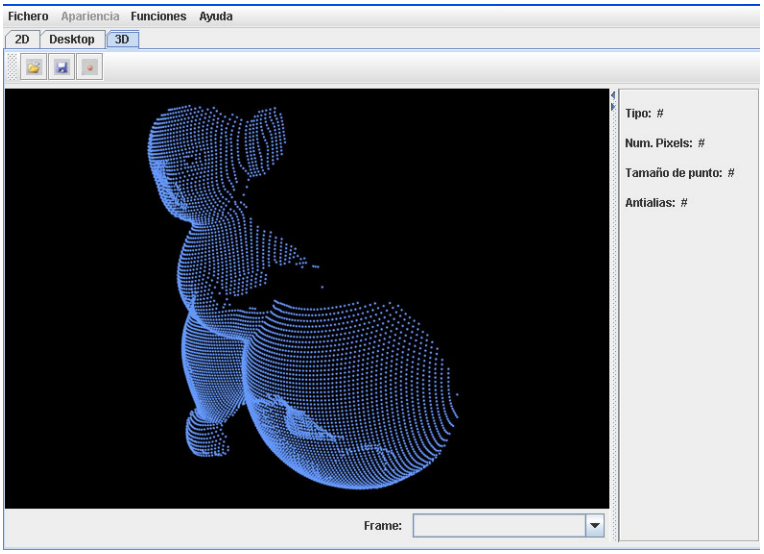

Fig. 3. JavaVis 3D Gui.

File format is different from images. 3D data are represented by a list of coordinates and, if available, color information. File has a header with metadata information and then the list of points, one for line. It supports several frames (a sequence) like images, but not bands. Functions follow the same schema than previously: they receive and return an 3D data object. Several additional classes and methods have been implemented for data manipulation.

4 JavaVisDesktop

Another new feature we have recently incorporated to JavaVis is JavaVis Desktop (see Fig. 4). The goal was to build a tool to allow a better understanding of partial results when processing an image. Different algorithms may be used in order to build more complex routines. Fig. 5 shows an example of an algorithm that counts the number of coins in an image. The algorithm is built using several already implemented algorithms, like Hough transform [11] and Canny algorithm [12]. The Desktop utility allows us to check partial results, showing the images obtained applying an algorithm.

This tool allows to join a sequence of functions, like an automata. Each node in the sequence is a function (algorithm) from JavaVis. Each node shows the

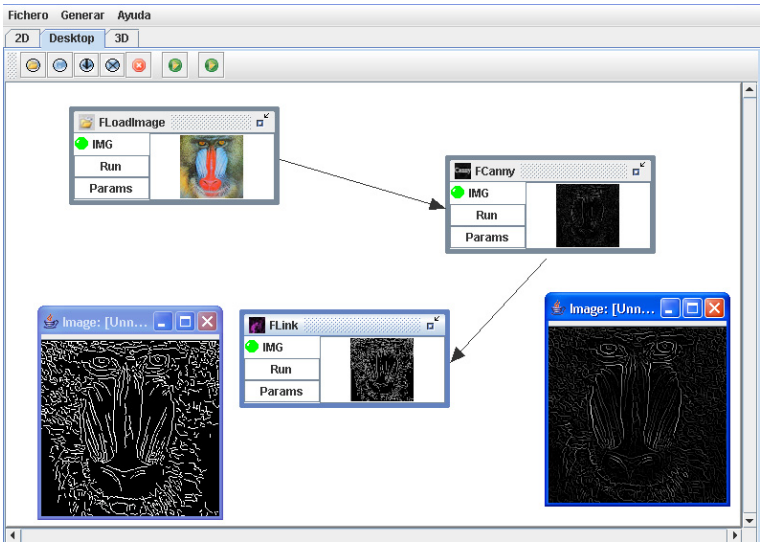

Fig. 4. JavaVis Desktop.

result of applying its algorithm as a thumbnail image. The full-size result of each step can be seen by clicking on the thumbnail. The result is only available if the function has been successfully executed. Each node includes a button to execute its function, as well as a button for setting its specific parameters.

This new routine created as a sequence of other algorithms, can be easily added to JavaVis as a new function, with a button on the Desktop interface. Thus, the user can run this new function from the JavaVis interface, from command line, or to use it from a java program. Also this is a way to facilitate the sharing of new Computer Vision (CV) algorithms. The aim is to impulse the community to incrementally construct the CV library. Many CV algorithms are easy to implement if some basic functions are already developed, as seen in the previous coin counting example. Also, the sequence of function, together with the parameters of the functions is saved in XML format, for further editing or running it in the JavaVis environment.

5 Conclusions

In this paper we have introduced our open source framework for teaching computer vision written in Java: JavaVis. Its features include a new image format capable of handle images and geometrical data, with multiple bands and frames for sequence processing, several launching methods (including graphical interface) and a large quantity of implemented computer vision algorithms. We have described the recently new modules added to JavaVis: the desktop, useful to understand how final result is obtained from partial results, and Javavis3D to support 3D data. This tool have been used during past years in computer vision

Fig. 5. Two examples of counting coins. The white circumferences are the coins detected. Note that the CD is not detected as it has a different color, and the inner circle of several coins is not detected too.

subjects in University of Alicante. Open source makes easier some tasks needed during our teaching experience: students can examine the code and see how algorithms are implemented, and as a consequence, how they work; and also, new functions can be easily added.

Our framework is under continuous development and improvement. New vision algorithms implemented by students and revised by teachers will be included. Currently we are developing a new version of JavaVis. It will be available by march 2007. Connectivity with Matlab could also be improved.

References

1. 2006. The Vista website. http://www.cs.ubc.ca/nest/lci/vista/vista.html.
2. 2006. The imagej website. http://rsb.info.nih.gov/ij.
3. 2006. The java advanded imaging website. http://java.sun.com/products/java-media/jai.
4. 2006. Java imaging and graphics library. http://rivit.cs.byu.edu/jigl.
5. 2006. The java vision toolkit website. http://marathon.csee.usf.edu/~mpowell/jvt.
6. 2006. Javavis web site. http://javavis.sourceforge.net.
7. 2006. Khoral. http://www.khoral.com.
8. 2006. Open source computer vision library. http://www.intel.com/research/mrl/research/opencv
9. 2006. The vigra library website. http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra.
10. Lyon, D. 1999. Image Processing in Java. Prentice Hall.
11. Duda, R. and Hart, P. "Use of the Hough transformation to detect lines and curves in pictures" (1972) Communications of the ACM, 15 (1), pp. 11-15
12. Canny, J. "A computational approach to edge detection". (1986) IEEE Trans. on Pattern Analysis and Machine Intelligence, 6 (8)