DEVELOPMENT OF IMAGE SEGMENTATION METHOD THROUGH CONTOUR ALGORITHMS

Henrique de Padua Valle, Teruo Matos Maruyama, Márcio Hosoya Name, Rosane Falate

Universidade Estadual de Ponta Grossa (UEPG) – Ponta Grossa – PR – Brasil

henriquepvalle@gmail.com, shinigam8@gmail.com, name@ufpr.br, rfalate@yahoo.com

Abstract. *The digital image processing is, among others, a technique that extracts data from an image in an attempt to automatically achieve specific information from an image. To help a computational system to have a better productivity and performance it is interesting to make the image background irrelevant, then the computational math are done just where it is important in the image. The goal of this work is to produce a system with this capability of separate the object of interest from the image background. For that, it was used the programming language Java and the open source library for computer vision, OpenCV, with the plugin JavaCV, which allows the use of the library in the chosen language. As result, it was observed that the program met the expectations, reducing the noises, taking off the undesired parts of images, and making the image parts of interest detached in the new image.*

Keywords. *Image Processing; JavaCV; OpenCV; Segmentation; Interest Object.*

1. Introduction

The digital image processing (DIP) is very important in many areas (medicine, remote detection, security, material engineering), with several purposes through software [6], [7]. In resume, such software is either to improve the quality of images or to attempt to extract a more useful knowledge, as specific information to be used in the application area.

The DIP, usually, involves the build of algorithms [6]. For this reason, with exception of acquisition and display stages, most of image processing functions can be implemented through software. Nevertheless, the numerical sequence or computational procedures are dependent of the knowledge area where the image processing is being applied, information which is desired, and, especially, the image itself [5]. Because of it, it is difficult to develop a generic solution that can be applied in more than one subject or on images that are totally different. However, almost every system can use of the image segmentation, which is the extraction and the separation of the existing and relevant objects of the images from the background, according to the final goal of the system [5]. It should be emphasized, however, that the success of the segmentation process depends on the prior definition of what is wanted in the final system (which parts and characteristics of image is desired), and just after it, one can start the development of the segmentation algorithm.

This study aims to develop a tool that allows the image segmentation by choosing the color boundaries from the RGB (red/green/blue) color space. We applied the developed software to detach single and multiple objects of different images.

This document is divided into five sections, besides this introduction: fundamentation theorical, methodology, results, concluding remarks and references. The section 2 is dedicated to the

explanation of the fundamentals related with the both, image segmentation and digital image processing in general. On sections 3 and 4 are presented the descriptions of the methodology and the results obained. Finally, the concluding remarks are reported in section 5.

2. Theoretical Background

This section presents the approaches and the theoretical background about the segmentation method, which served as the basis for production of the software.

2.1 Digital Image Processing

The digital image processing stands in several studying areas and has been of great importance in existing systems. Basically it consists of: (i) search of objects and data extraction, which can give information to be applied in a specific area; (ii) or quality improvement of image to help human viewpoint, making easy the analysis of the image.

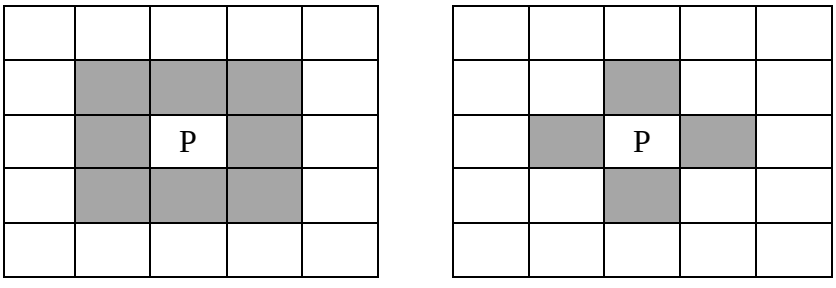
One of the substeps of DIP is the segmentation which is responsible for divide and detach the interesting objects from image. This task is considered, by researchers, the most complex to implement [5]. According to [1], the reason of this difficult is the relation, not exclusive of this step, but of the whole system with the area where it apply. On this way, an object of interest desired to extract from an image can drastically change from an area to other, becoming hard the production of a generic algorithm or a method.

2.2 Image and Pixel

Images are numerical matrix where each discrete element is denominated pixel. The most common form of a pixel is the rectangular or square. In addition to have finite size, each pixel has a set of values, according to the color space, which results in hue point of image [1]. In this way, image is built with the correctly ordered set of pixels and their respective hue or tone.

A monochromatic image can be defined as a discrete function of intensity light f(x,y) where each value corresponds to the brightness (or gray level) at the point (x,y). In the case of color image, hues are from intensity values of the pixel, which come from the channels of the color space. Therefore, color image has more than one function, one for each channel. Image processing uses of these pixel values to handle images by doing operations like comparisons, mean value etc. [5], [6].

An important concept in digital image processing is pixel neighborhood. A pixel P, in the middle of an image, has 2 horizontal, 2 vertical and 4 diagonal pixels. When all pixels around the pixel P are considered, it has the 8-connected neighborhood, figure 1(a). When the last 4 diagonal pixels are not considered, it is called 4-connected neighborhood, figure 1(b) [5], [6].



(a) (b)

Figure 1 – Pixel connectivity: (a) 8-connected neighborhood, and (b) 4-connected neighborhood.

[PECCINI and D'ORNELLAS, 2004].

The concept of neighborhood is used in some processing calculations applied on images.

2.3 Segmentation

The image segmentation consists of separate regions or objects of interest from the background in a way processing system only works with important parts of the image. Same example of segmentation methods are thresholding and edge detection [6].

2.3.1 Thresholding

Thresholding is a process where, from an input image with N color levels, an output image is generated with a smaller number of levels. When the output has only two color levels this process is denominated binarization [8]. In the case of binarization, the algorithm scans the image pixel to pixel and, according to a pre-defined threshold, each pixel of the image output receives a binary value. In the image analysis, pixel values that are bigger than threshold are considered as part or region of interest, and values on contrary are considered as background [4].

2.3.2 Edge Detection

Edge is the frontier between two regions where the intensity levels of their boundaries are considerably different [4]. The edge detection algorithm has been the most used segmentation method. It searches for discontinuities on the image, through the intensity analyses of pixels, and then marks when this event occurs [9].

2.3.3 Code of Freeman

The Code of Freeman is used to edge detection, and its goal is to obtain and to storage the contour of the interest object, from the contrast between regions [3].

From a binary image (black and white), the algorithm scans the image, pixel to pixel, from the left to the right and from top to bottom; until it finds the first white pixel. Then, it searches, in the neighborhood of this pixel, the first white pixel, which has a neighborhood black and is located in the direction of smallest value, repeating the process with the following white pixel. In the end, in the list created by the algorithm there are: the coordinates of the first found white pixel, and a list of integers values corresponding to directions that one must follow to contour the whole object. The directions can be either four (north, south, east and west) or eight, case where the diagonal pixels is considered, according to the neighborhood type used by the system.

After obtained all contour of an object, the algorithm continues scanning the image searching for others objects which can be on the image. At the end, the code returns a list with the contours of the objects found.

2.4 Mask

Mask constitutes on a matrix with values or coefficients variable according to what is wanted to manipulate or obtain in the image. The proper algorithm slides the chosen matrix through the image doing calculus with the pixel values and the ones of the mask, and stores the results, the desired change or image information, in a new image.

Calculations using masks are widely used on image processing. The use of appropriate coefficients becomes possible a great number of useful tasks like noise reduction, and sharpening, which can be useful on the segmentation process [5].

3. Metodology

The chosen development tool was OpenCV (Open Source Computer Vision), version 2.4.0, which is an open source library for computational vision, whose objective is provide a simply infrastructure of computational vision to help on the rapid development of different applications [3]. Behind this library, it was also selected the API (Application Programming Interface) javacv 0.1, to be able the use the OpenCV with Java language. The Java language was chosen because, at the end, it is possible to have multiplatform software, open source and with chance of web use, attributes provided by this language.

The software of segmentation was developed in two computers, one of them with operational system Windows 7 Ultimate 32 bits, 3GB of memory RAM, processor Intel Pentium Dual-Core 1.86GHz, and the other with operational system Windows 7 Home Premium 64 bits, 4GB of memory RAM, processor Intel Core i3 2.27GHz. Each computer has the IDE (Integrated Development Environment) Netbeans 6.9.1, to be possible the development of the Java code, using javacv. For software tests, we used, in the most part, images from the data bank available at http://staff.science.uva.nl/~aloi/, with dimensions (768 x 576), in PNG format. Hereafter, we described, in detail, the tools and methods (OpenCV, JavaCV, ALOI), in relation to evolution and technique.

3.1.1 OpenCV

OpenCV is a library with functions of computational vision programming. It was initially developed by Intel with languages C and C++, and now it is maintained by Willow Garage and Itssez, being compatible with Windows, Linux, Android and Mac OS [11]. OpenCV was designed in order to enjoy the better of hardware and to have a strong focus on real-time applications. Moreover, this tool enables better utilization of multi-core processors, in other words, it can be used with parallel architecture [3].

3.1.2 JavaCV

The API JavaCV is from Okutomi & Tanaka Lab, at Tokio Technological Institute, by Samuel Audet, during his PhD research in 2009. The JavaCV provides an interface for the use of programming libraries used by researchers in the field of computer vision as: OpenCV, FFmpeg, libdc1394, PGR, FlyCapture, OpenKinect, videoInput and ARTollKitPlus. The classes are separated according to the utility, making it easier to use at platforms Java and Android [2].

3.1.3 ALOI

ALOI (Amsterdam Library of Object Images) is a color image data base, with thousands of small objects, stored for scientific purposes. To have changes in image attributes, image of objects are systematically recorded after variation of: view angle, light angle and light color. In this way, it is stored over than a hundred pictures of each object, producing a total of 110,250 images per collection [10].

3.2 Developed of the Segmentation System

3.2.1 Development Sequence

Each image used on the application is stored in a data structure similar to a matrix, where relevant calculations are applied; this structure is known as objects of type IplImage. This object has various methods and information that are used in functions that the program uses, like number of channels, image size and pixel size [3]. During the application execution, the image

that is wanted to segment is always kept in cache memory. This makes the image processing faster than in the case where images are stored on hard disk.

The developed segmentation system has some distinct steps: load image, thresholding, obtaining contours, filtering, creating the mask, and segmentation of objects of interest in independent files. To facilitate the implementation of test routines and to verify the results, a graphical interface was built using the available APIs that are allowable with the Java language.

For load the image in the graphic interface, it is used the component browser, that allows one to search and choose files and folders present on the computer. Once the image is chosen, this component gets the directory path of the image and then creates an object of type File with it that, after, creates an IplImage structure with this file. JavaCv allows the use of the cvLoadImage() function with the image directory as parameter. However, it was used the process described above because of the incompatibility between the cvLoadImage() function and cases where special characters occurs on the name of the directory address, like accents and cedilla.

All of the images used on the development and tests were colored and became black-white images (or binary), with the process known as thresholding. This process requires, beyond the image to be thresholded, an output image, which has just one color channel and where it is saved the result of this process; a value for the lower threshold and a value for the higher threshold. It must note that is needed threshold values for all color channels. In other words, when the system uses the channels RGB, it is required three pairs (a lower and a higher one) of threshold values.

With the chosen threshold values, it is compared the channel value of each image pixel and the thresholds of the corresponding channel. So, it is assigned a value of 255 (white) for those pixels which are between the thresholds (object of interest), and 0 (black) to the remaining pixels (background). For the higher threshold it was adopted a constant value of 255 at each channel, meanwhile the lower threshold values were defined manually for the user. Graphic interface allows continuous change of lower threshold values, through the *JSlider* components; and real time view of the thresholded image, which makes with ease the choice of the values where the object of interest were detached from the input image.

Due to the channel manipulation be done separated, both original and thresholded image are of the same type, so the last one has the same size and channel numbers of the original image [3]. The cvInRangeS() function was responsible to realize thresholding in the three channels. With the following parameters at cvInRangeS() function: input image; two objects of type CV_RGB, one with lower threshold values and other with the 255 higher threshold values; and output image, the result of thresholding was generated and shown immediately to the user.

To obtain the contours in the image that was thresholded, it is applied the cvFindContours() JavaCV function, which implements the Code of Freeman to get the contours of the interest objects. This function needs some parameters: thresholded image, list to store the contours of the found objects, region to memory allocation that will store the list, and the type of algorithm to be used (Code of Freeman).

With the obtained contour points of each object, the object position on the image is determined and the object size, through the cvContourArea(), which needs as input only the object contour, and returns the object area, in pixels. To eliminate small artifacts or noise, we determined a minimum value of area for an object is considered of interest. The chosen standard value was 50; however, the developed system allows changes in this value according to the will of the user.

To apply a filter in each object filtering, it was created a mask. For this, first, an image with the same size of the original image was created, with all of its pixels equals to zero, by

using the cvZero() function. After, with the cvDrawContours() method, the object contour was saved on the created image with zeroes, through the following parameters: image where to draw the contour, the contour itself and the color of it. With the masks, each object of interest was separated on independent images by applying the logical operation "AND" between the original image and the before obtained mask. From this result, it was used the cut image operation, which uses the size and position of each object obtained with the cvBoundingRect() function. More precisely, this function creates a rectangle around the object contour, and with cvGetSubRect(), a new image is created with just the internal content of the rectangle. From that, it is saved the image of each object individually with the cvSaveImage() function, given as parameter the file name to be saved and the directory where should be saved the file.

On this way, proposed segmentation method is based on concept of mask, where the non-zero values are the objects of interest on the image. The developed software class diagram can be seen on figure 3.

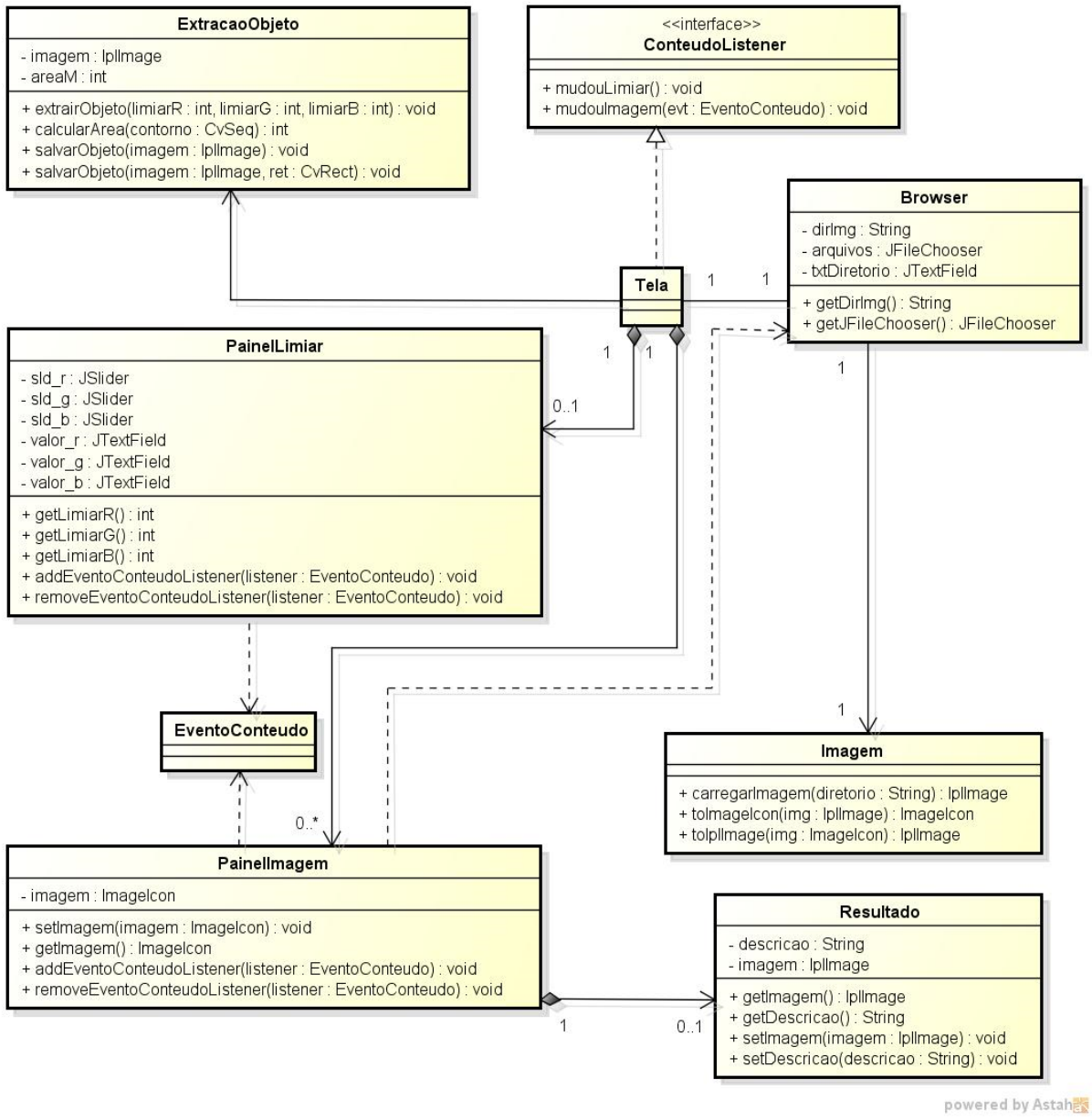


Figure 3 – Software class diagram of the segmentation system.

3.2.2 The Sequence of Use of The System

The application has been divided into three distinct stages of use. The first step refers to the location of the image by the user. To begin this step, user must place the cursor on the word "Arquivo" of the menu, Figure 4, located at the top of the application window, click on it, and then click on "Abrir." During this step, user can navigate through directories by clicking on folders or typing the desired directory in the specified field, as can be seen in Figure 5. After the selection of the desired image, one must click on button 'Abrir', to load image, and, then, the image will appear in the upper left corner of the program.

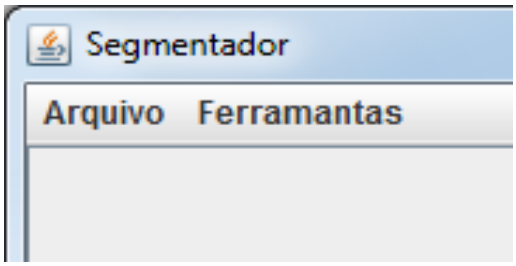


Figure 4 – Main menu of the application.

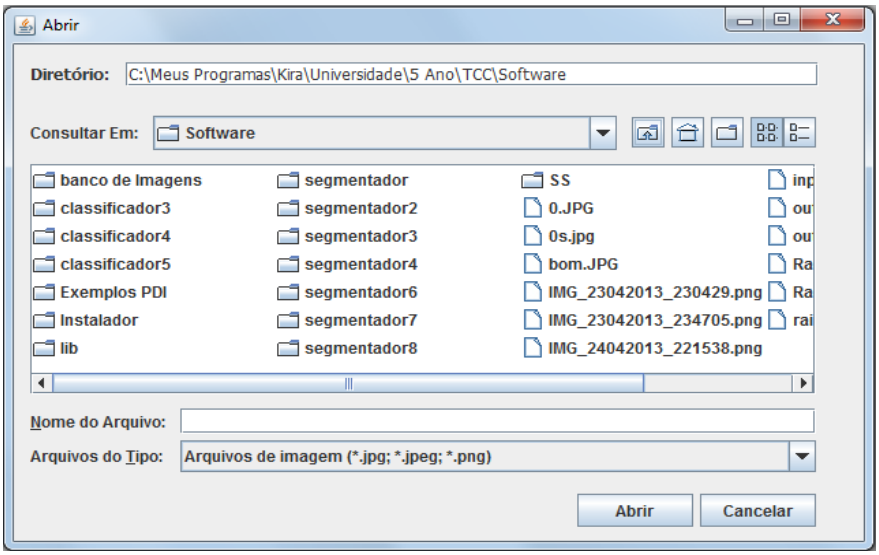


Figure 5 – Example of navigation in directories.

The second step is the formation of the mask. For this, user defines a threshold for each channel RGB, dragging the three sliders (Figure 6). From the selected values for each channel, the image on top corner right is changed, according to the applied thresholding method.

As can also be seen from Figure 6, the thresholded image of the object is recognized by the presence of only black and white pixels. It must highlight that this step is going to define the result of the next step, since it defines which are the objects of interest for the segmentation process.

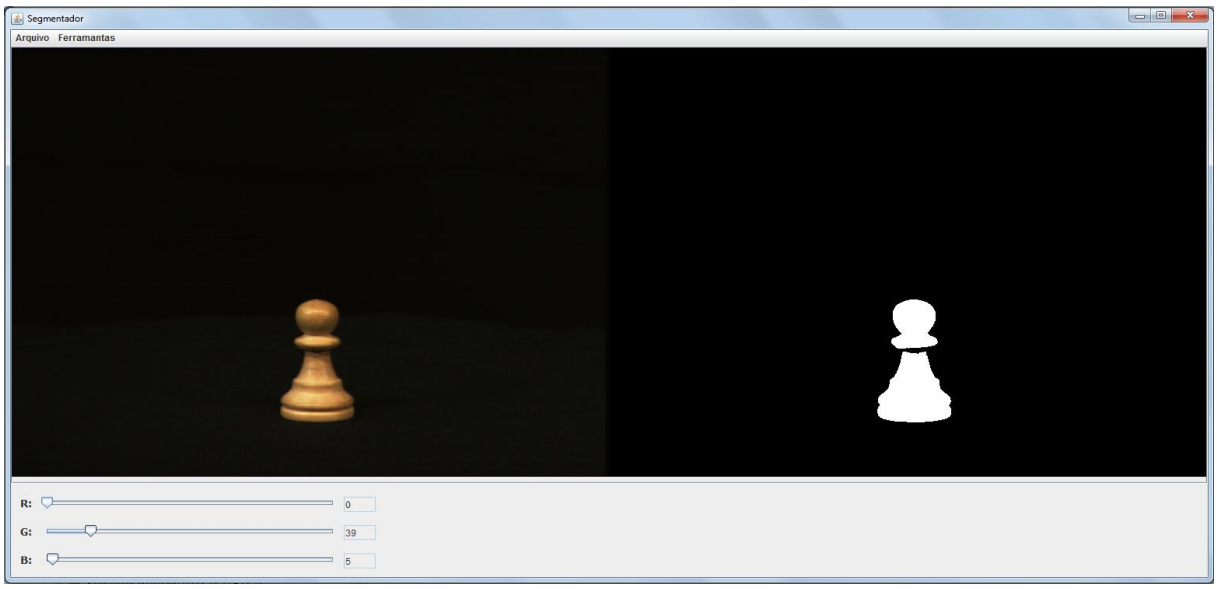


Figure 6 – Demonstration of the application window and its tools.

To perform the last step, it is needed to put the cursor on 'Ferramentas' menu, click on it, and then click on 'Salvar imagem'. With the original image and the image generated in the second step, the elaborated algorithm executes the operations to separate the objects of interest in independent files, according to the white pixels present in thresholded image, and later the resulting images are saved in the 'subimg' application folder.

3.3 Developed Segmentation System Evaluation

Once developed the system, its performance was visually evaluated for three different types of images: image with a single object of interest (from ALOI library); image with two objects of interest and with similar colors; image with several objects with similar colors.

4. Results and Discussion

During the system development, the thresholding step presented a problem. This because the threshold values are for the whole image, and small points, with some pixels (noises), could be considered objects, if they had similar colors to the region or object of interest. This problem was solved with the minimum area that objects should have to be considered an object of interest, section 3.2.1. In this way, we eliminated great part of the isolated points (noise) or, on other words, small occurrences that were not connected or did not belong to the object of interest.

4.1 Image with Single Object of Interest

Figure 7 shows the obtained results with the developed segmentation system when an image with a single object was used, Figure 7(a). Figure 7(b) presents the image of the isolated object, one step before the image cutting. To obtain this result, the original image was loaded and then using the scroll bars, the thresholds were changed until, visually analyzing the binary image, there was only the contour of the object of interest. The threshold values used in this case were 0 (zero) for the R channel, G channel for the 39 and 5 for channel B.

(a) (b)

Figure 7 – Results of the segmentation developed system - step before the image is cut: (a) original image, and (b) image with the isolated object.

Tests with other threshold values resulted that: either part of the object of interest was not separated from the background, or part of the background was segmented along with the object, or occurred of both situations. In summary, for a single object on image, the system showed good results, using proper thresholds, once separated the object of interest from the background.

Figure 8 shows the contour of the object of interest for the segmentation process of Figure 7(a). This image is the one that was subsequently used as a mask to detach the object of interest from the background.

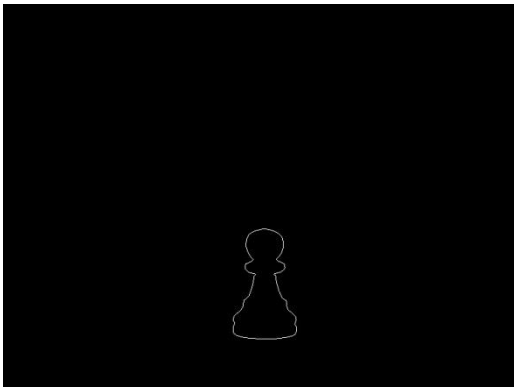


Figure 8 – Mask used to detach the object of interest from the background for image with single object, Figure 7(a).

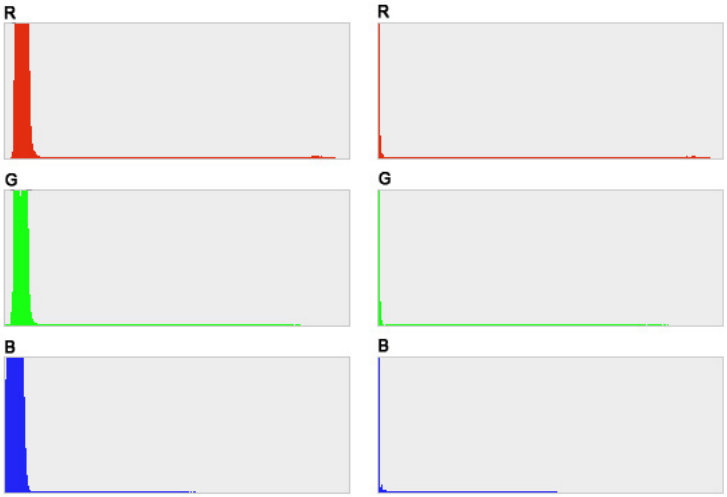
Figure 9 shows the final result of the developed segmentation system. As can be seen, only the object of interest is present in the final image.

Figure 9 – Segmented object.

In the analysis of the image histograms of Figures 7 (a) and 7 (b), Figure 10, we notice a considerable change in the amount of pixels with different color shades. In the original picture there is a large amount of pixels in different dark hues, Figure 10 (a), which did not occur for the segmented image, Figure 10 (b). After segmentation, pixels were practically concentrated in the zero (black); and the remaining pixels, related to the object of interest, were elsewhere.

4.2 Image with Two Objects of Interest and with Similar Colors

Figure 11 shows an image where there is two object of interest, in this case, the birds and the branch, and Figure 12 shows how the birds and branch were separated by the image segmentation system, using the following threshold values: zero for R and G, and 189 for the channel B. To obtain the white background, the code was changed in a manner that the pixels which do not belong to the objects of interest were white.



(a) (b)

Figure 10 – Histograms of three RGB channels of (a) original figure, and (b) segmented figure.

Figure 11- Image with two objects of interest with similar colors.

Font: http://photos1.blogger.com/x/blogger/7/428/400/583633/passaros.jpg.

When objects of interest have shades of similar colors, images with such objects can be segmented in one implementation, due to every area of interest to be located in same threshold

range. In other words, pixels belonging to the background have not equal tones of the objects of interest, in all color channels.

(a) (b)

Figure 12 – Segmentation results of the figure with two objects of interest with similar colors (Figure 11): two images (a) and (b), one for each object, were generated.

This result demonstrates the complexity of developing a generic segmentation system. This case, for example, as the objects of interest have similar colors, it was enough to determine once the threshold values for each channel RBG to get the individual images of all objects of interest.

4.3 Image with Several Objects of Interest with Similar Colors

Figure 13(a) and (b) present, respectively, an image with several objects of interest with similar colors, more specifically: sky, mountains and plantations; and the obtained result with the image segmentation system when one wanted separate the sky from the rest of the image.

(a) (b)

Figure 13 – Segmentation results of the figure with several objects of interest with similar colors: (a) Original image, and (b) image with the sky segmented.

Font (original image): http://files.myopera.com/brokenheartvn/albums/370672/Japan-Hokkaido-Landscape-WUXGA_country_field_0166.jpg.

Even following the object selections after changes in thresholds, it was not possible obtain an image where only the sky was segmented. The threshold values that came closest to fulfilling this desire were respectively 49, 98 and 146, for the R, G and B channel.

As can be saw, for objects of interest with similar color to other parts of image, it is not possible separate these objects from others, in the case, the sky from the mountains. Nevertheless, the other parts of the image parts were discarded because they have different colors of the objects in the upper portion of the image.

In this way, the system had satisfactory result on object separations, but with some restriction: (i) objects with the same background colors can be perceived by the system as the same object, making difficult or even impossible to determine a threshold that separates both; (ii) it is needed execute the program several times on images with more than one object of interest, unless they are in the same thresholding zone, to detach an object or set of objects at a time.

5. Conclusion

We have developed a method for segmentation of images that allows one to modify the background without changing the objects of interest. The performance of this method was satisfactory, executing tasks that have been proposed, especially for situations in which the area, or objects of interest, were in evidence with respect to the background.

It was also evidenced that the result of the segmentation is highly dependent on the choice of threshold, which is a manual process values chosen as thresholds can directly affect the results and also the quality of the images that the developed system produces depends, besides the image itself, which handles user and determines the threshold values.

It was also confirmed that the result of the segmentation process is very dependent on the choice of threshold, which varies for different situations or interests, even for similar images.

References

[1] Albuquerque, M. P.; Albuquerque, M. P. (2000). Processamento de Imagens: Métodos e Análises. Rio de Janeiro.

[2] Audet, S. (2010). API JavaCV. Disponível em: <https://code.google.com/p/javacv/>. Acesso em: 22 fev. 2013.

[3] Bradski, G. ; Kaehler, A. (2008). Learning OpenCV. O'Reilly.

[4] Jory, N. M. (2011). Reconhecimento de Moedas Via Processamento de Imagens. Trabalho de Conclusão de Curso – CEFET- Centro Federal de Educação Tecnológica de Santa Catarina. São José.

[5] Filho O. M.; Neto, H. V. (1999). Processamento Digital de Imagens. Rio de Janeiro: Brasport.

[6] Gonzalez, R. C.; Woods, R. E. (2010) Digital Image Processing (3rd Edition), Prentice-Hall, Inc.

[7] Name, M. H.; de Lima, J. R.; Boff, F. A.; Jaccoud-Filho, D. D. S.; Falate, R. (2013). Histogram Comparison using Intersection Metric applied to Digital Images Analysis. Iberoamerican Journal of Applied Computing, v. 2, n. 1.

[8] Neves, S. C. M (2001). Estudo e Implementação de Técnicas de Segmentação de Imagens. Revista Virtual de Iniciação Acadêmica da UFPA, v. 1, p. 1-11.

[9] Peccini, G.; D'ornellas, M.C. (2004). Segmentação de Imagens por Watersheds: Uma Implementação Utilizando a Linguagem Java. Universidade Federal de Santa Maria.

[10] Geusebroek, J. M.; Burghouts, G. J.; Smeulders, A. W. (2005). The Amsterdam library of object images. International Journal of Computer Vision, v. 61, n. 1, p.103-112.

[11] OpenCV, 2013. Disponível em: <http://opencv.org/about.html>. Acesso em: 18 jun. 2013.