USING PYTHON WITH SIMPLECV TO DETECT A CORN KERNEL IN DIGITAL IMAGE

Sergio Silva Ribeiro, Adriano Ferrasa, Rosane Falate
Universidade Estadual de Ponta Grossa (UEPG) – Ponta Grossa – PR – Brasil
professor@sergioribeiro.com.br, ferrasa@uepg.br, rfalate@yahoo.com

Abstract. *Corn kernels can have their nutritional and economic value hindered by factors such as breakage and rot on the grain. The grain classification, in general, is made by visual inspection of licensed professionals, and is a tedious and tiresome process. Besides, the possibility of error is large, due to visual human subjectivity. In an attempt to reduce this subjectivity by using image processing techniques, this paper presents a computational method for acquiring, preprocessing, and segmentation of corn kernels of digital images. It is expected that, with the proposed program, researchers can have either their focus or their efforts on feature extraction, recognition and interpretation of grains under analysis without spend time with steps related to image manipulation issues.*

Keywords: *digital image processing, segmentation, grain, Zea Mays.*

1. Introduction

Corn (Zea mays) is a very important food for animal and human diets, and is basically composed of starch, protein, fiber and oil. Considered an energy food, its byproducts are used in the composition of margarine, salad dressings, edible oils, breads, etc. It is also used, among others, in the chemical and pharmaceutical industry. The mass of corn kernels ranges from 250mg to 300mg [1, 10].

Corn kernels can have their nutritional, health and economic value hindered by factors such as breakage, crack and rot on grain. However, one of the most worrying problems is related to production of mycotoxins, which occurs due to the fungi presence. These mycotoxins when swallowed can cause severe illness in animals and in humans [7, 13].

Grain classification, in general, is made by visual inspection of a licensed professional, and is a tedious and tiresome process. Furthermore, due to human subjectivity, the possibility of error is large; so, classification of one sample can vary widely [3], [8]. This way, it is of great importance an automated process for detecting and analyzing grain, which can eliminate human subjectivity and improving its accuracy.

Digital Image Processing (DIP) is a computational method used in many areas to help with decision-making by extracting visual information from images [11]. A digital image is a matrix composed of pixels whose positions is given by x and y values, that is, a mathematical function *f (x, y)* [4].

DIP system can be divided in following steps (Figure 1): acquisition, preprocessing, segmentation, feature extraction, recognition, and interpretation. Acquisition is responsible for converting a real scene captured by digital camera or other device in a digital image. In preprocessing some techniques for digital image correction are applied, so that the image can be better processed in the following steps. Segmentation seeks to split an image into regions or objects of interest. This process is guided by object features such as color, texture, and shape. In feature extraction step is generated a database with characteristics of detected objects in segmentation step, such as size, area, and shape. Finally, in recognition and interpretation step, objects detected in the segmentation stage, together with their characteristics, are classified and interpreted according to the desired solution [5, 6].

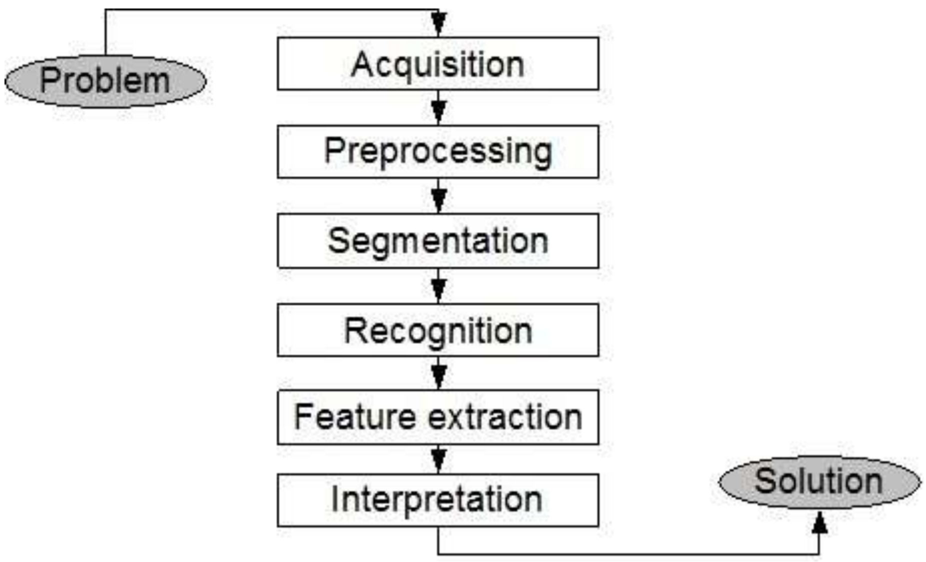


Figure 1 – Steps of a digital image processing system.

For software development solutions that implement the DIP techniques, a programming language which supports the abilities of this technique is necessary. Python is an object-oriented language, with excellent learning curve, which supports development of algorithms for image processing. Furthermore, Python is a language with compact syntax, simple, and multi-platform, which facilitates the use of their programs in automation solutions. SimpleCV is a specific framework of digital image processing for use with Python [2], [12].

The detection of maize grain from a digital image is the first requirement, for systems based on digital image processing, for morphological analysis of the grain, detection of disease, detection of ear rot, or grain classification. This paper presents a computational method for acquiring, preprocessing, and segmentation of corn kernels of digital images. Images generated by the developed method are ready for use in functions that focus on feature extraction, recognition and interpretation of a grain under analysis.

2. Materials and Methods

For this work was used a Notebook ACER Aspire, model 5733-6663, with processor Intel Core i3 and 4GB of memory, Linux UBUNTU 12.04 LTS version. The computational method was developed using Python 2.7.3 [9] and framework SimpleCV 1.3.0 [2]. To write the developed method we used the Stani's Python Editor (SPE), which is a Python IDE wrote in Python available in [14].

2.1 Acquisition

Image acquisitions were made with a scanner SAMSUNG SCX-4600. The software used to acquisition was Xsane Imaging Sanner Program 0.998 with following configuration: File type: png; Background: plane; Color: all; Resolution: 75/300/600 dpi; Size: A4 and location of scanner area: *x* top left value of 0.5 cm, *y* top left value of 0.5 cm, *x* bottom right value of 21.0 cm, and *y* bottom right value of 29.0 cm. To be contrast between grain and the image background we used a blue EVA affixed to the scanner, Figure 2(a), and a blue EVA array with 88 rectangular holes of approximately 1.5 cm (width) x 2.0 cm (h), Figure 2(b), for positioning the grains.

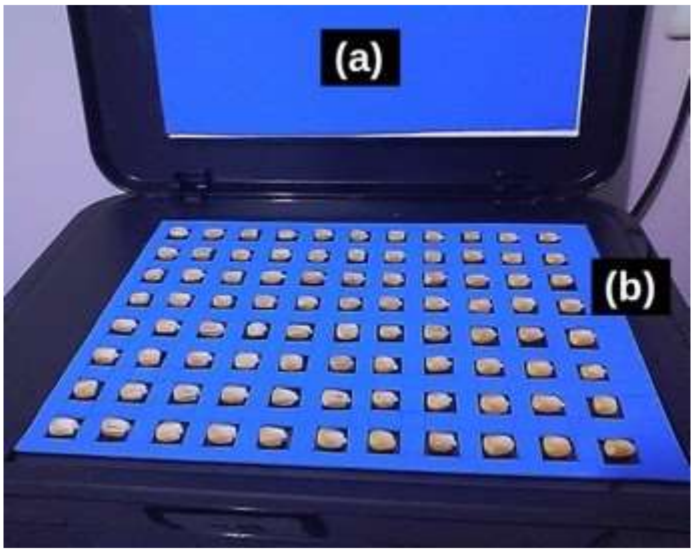


Figure 2 – Image acquisition system with (a) EVA background and (b) EVA array to place grains.

After one put grain in the holes of EVA array, at time of image acquisition, the EVA was removed, i.e., only the grains were scanned. We scan both sides of each grain, named in this work front and back side. Images of the two sides of grain are necessary because corn kernels have different sides and a certain characteristic (object of study) can manifest only on one side.

The scanned images were saved in a folder corresponding to the image resolution. Therefore, an image with a resolution of 75 dpi was saved in a folder called 75DPI. Images with 300 dpi resolution were stored in a folder named 300DPI and images with 600 dpi resolution were placed in the folder 600DPI. The images were saved with the prefix "IMG", followed by a 2-digit sequence number, and then "-F" for front side and "V" to the backside, and the image file extension. For example, the first image of an acquisition was called "IMG01-F.PNG" and the image of the other side of this grain named "IMG01-V.PNG".

After the segmentation process, the resulting images of each grain were stored in IMG subfolder. These images were saved with the prefix "IMG", followed by the 2-digit sequence number of the original image, then the "-" prefix followed by a sequential number of each image (which for this work goes from 1 to 88, the amount of grains or the number of holes in the EVA array). Thus, "IMG01-F.PNG" and "IMG01-V.PNG" files generated images "IMG01-1.PNG" to "IMG01-88.PNG".

2.2 Segmentation

The segmentation process was carried out by a program that uses the Img class (Figure 3), developed in Python. The code for this class is in Algorithm 1.

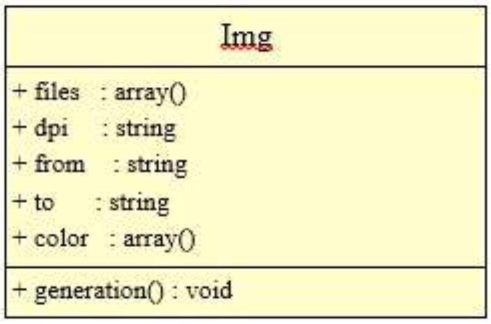
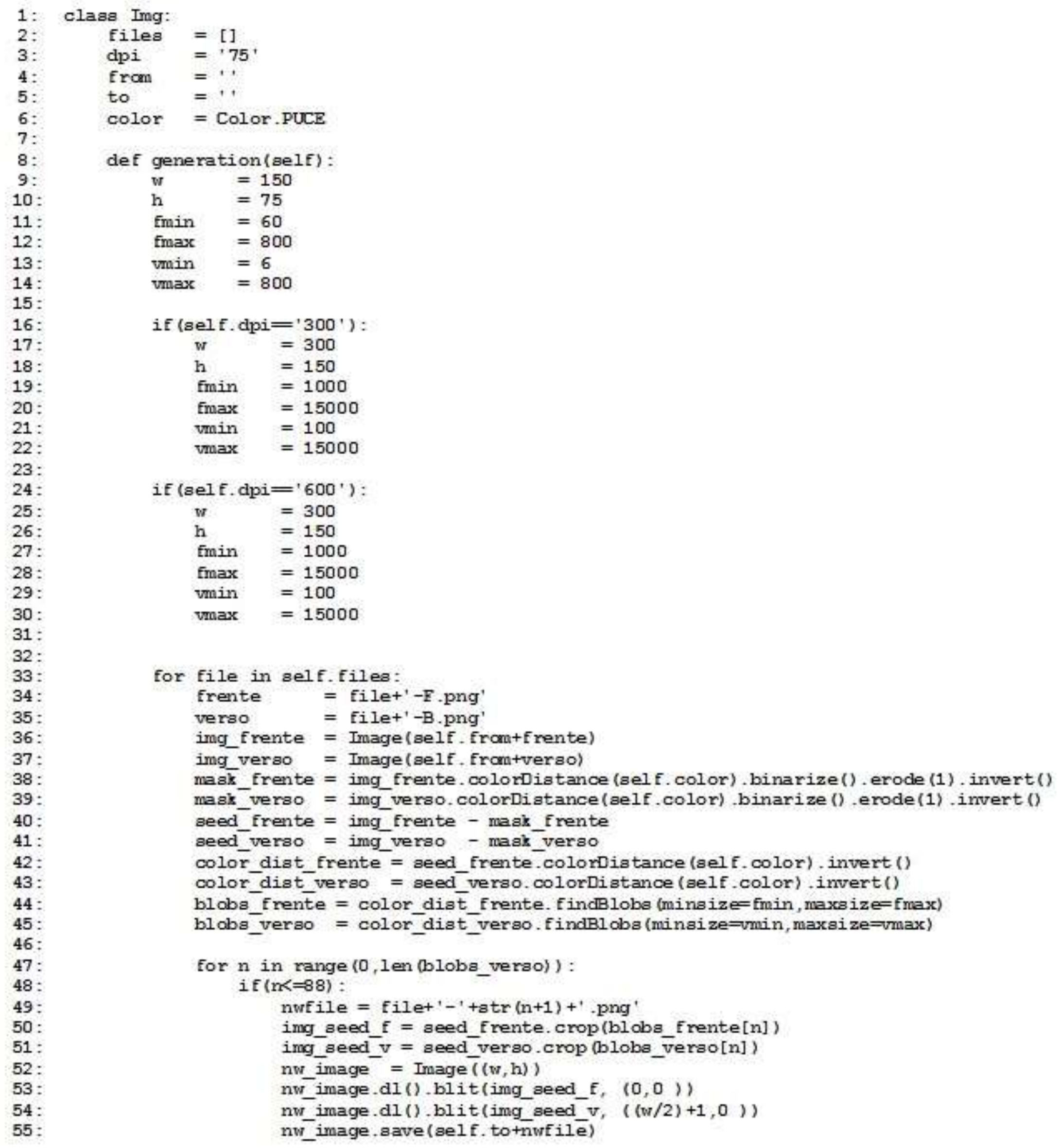


Figure 3 – UML representation of Img class developed.

Algorithm 1 – Source code of Img class



```
 1:  class Img:
 2:      files    = []
 3:      dpi      = '75'
 4:      from     = ''
 5:      to       = ''
 6:      color    = Color.PUCE
 7:
 8:      def generation(self):
 9:          w         = 150
10:          h         = 75
11:          fmin      = 60
12:          fmax      = 800
13:          vmin      = 6
14:          vmax      = 800
15:
16:          if(self.dpi=='300'):
17:              w         = 300
18:              h         = 150
19:              fmin      = 1000
20:              fmax      = 15000
21:              vmin      = 100
22:              vmax      = 15000
23:
24:          if(self.dpi=='600'):
25:              w         = 300
26:              h         = 150
27:              fmin      = 1000
28:              fmax      = 15000
29:              vmin      = 100
30:              vmax      = 15000
31:
32:
33:          for file in self.files:
34:              frente       = file+'-F.png'
35:              verso        = file+'-B.png'
36:              img_frente   = Image(self.from+frente)
37:              img_verso    = Image(self.from+verso)
38:              mask_frente = img_frente.colorDistance(self.color).binarize().erode(1).invert()
39:              mask_verso  = img_verso.colorDistance(self.color).binarize().erode(1).invert()
40:              seed_frente = img_frente - mask_frente
41:              seed_verso  = img_verso  - mask_verso
42:              color_dist_frente = seed_frente.colorDistance(self.color).invert()
43:              color_dist_verso  = seed_verso.colorDistance(self.color).invert()
44:              blobs_frente = color_dist_frente.findBlobs(minsize=fmin,maxsize=fmax)
45:              blobs_verso  = color_dist_verso.findBlobs(minsize=vmin,maxsize=vmax)
46:
47:              for n in range(0,len(blobs_verso)):
48:                  if(n<=88):
49:                      nwfile = file+'-'+str(n+1)+'.png'
50:                      img_seed_f = seed_frente.crop(blobs_frente[n])
51:                      img_seed_v = seed_verso.crop(blobs_verso[n])
52:                      nw_image   = Image((w,h))
53:                      nw_image.dl().blit(img_seed_f, (0,0 ))
54:                      nw_image.dl().blit(img_seed_v, ((w/2)+1,0 ))
55:                      nw_image.save(self.to+nwfile)
```

Between lines 2 to 6 of Algorithm 1 are standard attributes with their initial standard values that can be choose after object creation. The unique and principal method of this class was written from line 8 to end. This method uses 6 principal variables: w for width, h for height, fmin and fmax for minimal and max front image area size, and vmin and vmax for back image area size. These values are changed according to dpi value: lines 16 to 22 for 300 dpi, and lines 24 to 30 for 600 dpi.

In line 33, it is started a loop that processes all files defined in a file attribute. In this loop, the grains are detected in each image, with binarization and erosion techniques, and by the color distance calculation (lines 34-45); and then the images of the front and back of each grain are joined in a new image (line 47 onwards).

Algorithm 2 presents a way to use the Img class and how it works is shown following.

Algorithm 2 – Source of program that uses the Img class.

```
 1:  from SimpleCV import Image,Color
 2:
 3:  nwImg = Img()
 4:
 5:  nwImg.files   = ["IMG01"]
 6:  nwImg.dpi     = '300'
 7:  nwImg.from    = '/home/sergio/Pictures/aquisicoes/'+nwImg.dpi+'DPI/'
 8:  nwImg.to      = nwImg.origem+'IMG/'
 9:  nwImg.color   = Color.ORANGE
10:
11:  nwImg.generation()
```

In line 5 of Algorithm 2 it was passed to nwImage.files an image called IMG01, corresponding to the image to be processed. The location of this file is reported on line 7. How the file names of the grain images will be generated is reported on line 8. Line 9 is set up the orange color for the program makes the detection of grain on the image and its segmentation.

3. Results and Discussions

Figure 4 (a) and Figure 4 (b) respectively show the images of the front side (IMG01-F) and back side (IMG01-V) of the grains with resolution of 300 dpi. The images are not to scale 1: 1.

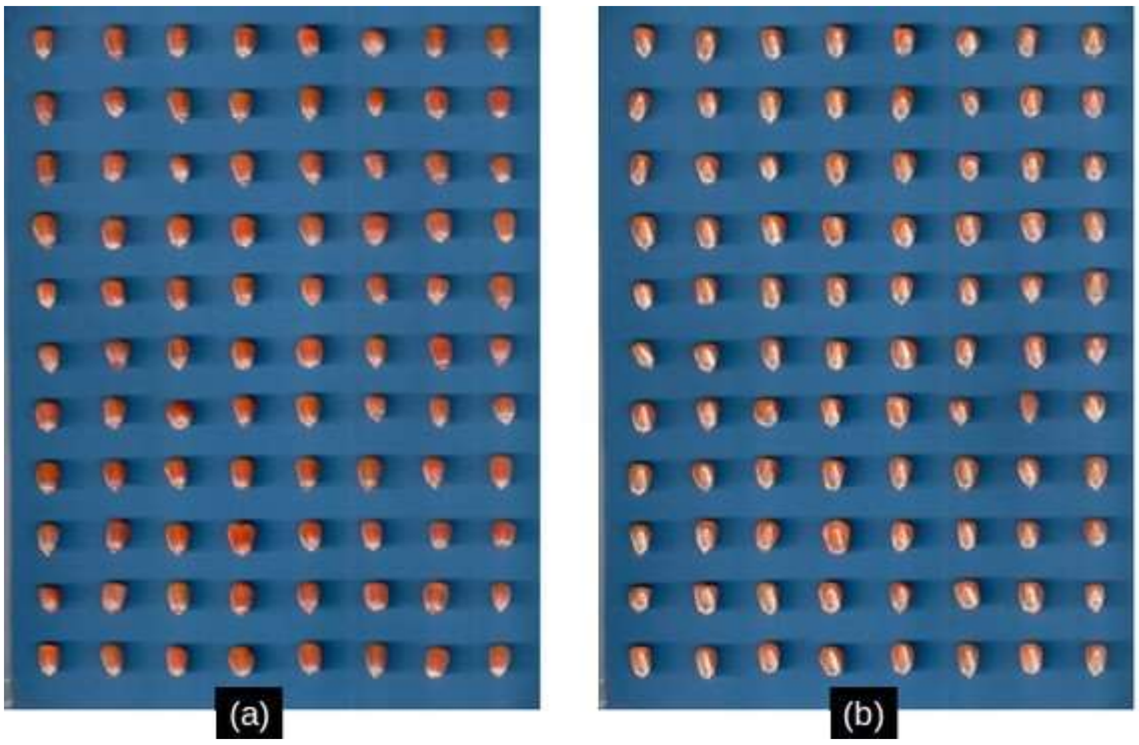


Figure 4 – Images of acquisition of grains: (a) Fron and (b) Back of maize grain.

Figure 5 shows an example of image obtained (from the 88 generated) after the segmentation process of Figure 4 (a) and Figure 4 (b) and the junction of the images on both sides of a given grain.

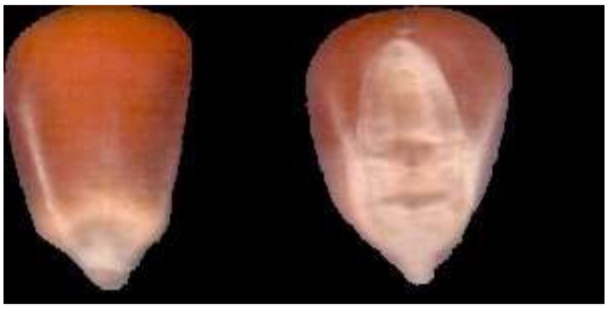


Figura 5 – Front and back side images of a corn grain.

Although it has presented only the results for images with resolution of 300 dpi, the developed method was also able to detect and segment the grains in images with resolutions of 75 and 600 dpi, predefined resolutions for this work.

4. Conclusion

We have developed a computational method for image acquisition, detection and segmentation of corn kernels present in the image in order to assist researchers in studies that are targeted on feature extraction, recognition and interpretation of the grain under analysis. That is, for researchers who require the acquisition and detection of grain corn as a preliminary step.

It was also verified that the Python language in conjunction with the framework SimpleCV can meet the needs of researchers using PDI in their jobs for processing and analysis of images of corn kernels.

Although this work has focused on corn kernels, the methodology can also be extended to other objects of interest.

References

[1] J. C. CRUZ et al. (2011). *Produção de milho na agricultura familiar*. Sete Lagoas: Embrapa Milho e Sorgo.

[2] K. Demaagd; A. Oliver; N. Oostendorp; K. Scott. (2012). Practical *Computer Vision with SimpleCV*. O'Reilly Media, Inc.

[3] T. Draganova; P. Daskalov; R. Tsonev. (2010). Model of Software System for automatic corn kernels Fusarium (spp.) disease diagnostics. *In: Proceedings of the 14th WSEAS international conference on Computers: part of the 14th WSEAS CSCC multiconference-Volume I. World Scientific and Engineering Academy and Society (WSEAS)*, p. 362-367.

[4] R. C. Gonzalez; R. E. Woods. (2008) *Digital image processing*. 3 ed. Cloth : Prentice Hall.

[5] M. Marengoni; S. Stringhini. (2009) Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125-160.

[6] O. Marques Filho; H. Vieira Neto. (1999) *Processamento Digital de Imagens*. Rio de Janeiro: Brasport.

[7] N. F. J. A. Pinto. (2001). Qualidade sanitária de grãos de milho. Sete Lagoas: Embrapa Milho e Sorgo.

[8] P. Potter; J. M. Valiente; G. Andreu-García. (2014) Automatic Visual Inspection of Corn Kernels Using Principal Component Analysis. Available in < http://www2.atb-potsdam.de/cigr-imageanalysis/images/images12/tabla_137_C2357.pdf >.

[9] PYTHON SOFTWARE FOUNDATION. (2012). Welcome to Python.org. Available from <http://www.python.org>. [10 May 2014].

[10] S. S. Ribeiro. (2014). Cultura do Milho no Brasil. Revista Científica Semana Acadêmica, v. 1, n. 49.

[11] S. S. Ribeiro; R. Falate (2014). Uso de Histograma para Detecção de Fusarium no Milho. Revista Científica Semana Acadêmica, v. 1, n. 52, p. 1.

[12] J. E. Solem. (2012). Programming Computer Vision with Python: Tools and algorithms for analyzing images. O'Reilly Media, Inc.

[13] G. Viana. (2009). Milho: novos sistemas de produção e busca por maiores produtividades provocam aumento da severidade das doenças. *Jornal Eletrônico da Embrapa Milho e Sorgo*, Ano 03 - Edição 19, Sete Lagoas-MG.

[14] Stani (2010). SPE IDE - Stani's Python Editor. 17 February 2010. Stani : Blog. Available from < http://pythonide.stani.be>. [22 September 2014].