International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 23 (2017) pp. 13807-13811

Performance Analysis for Algorithms of Recognition of Geometric Patterns in Mechanical Pieces

Henry Montaña Quintero, Holman Montiel Ariza[1] and José Reyes Mozo

Assistant Professors Technological Faculty, District University Francisco José de Caldas
Calle 68 D Bis A Sur No. 49F – 70, Bogotá D.C., Colombia.
[1]ORCID ID: 0000-0002-6077-3510

Abstract

This paper aims to evaluate the performance of two development tools associated with the morphological processing of images (Python and Matlab). It seeks to evaluate the performance by measuring the execution time and the efficiency in the detection of circular geometric patterns associated with mechanical pieces that serve as a structural base in the assembly of bearings. The goal is to implement the same geometric pattern recognition algorithm, evaluating the effectiveness of these development tools, in order to determine the levels of applicability and flexibility of possible solutions linked to image processing regarding production processes in real environments of the productive sector.

Keywords: Morphological image processing, OpenCV, Hough circle Transform.

INTRODUCTION

Day by day the artificial vision techniques implemented in the productive sector are increasing [1], they are used for the classification of parts and products [2], the identification of sizes and colors [3-4], quality control [5-6], among many other activities [7-9]; all of this in order to improve the times and levels of production required by companies to cope with the broad demand established by our society.

However, there are several factors that must be analyzed when performing real-time images processing and even more if these applications are linked to automation processes, factors such as: execution times, implementation resources and development tools. These factors must be evaluated in order to provide an efficient and practical solution to the multiple challenges posed by a diverse work environment [14].

The main objective of this paper is to provide a practical comparison case for two of the most used tools in terms of image processing techniques implementation. The same geometric pattern recognition algorithm will be implemented in the two environments and will be evaluated the execution time and the level of effectiveness on the same number of samples or images to be recognized.

METHODOLOGY

The analysis process focuses specifically on comparing the basic structures of recognition of geometric patterns oriented to circles, both Python language and Matlab; it seeks to analyze the functions established in these development environments in such a way that they identify its requirements, this in order to evaluate its performance at the moment of being used in real applications oriented to the optimization of automation processes for the industrial work environments.

The cv2.HoughCircles Function

This is the pre-established function of OpenCV on python to perform the detection of circles, initially the Hough Transform was developed to identify lines in an image [13], however it was adapted to identify arbitrary shapes such as circles or ellipses [12].

The parameters established for this function, according to the specific documentation of OpenCV are [10]:

HoughCircles (image, circles, method, dp, minDist, param1, param2, minRadius, maxRadius)

Where:

- *image*: Refers to the grayscale input image (8 bits).
- *circles*: Output vector of found circles, where each one is composed by X, Y, Radio.
- *method*: Detection method (HOUGH_GRADIENT).
- *dp*: Inverse ratio of the accumulator resolution to the image resolution.
- *minDist*: Minimum distance between the centers of the circles detected.
- *param1*: It is the highest threshold of the two passed to the Canny Edge detector
- *param2*: It is the threshold of the accumulator for the centers of the circle in the detection stage.
- *minRadius*: Minimum radius of the circle.

- *maxRadius*: Maximum radius of the circle.

Regionprops Function

This function is one of the most used tools in Matlab related to the morphological image processing. In general terms this function measures a set of properties for each region labeled within a binarized image.

The syntax established for this function, according to the specific documentation of Matlab is [11]:

stats = regionprops(BW,properties)

stats = regionprops(CC,properties)

stats = regionprops(L,properties)

stats = regionprops(___,I,properties)

stats = regionprops(output,___)

stats = regionprops(gpuarrayImg,___)

The implementation of this function can be carried out in contiguous and discontiguous regions, applying a wide variety of properties such as: Area, BoundingBox, Centroid, ConvexArea, EquivDiameter, Extent, Extreme, FilledArea, Orientation, Solidity, among many others; always returning a structured arrangement for each object found in the respective image.

IMPLEMENTATION AND RESULTS

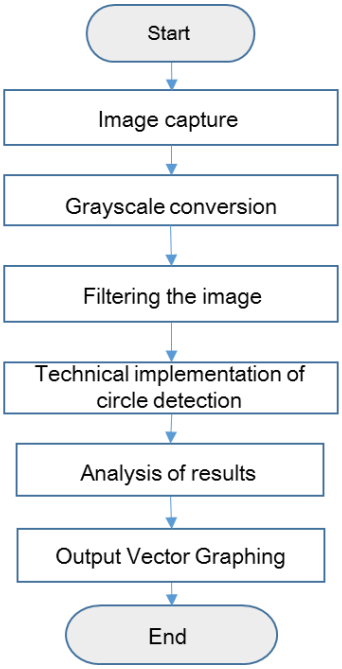


Figure 1: Flow diagram

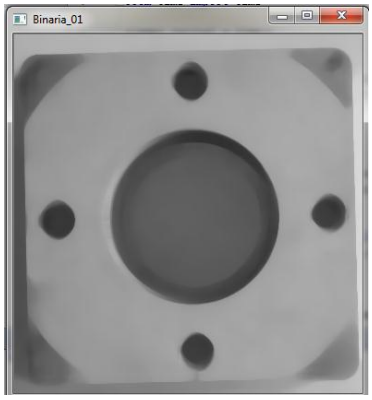
The description of the implemented algorithm is clearly described in Fig. 1. The performance analysis for the detection of geometric patterns established on mechanical pieces begins with the image acquisition, followed by a color conversion process (RGB format) to black and white. Once this conversion has been carried out, the image is filtered, this in order to eliminate noise components in the information matrix to be analyzed; once the information matrix is ready, the pre-established detection algorithms are applied, in order to finally analyze the obtained information and be able to graph it in order to validate the response time and the detection of the predetermined objects.

The procedure described above is executed in both Python and Matlab; it will be proceeded to evaluate both execution time and the graphing of results.

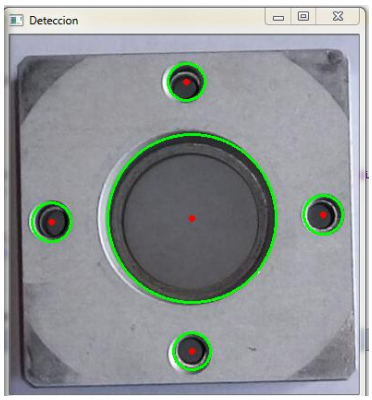
Algorithm developed in Python

```
import cv2
import numpy as np
from time import time

tiempo_inicial = time()
im_in = cv2.imread('muestra_d.jpg')

gray_img = cv2.cvtColor(im_in,
cv2.COLOR_BGR2GRAY)
img_bin = cv2.medianBlur(gray_img, 17)

cv2.namedWindow("Binaria_01")
cv2.imshow("Binaria_01", img_bin)

circles = cv2.HoughCircles(img_bin,
cv2.cv.CV_HOUGH_GRADIENT,1,120,param1=100,par
am2=30,minRadius=0,maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles [0,:]:
   cv2.circle(im_in,  (i[0],i[1]),i[2],(0,255,0),2)
   cv2.circle(im_in,  (i[0],i[1]),2,(0,0,255),2)

print circles
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print 'El tiempo de ejecucion fue:',tiempo_ejecucion
#En segundos

cv2.imwrite('deteccion.jpg',im_in)
cv2.imshow("Deteccion", im_in)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

This algorithm is executed on JetBrains PyCharm Community Edition 2016.3.2, a development tool for freely distribution for python, which was loaded with the OpenCV library that directly contained the HoughCircles Function, obtaining the results shown in Fig. 2.



a)



b)

Figure 2: a) binarized image b) detection of circles on Python.

The second part of the analysis focused on implementing the same stages described in Fig. 1 on Matlab, a graphical user interface (GUI) was designed on the 32-bit R2013b version where the following code was executed:

Algorithm developed in Maltab

```
% --- Executes on button press in aplicar.
function aplicar_Callback(hObject, eventdata, handles)
% hObject    handle to aplicar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB

global imagen seleccion diam

 %%%% Filtros para mejorar la imagen
tiempo_inicio = cputime;
ima_gris = rgb2gray(imagen);
vector=find(ima_gris<=120);

   for(i=1:1:length(vector))
      ima_gris(vector(i))=0;
   end

vector=find(ima_gris>120);

   for(i=1:1:length(vector))
      ima_gris(vector(i))=255;
   end

im_bin=~im2bw(ima_gris);
image = imfill(im_bin,'holes');

%%%% captura de parámetros por el usuario

%diámetro mínimo permitido
diam = str2num (get(handles.edit1,'String'));
diam_pix = 0.626664577*diam;
diam_rad = diam_pix/2;

%%%% Detección

%%%% Detectar Areas
   [L Ne]=bwlabel(image);
   propied= regionprops(L);
   s=find([propied.Area]>=500);

   %%%% Graficar
   axes(handles.axes2)
   imshow(imagen)
   for n=1:size(s,2)
      %rectangulo
```

```
caja=propied(s(n)).BoundingBox;
%rectangle('Position',caja,'EdgeColor','c','LineWidth',2)

%circulo
circulo=propied(s(n)).Centroid;
dif1=abs(caja(1)-circulo(1));
dif2=abs(caja(2)-circulo(2));

if dif1<dif2
   dif=dif2;
else
   dif=dif1;
end

if dif<100
   if diam_rad>dif
      viscircles(circulo,dif,'Edgecolor','b')
   else
      viscircles(circulo,dif,'Edgecolor','r')

   end
  end

end

tiempo_ejecucion = cputime - tiempo_inicio
```

The results obtained in this development tool can be seen directly in the GUI designed, see Fig. 3, where you can see the image entered and the resulting image in a single work form.

Figure 3: GUI designed in MATLAB with displayed circle detection

The developed algorithms were validated with a total of 40 samples of mechanical parts, it was validated that the detection efficiency is directly linked to the configuration and detection ranges associated with each preset function, meaning it must have associated calibration values to the scale in pixels that it is wanted to detect. However, it is important to emphasize that the execution times obtained after the calibration process and validation of the values used in the algorithms, guaranteed a detection efficiency of 100%, see Table 1.

Table 1: Average algorithm execution time

Development Tool	Execution time (miliseconds)
Python	35
Matlab	115

The validation tests run on a Windows 7, 32-bit operating system, with an Intel Core I5-4200U CPU 2.3 GHz processor and 6 Gb RAM.

CONCLUSIONS

The performance analysis carried out on the two development tools associated with image processing and the contours detection of geometric figures, determined for this particular case that Phyton takes just 30.43% of the execution time in comparison with Matlab for the detection of the same circular patterns, demonstrating in this way the robustness and practicality of this tool, with a view to adapting practical and effective solutions that can be presented in diverse industrial environments of the productive sector.

ACKNOWLEDGMENTS

This work was supported by the District University Francisco José de Caldas Technological Faculty. The views expressed in this paper are not necessarily endorsed by District University. The authors thank the research group ARMOS for the evaluation carried out on prototypes of ideas and strategies.

REFERENCES

[1] Nandini, V., Deepak, R., Arun, C. and Aishwarya, S.: A Review on Applications of Machine Vision Systems in Industries. In: Indian Journal of Science and Technology, 9 (48), pp. 1-5 (2016)

[2] Padmavathi, K., Thangadurai, K.: Implementation of RGB and Grayscale Images in Plant Leaves Disease Detection – Comparative Study. In: Indian Journal of Science and Technology, 9 (6), pp. 1-6 (2016)

[3] Amaya, D., Rojas, A. and Gutierrez, D.: Algorithm for Detection of Aonidiella aurantii in Citrus × tangelo Fruits using DIP Technique. In: Indian Journal of Science and Technology, 10 (27), pp. 1-6 (2017)

[4] Lopez, F., Andreu, G., Blasco, J., Aleixos, N., Valiente, J.: Automatic detection of skin defects in citrus fruits using a multivariate image analysis approach. In: Computers and Electronics in Agriculture. 71(2), pp.189–97 (2010)

[5] Wu, D. and Sun, D.: Colour measurements by computer vision for food quality control – A review. In: Trends in Food Science & Technology,29 (1), pp. 5-20 (2013)

[6] Brosnan, T. and Sun, D.: Improving quality inspection of food products by computer vision—a review In: Journal of Food Engineering, 61 (1), pp. 3-16 (2004)

[7] Kumar, T., Verma, K.: A theory based on conversion of RGB image to gray image. In: International Journal of Computer Applications. 7(2), pp.7–10 (2010)

[8] Bonin-Font, F., Ortiz, A., Oliver, G.: Visual Navigation for Mobile Robots: A Survey. In: Journal of Intelligent and Robotic Systems. 53(3). pp. 263 (2008)

[9] Montiel, H., Mártinez, F. and Jacinto, E.: Visual Patterns Recognition in Robotic Platforms Through the Use of Neural Networks and Image Processing. In: International Journal of Applied Engineering Research,12 (18), pp. 7770-7774 (2017)

[10] OpenCV (2015). Documentation Open Source Computer Vision 3.1.0. https://docs.opencv.org/3.1.0/

[11] MathWorks (2017). Documentation regionprops. https://www.mathworks.com/help/images/ref/regionprops.html

[12] Bradski, G. and Kaehler, A.: Learning OpenCV. First Edition O'Reilly Media, Inc. United States of America. (2008)

[13] Mukhopadhyay, P., Chaudhuri, B.: A survey of Hough Transform. In: Pattern Recognition. 48(3), pp.993–1010 (2015)

[14] Bonin-Font, F., Ortiz, A., Oliver, G.: Visual Navigation for Mobile Robots: A Survey. In: Journal of Intelligent and Robotic Systems. 53(3). pp. 263 (2008)