Inspection of Hexagonal Profile Screw Head using Image Processing Technique

Amar Yekane[1],Sudhanva Kulkarni[2],Ashwini Vaidya[3], Pratik Kamble[4]
[1, 4] Asst. Professor in Department of Mechanical Engineering, FAMT,India
[2, 3]UG Student in Department of Mechanical Engineering, FAMT,India

Abstract -The paper represents an algorithm for segregation of defective screw heads (non-hexagonal punch) using algorithm in python .In this work image of the screw is captured.The image for defect free component has been compared with the image of screw which is to be inspected. Based on the image quantification technique, the decision making has been made for acceptance and rejection of screw. The image processing is one of the effective tool which is used as a basis for screw head inspection.Although the process is bit of complex, but it's assures guaranty and reduces process time. The algorithm testing is executed in computer using python software and the algorithm shows the promising results.

Keywords – Hexagonal Profile of screw, Algorithm in Python, Raspberry Pi

1. INTRODUCTION

Image processing or so called image quantification is a widely used technique. Mostly this technique is used to check the quality of fruit which are being exported. The images of fruits are processed with previous data to eliminate the defected fruit. Nowadays this technique is used in the field of Medical for identification of diseases, in geology for identification of minerals and etc. Here we have applied this technique for identification of bolts with perfect hexagonal profile. The images are captured using a photographic lens along a photographic sensor for image capturing and later processed using Arduino board and other electronic circuitry. It is estimated that the system gives only 99.99 % of accuracy.

2. LITERATURE REVIEW

Image processing technique is used for excavation for archeological purposes by predicting the piece of land to be dig by image processing and satellite mapping methods. [1]

Analysis of image by computer is discussed by the author. The pixelating of image by major colours as RBG is the main focus of the author. By using matrix mathematical model the image is analyzed and can be used as a reference for image quantification technique.[2]

The research given by Heather Dunlop says that image processing is a valuable tool in geology for identification of different rocks, crystals and minerals.[3]

The image processing algorithm given by Domingo Mery helps in identification of defective fruits by its surface peel image, this process is very helpful in the export of fruits and vegetables.[4]

3. RESEARCH METHODOLOGY

The perfect screw is set as a sample for image processing and the image of rest of the screws are compared with the master image , which decides the screw is accepted or rejected.

For this process programming language used is python, which is user friendly.The processing of the image will be done by electronic system which includes camera and Raspberry Pi processing board.

3.1 COMPONENTS AND SOFTWARE USED

The software used in programming is "Python 3.7"version. The software and Library function allows the camera to capture the image in real time and the Raspberry pi processor compares the image with the master image and decide whether the screws hexagonal profile is perfect or defective. Fig 3.1 and 3.2 shows the specifications of the screw while Fig3.3 shows 2D CAD model of the screw.

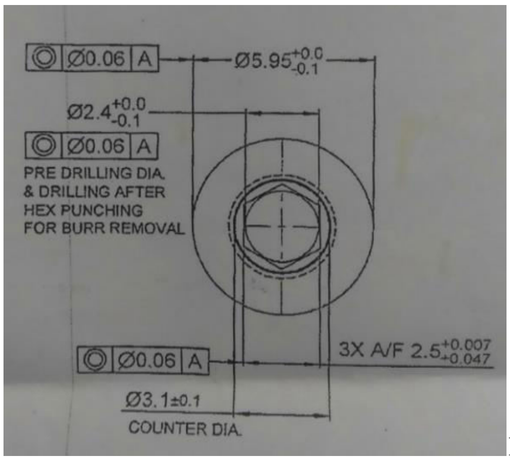


Fig 3.1: Specifications of the Screw

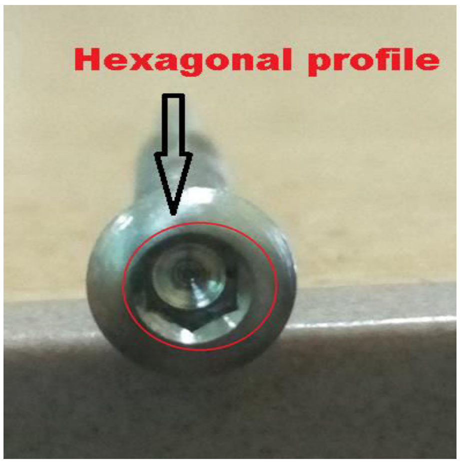


Fig 3.2: The Image of screw head to Be Tested

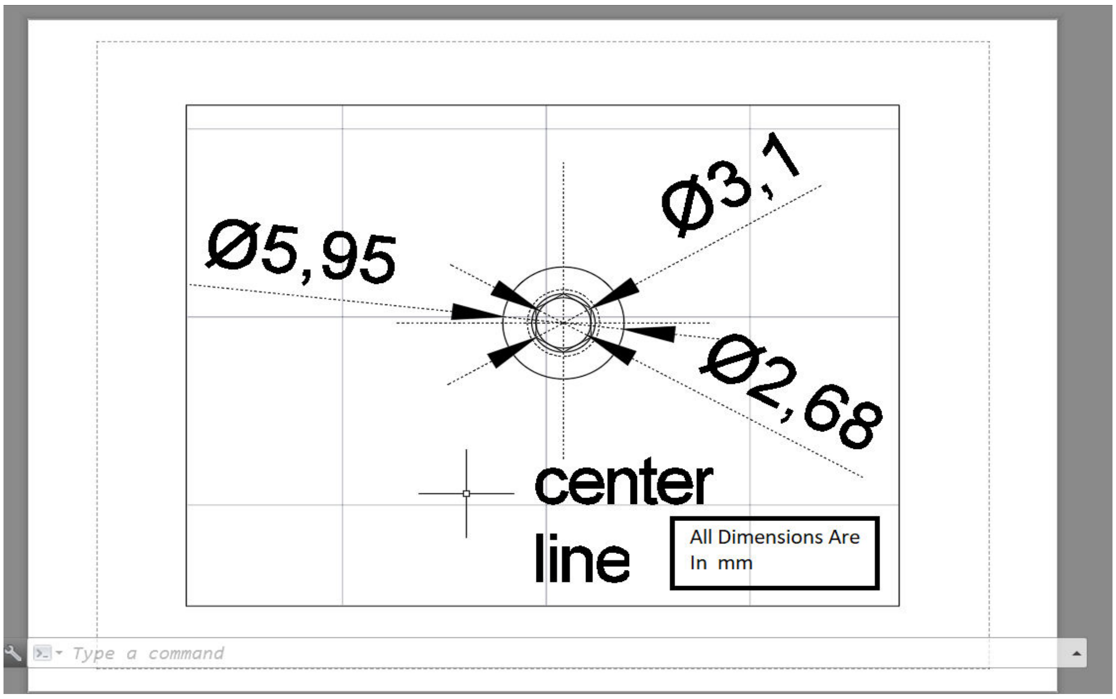


Fig 3.3: 2D AUTO CAD Drawing Of Screw

Table 3.1: The list of components is as given below

Sr.no.	**List of Components**
1)	Raspberry pi 4GB model
2)	Memory card 16GB (Class C)
3)	Web Cam Module 8 MP
4)	Jumper cables

Table 3.1 shows the list of components to be needed.According to Raspberry Pi organization following steps should be followed for connecting the raspberry pi camera module to the board. Fig 3.4 shows schematic model of Camera module with Raspberry Pi.

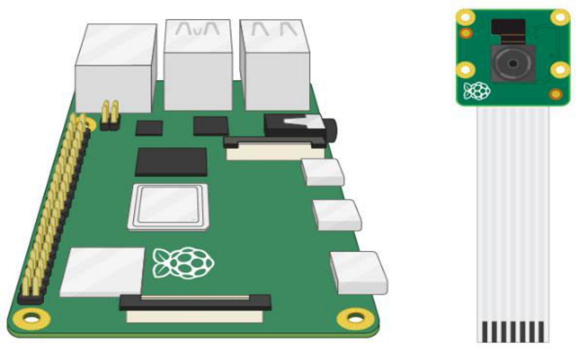


Fig 3.4 Interfacing Camera Module with Raspberry PI

i. Ensure your Raspberry Pi is module off.
ii. Locate the camera Module port.
iii. Gently pull up on the edge of the port's plastic clip.
iv. Insert the camera Module ribbon cable; make sure the cable is the right way around. (Fig 3.5 shows actual Camera and Raspberry Pi connected)
v. Push the plastic clip back into place.

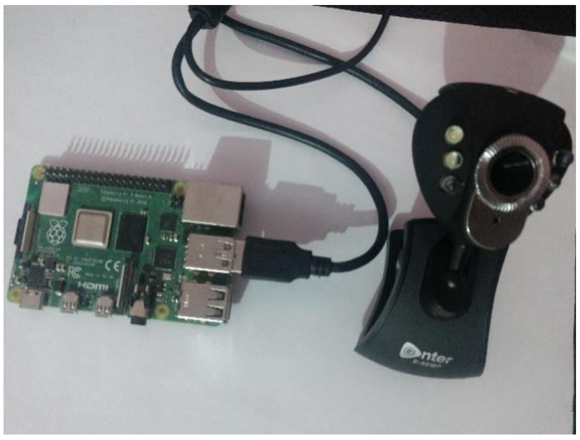


Fig 3.5 Connecting Camera Module to Raspberry Pi

vi. Start up your Raspberry Pi.
vii. Go to the main menu and open the Raspberry Pi configuration tool.
viii. Select the Interfaces tab and ensure that the camera is enabled.
ix. Reboot your Raspberry Pi.

3.3 Shape detection using OpenCV

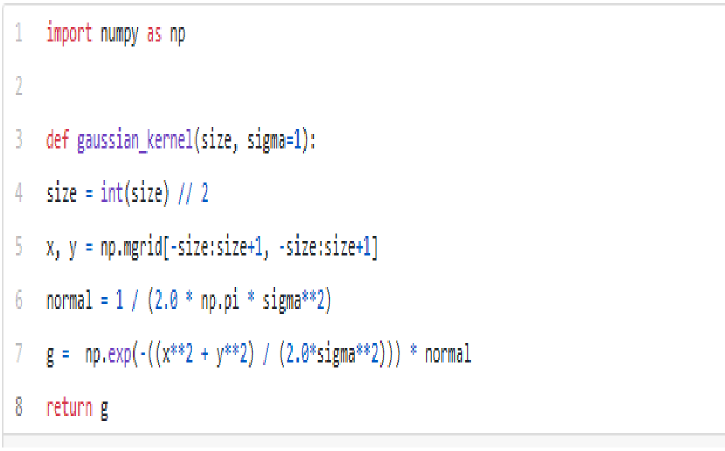
The very first step using Allen key is identification of the hexagonal shape on screw head. OpenCV provides the simple techniques for detection of the shape using basic techniques. The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

x. The general criteria for edge detection include:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the centre of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

The shape detection through OpenCV consist of following steps.

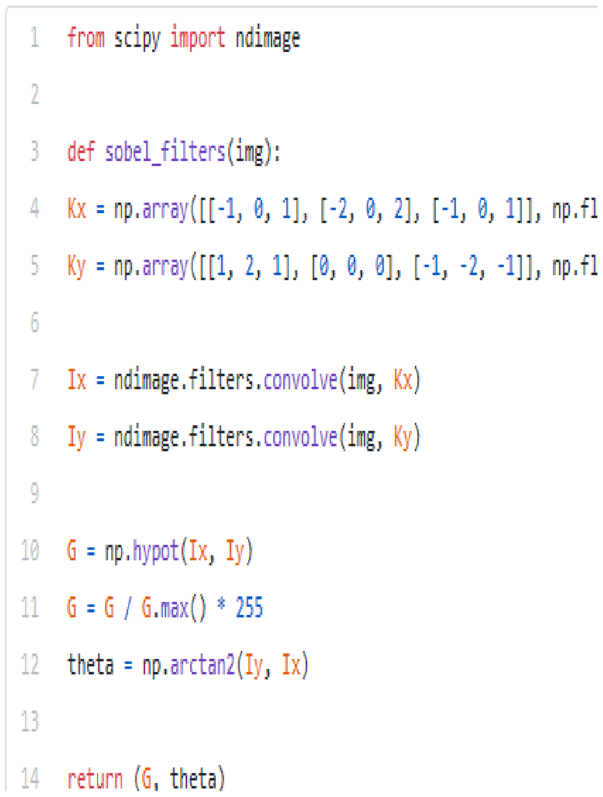
i. **Noise Reduction:** Since the mathematics involved behind the scene are mainly based on derivatives (cf. Step 2: Gradient calculation), edge detection results are highly sensitive to image noise. One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it. To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc…). The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur. In our example, we will use a 5 by 5 Gaussian kernel. The code for Gaussian 5x5 kernel is given below in Fig 3.6.



```
import numpy as np

def gaussian_kernel(size, sigma=1):
size = int(size) // 2
x, y = np.mgrid[-size:size+1, -size:size+1]
normal = 1 / (2.0 * np.pi * sigma**2)
g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
return g
```

Fig 3.6: Python code to generate the Gaussian 5x5 kernel

ii. **Gradient Calculation:** The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators. Edges correspond to a change of pixels' intensity. To detect it, the easiest way is to apply filters that highlight this intensity change in both directions: horizontal (x) and vertical (y). The code for Gradient Calculation is given below in Fig 3.7 .

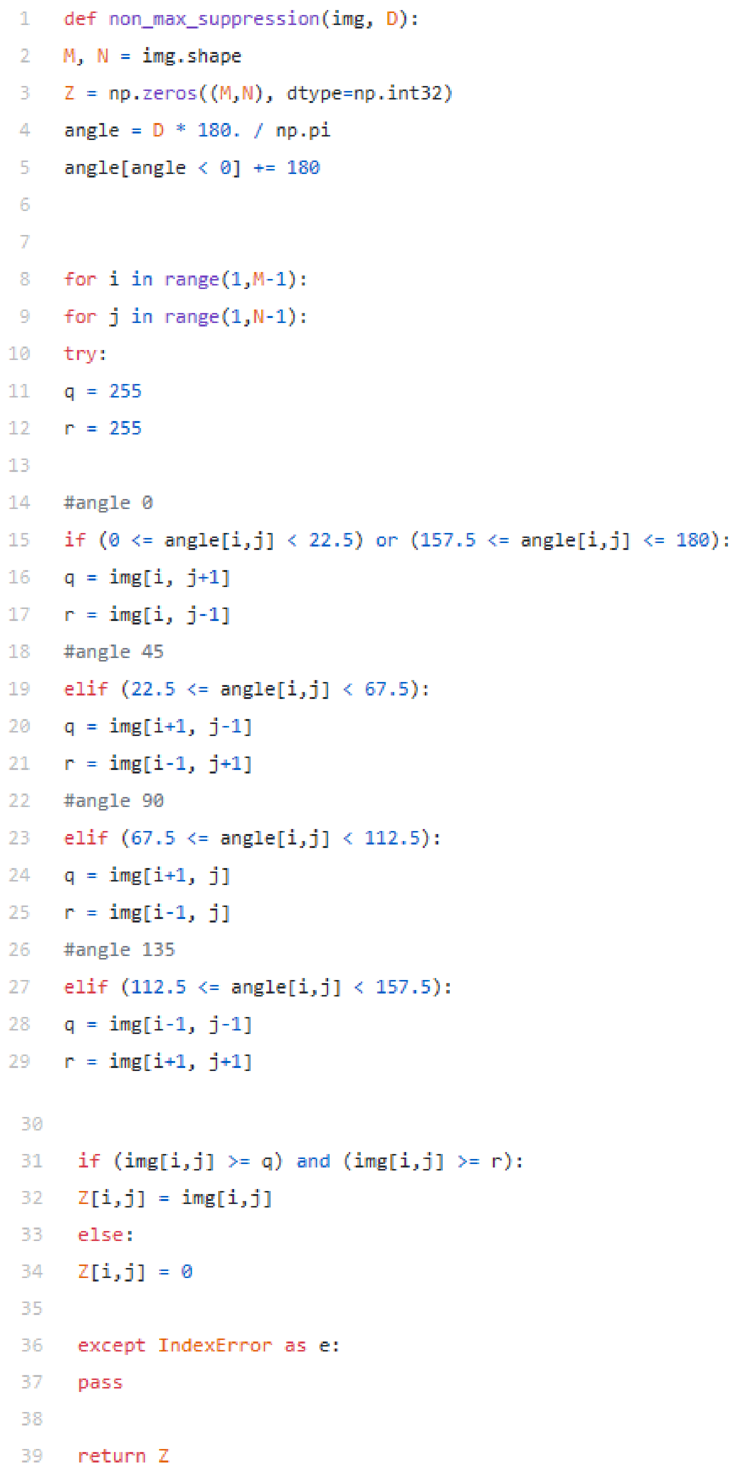


```
from scipy import ndimage

def sobel_filters(img):
Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.fl
Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.fl

Ix = ndimage.filters.convolve(img, Kx)
Iy = ndimage.filters.convolve(img, Ky)

G = np.hypot(Ix, Iy)
G = G / G.max() * 255
theta = np.arctan2(Iy, Ix)

return (G, theta)
```

Fig 3.7 Python code for gradient calculation

iii. Non-Maximum Suppression: Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges.

The principle is simple: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions. The python code for generating object edges is given in Fig 3.8 .



```
def non_max_suppression(img, D):
M, N = img.shape
Z = np.zeros((M,N), dtype=np.int32)
angle = D * 180. / np.pi
angle[angle < 0] += 180


for i in range(1,M-1):
for j in range(1,N-1):
try:
q = 255
r = 255

#angle 0
if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
q = img[i, j+1]
r = img[i, j-1]
#angle 45
elif (22.5 <= angle[i,j] < 67.5):
q = img[i+1, j-1]
r = img[i-1, j+1]
#angle 90
elif (67.5 <= angle[i,j] < 112.5):
q = img[i+1, j]
r = img[i-1, j]
#angle 135
elif (112.5 <= angle[i,j] < 157.5):
q = img[i-1, j-1]
r = img[i+1, j+1]

if (img[i,j] >= q) and (img[i,j] >= r):
Z[i,j] = img[i,j]
else:
Z[i,j] = 0

except IndexError as e:
pass

return Z
```

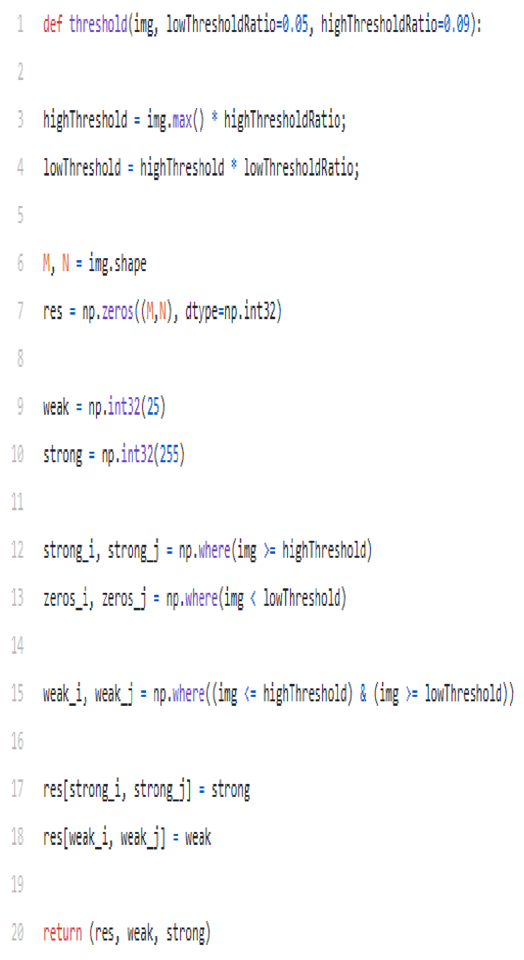
Fig 3.8 Python code for generating edges of an object

iv. **Double Threshold:** The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant

Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.

Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.

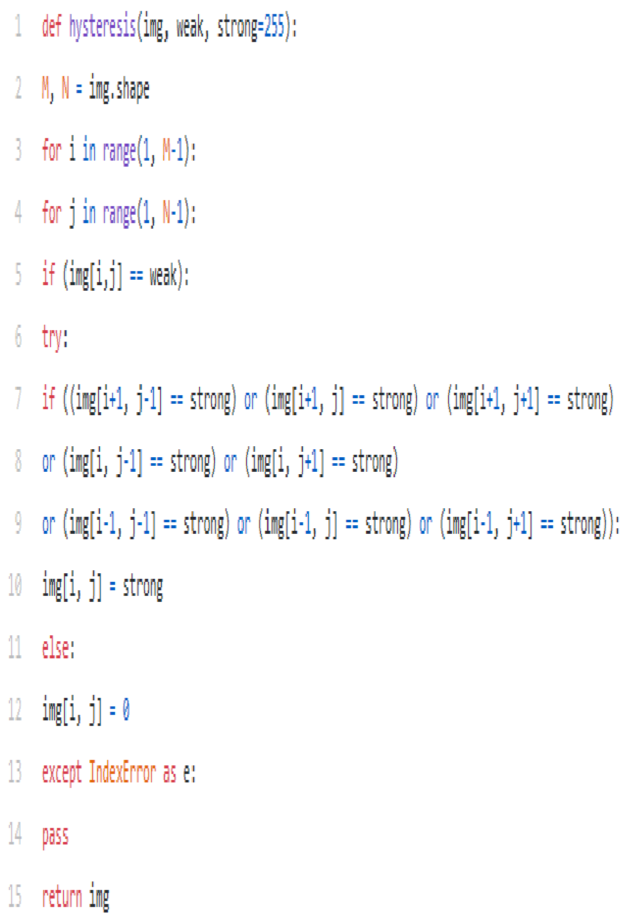
Other pixels are considered as non-relevant for the edge. The image (Fig 3.9) shows image processing code for Double Threshold.



```
def threshold(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):

highThreshold = img.max() * highThresholdRatio;
lowThreshold = highThreshold * lowThresholdRatio;

M, N = img.shape
res = np.zeros((M,N), dtype=np.int32)

weak = np.int32(25)
strong = np.int32(255)

strong_i, strong_j = np.where(img >= highThreshold)
zeros_i, zeros_j = np.where(img < lowThreshold)

weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

res[strong_i, strong_j] = strong
res[weak_i, weak_j] = weak

return (res, weak, strong)
```

Fig 3.9 Python code for double threshold

v. **Edge Tracking:** Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one. Fig 3.10 shows coding for hysteresis.

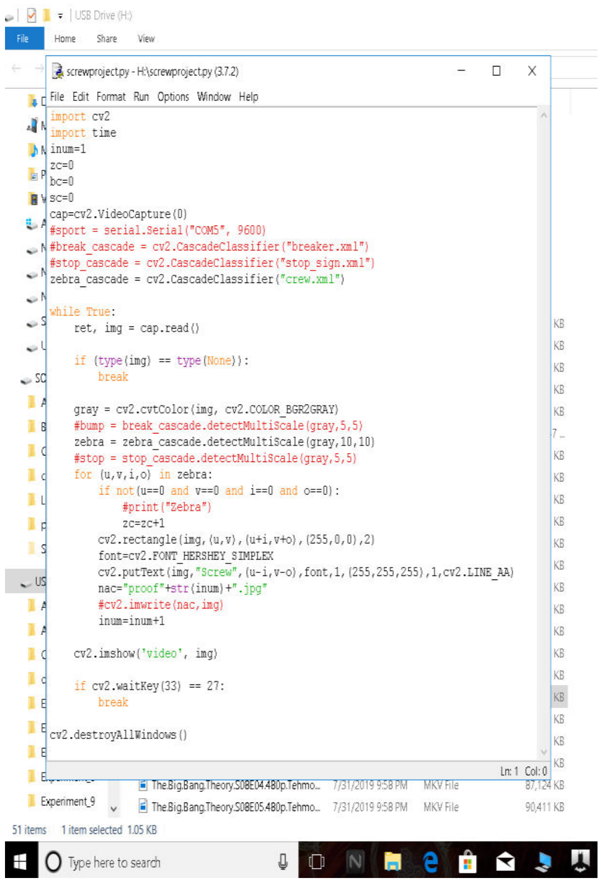


```
def hysteresis(img, weak, strong=255):
M, N = img.shape
for i in range(1, M-1):
for j in range(1, N-1):
if (img[i,j] == weak):
try:
if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1, j+1] == strong)
or (img[i, j-1] == strong) or (img[i, j+1] == strong)
or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[i-1, j+1] == strong)):
img[i, j] = strong
else:
img[i, j] = 0
except IndexError as e:
pass
return img
```

Fig 3.10 Python code for hysteresis

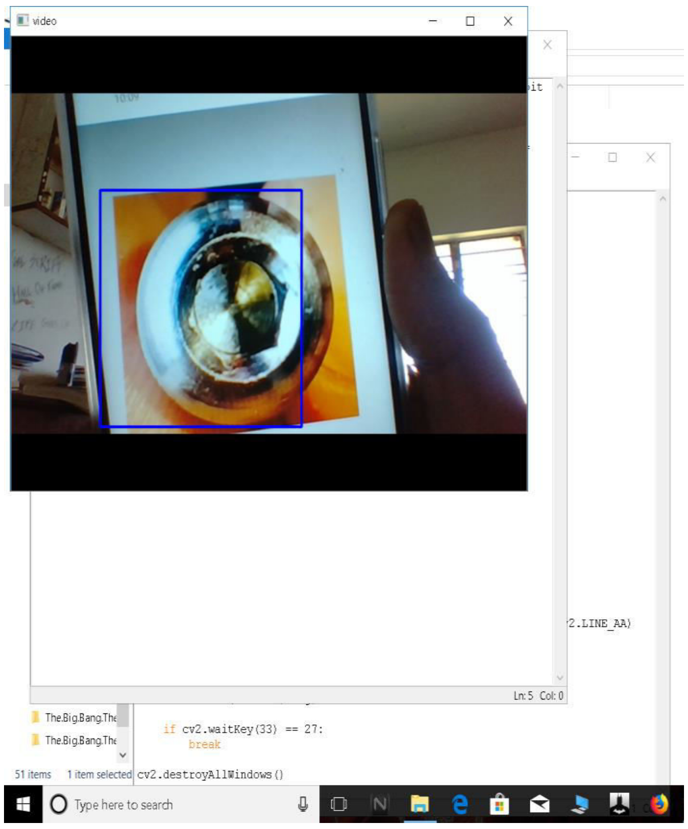
4. Results from computer processing

As mentioned earlier, python is used for coding of image processing technique, the results can be seen in the images. The program is capable of identifying screw and its hexagonal profile. The program can identify the hexagonal on the screw head and if it is not found then the screw is rejected. Fig 4.1 shows actual complete code for the operation of Raspberry Pi and Camera module.



```
import cv2
import time
inum=1
zc=0
bc=0
sc=0
cap=cv2.VideoCapture(0)
#sport = serial.Serial("COM5", 9600)
#break_cascade = cv2.CascadeClassifier("breaker.xml")
#stop_cascade = cv2.CascadeClassifier("stop_sign.xml")
zebra_cascade = cv2.CascadeClassifier("crew.xml")

while True:
    ret, img = cap.read()

    if (type(img) == type(None)):
        break

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #bump = break_cascade.detectMultiScale(gray,5,5)
    zebra = zebra_cascade.detectMultiScale(gray,10,10)
    #stop = stop_cascade.detectMultiScale(gray,5,5)
    for (u,v,i,o) in zebra:
        if not(u==0 and v==0 and i==0 and o==0):
            #print("Zebra")
            zc=zc+1
        cv2.rectangle(img,(u,v),(u+i,v+o),(255,0,0),2)
        font=cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img,"Screw",(u-i,v-o),font,1,(255,255,255),1,cv2.LINE_AA)
        nac="proof"+str(inum)+".jpg"
        #cv2.imwrite(nac,img)
        inum=inum+1

    cv2.imshow('video', img)

    if cv2.waitKey(33) == 27:
        break

cv2.destroyAllWindows()
```

Fig 4.1 Image processing codes for PC



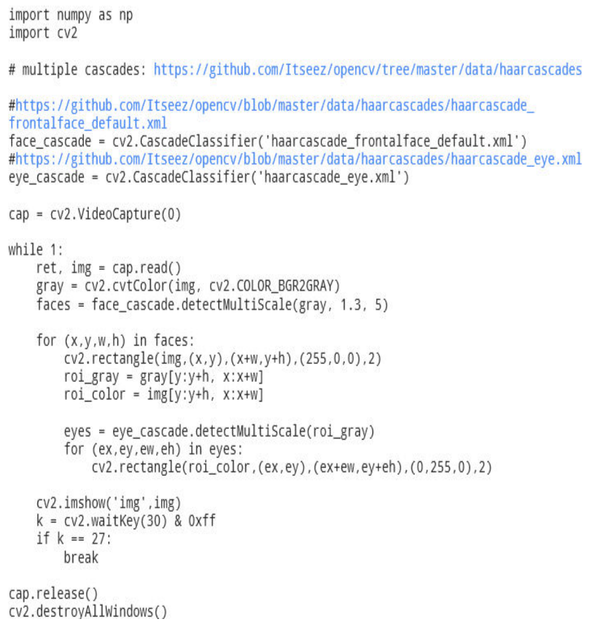
```
    if cv2.waitKey(33) == 27:
        break
cv2.destroyAllWindows()
```

Fig 4.2 sample results of the program on PC

4.1 Results on PC

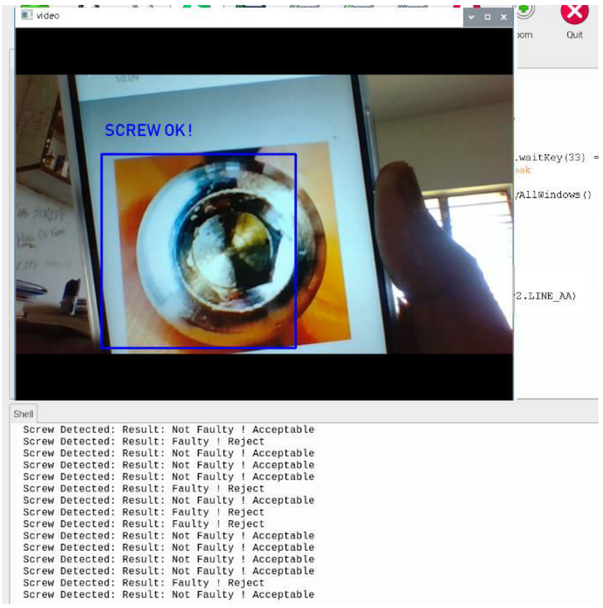
Before carrying out the experiment on the Raspberry pi the simulations were ran on PC. The image (Fig 4.2) given below shows the sample results on PC.

4.2 Results on Raspberry Pi



```
import numpy as np
import cv2

# multiple cascades: https://github.com/Itseez/opencv/tree/master/data/haarcascades

#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_
frontalface_default.xml
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#https://github.com/Itseez/opencv/blob/master/data/haarcascades/haarcascade_eye.xml
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

cap = cv2.VideoCapture(0)

while 1:
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]

        eyes = eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    cv2.imshow('img',img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Fig 4.3 Image processing codes for Raspberry Pi



```
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Faulty ! Reject
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Faulty ! Reject
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Faulty ! Reject
Screw Detected: Result: Faulty ! Reject
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Not Faulty ! Acceptable
Screw Detected: Result: Faulty ! Reject
Screw Detected: Result: Not Faulty ! Acceptable
```

Fig 4.4 Raspberry pi output

As shown in (Fig 4.3) the Electronic circuit (Raspberry Pi, Camera Module, etc) is able to identify the Hexagonal punch on Screw head and tell whether the Screw Head is OK or not. Which shows us that the Program Codes (Fig 4.4) have no error and can identify whether the screw is Ok or not.

5 Conclusions

From obtained results on PC and Raspberry pi it can be concluded that the separation of the defected screws is possible by image quantification process. The results given by python programming are accurate and efficient. As the images taken real time images using web-cam results are more exact. Processing speed is electronic circuit is fast enough to accept or reject the screw by its hexagonal profile. The time and cost of the company will be saved and will also give rise in efficiency. The production rate hasbeen increased. This image quantification and processing technique has shown significant results saving human fatigue along with productivity enhancement.

6 References

1. Salomon Cesar Nguemhefils, e.t.al. “Radarsat-1 image processing for regional-scale geological mapping with mining vocation under dense vegetation and equatorial climate environment, Southwestern Cameroon Salomon.” (2018)
2. Fernando Mendoza and Renfulu. “Basics of Image Analysis”. January (2015).
3. Heather Dunlop. “Automatic Rock Detection and Classification in Natural Scenes”. August (2006).
4. Domingo Mery e.t.al.,“Segmentation of Colour Food Images using a Robust Algorithm”, April (2004).