SCIENTIFIC VISUALISATION
Project 1- Image processing

Chandrasekaran, Vivek
Matriculation No. 4941693
Exam code: BRC102

20th September, 2019

Contents

1 Problem 1 **2**
1.1 Problem statement . . . 2
1.2 Solution for filtering: . . . 2
1.2.1 Red Content: . . . 3
1.2.2 Blue Content: . . . 4
1.2.3 Green Content: . . . 5
1.2.4 Gray scale: . . . 6
1.3 Solution for Counting Annual Growth Rings: . . . 7
1.3.1 Step 1: Gaussian Blur . . . 7
1.3.2 Step 2: Gray scale filtering . . . 8
1.3.3 Step 3: Laplace fitering . . . 8
1.3.4 Step 4: Finding the center . . . 8
1.3.5 Step 5: Counting the rings in the horizonatal direction . . . 9
1.3.6 Step 6: Counting the rings in the vertical direction . . . 10

Python code for the project: **13**

1 Problem 1

1.1 Problem statement

Take one of the files wood-1.bmp or wood-2.bmp in the stud.ip folder and create new bmp files for i) red content, ii) blue content, iii) green content, iv) converted to grey scale. Apply a laplace filter to the file with grey scale, store the result in a bmp file. Rings should now be visible which represent the growth in each year. Write a code which counts the rings in two different directions (from the center of the piece of wood). Submit the bmp files as well as the program code and a documentation of the result (how is the result obtained?).

1.2 Solution for filtering:

For this project the file wood-2.bmp is chosen. All the following solutions are pertaining to the same.

Figure 1: Given wood-2.bmp file

Step 1: The program code for processing the image is written in Python 3.7.0 using an additional 'cv2' package. The image can be read in a matrix form with corresponding pixel data using the 'matplotlib.pyplot' or the 'cv2' package as given below:

```
im = plt.imread('wood_2.bmp')
im = cv2.imread('wood_2.bmp',0)
```

Step 2: The read image is copied using function 'deepcopy' in order to enable manipualtion of pixel data in matrix form. The respective code is given below:

```
im1 = deepcopy(im)
```

Step 3: The copied image is then sent ot the respective filtering functions whose return vlaues are saved in a new variable. The processed is then converted to an image, saved and displayed in the output window. Following is an example that fulfils this purpose:

```
red = Red(im1)        # Calling the Red function
misc.toimage(red).save('Red.bmp') # Saving red content image
plt.imshow(red)
```

1.2.1 Red Content:

To create a 'Red Content Image' from the wood-2.bmp file, a class 'Red()' is created to take an 2D-image input and make the green and blue values of every pixel as '0'. The function is defined as given below:

```
def Red(image):          # Variables received as parameters
        image[:,:,1] = 0        # Making green content 0
        image[:,:,2] = 0        # Making blue content 0
        return image            # Returning the image
```

The result of such filtering is as shown in fig(2):

Figure 2: Red content image

1.2.2 Blue Content:

To create a 'Blue Content Image' from the wood-2.bmp file, a class 'Blue()' is created to take an 2D-image input and make the green and red values of every pixel as '0'. The function is defined as given below:

```
def Blue(image):
        image[:,:,0] = 0    # Making red content 0
        image[:,:,1] = 0    # Making green content 0
        return image
```

The result of such filtering is as shown in fig(3):

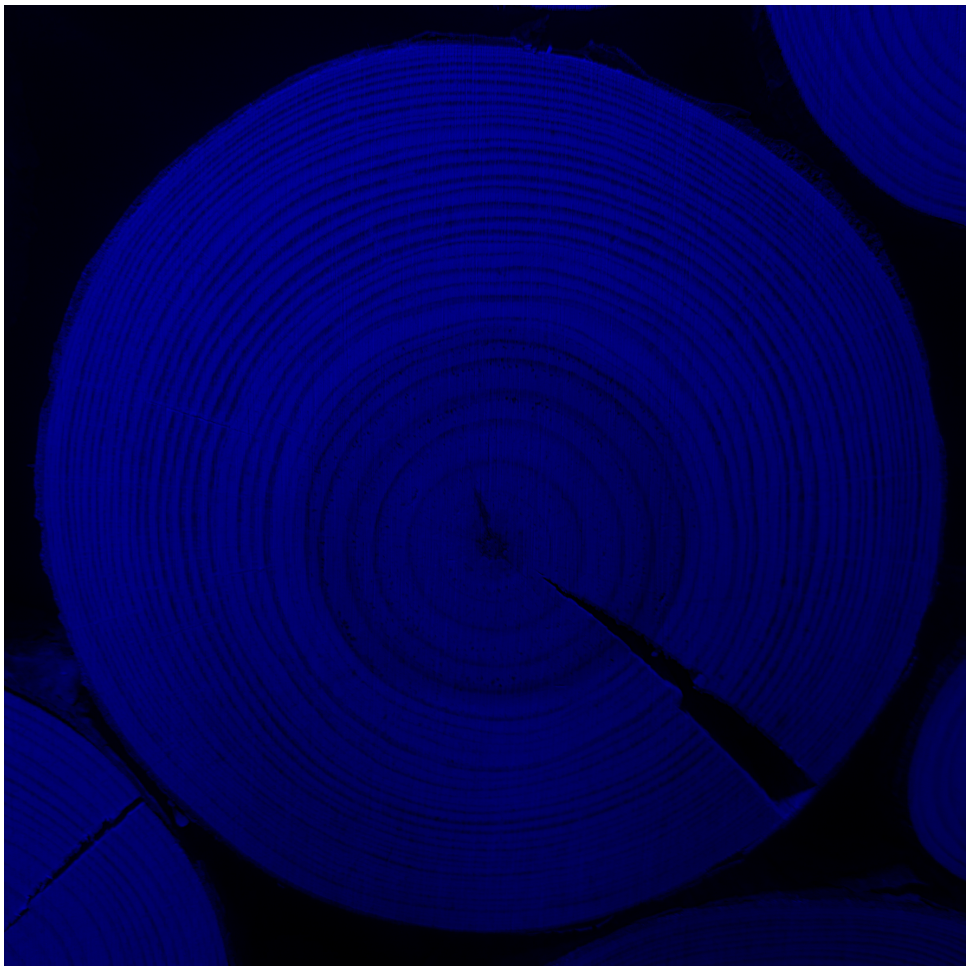


Figure 3: Blue content image

1.2.3 Green Content:

To create a 'Green Content Image' from the wood-2.bmp file, a class 'Green()' is created to take an 2D-image input and make the blue and red values of every pixel as '0'. The function is defined as given below:

```
def Green(image):
        image[:,:,0] = 0    # Making red content 0
        image[:,:,2] = 0    # Making blue content 0
        return image
```

The result of such filtering is as shown in fig(4):

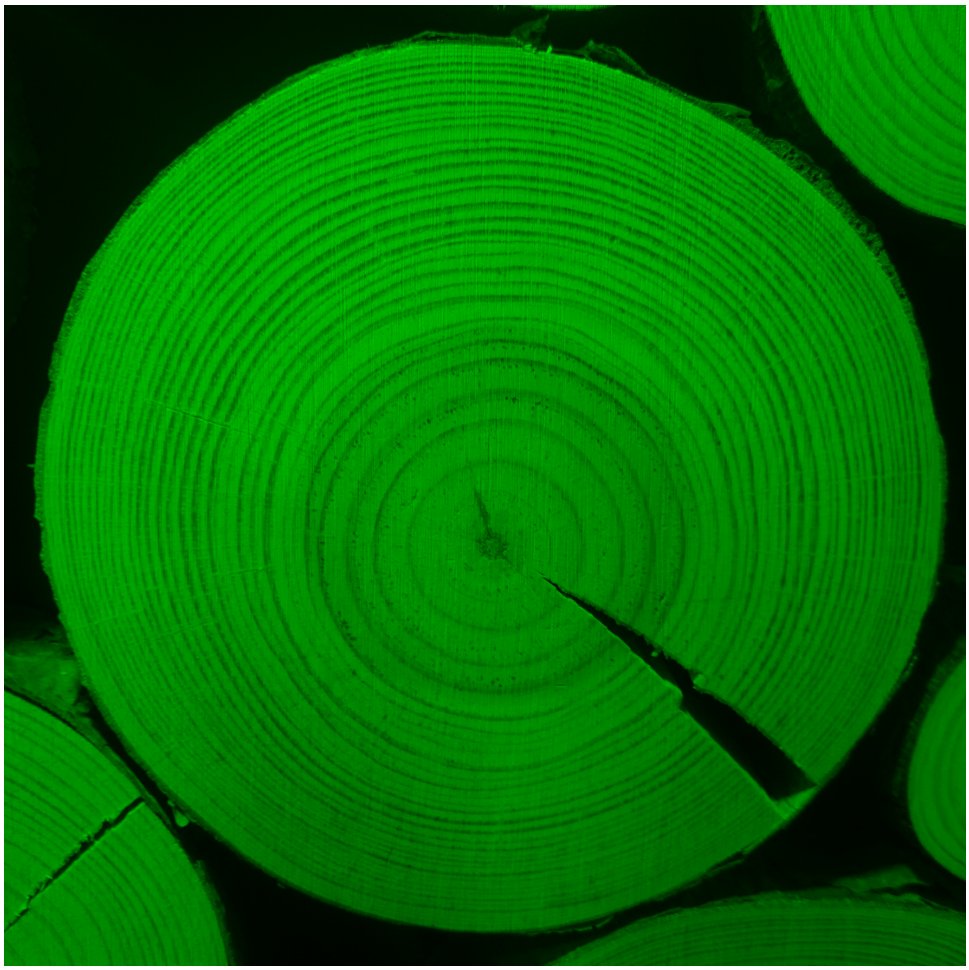


Figure 4: Green content image

1.2.4 Gray scale:

The method used for making the image gray is based on 'Luminosity Average' in which pixel values of Red, Green and Blue are multiplied by [0.299, 0.587, 0.114] respectively and then added. The same is explained in the reference[1]. The average result obtained is a 2-d matrix which is made 3-d again by giving same values to the third dimension for RGB. The function to do the same is defined as given below:

```
def Gray(image):
        # Luminosity average for getting agray image
        avg = np.dot(image[...,:3], [0.299, 0.587, 0.114])
        # Storing the average 2-d in all 3 axes
        image[:,:,0] = avg
        image[:,:,1] = avg
        image[:,:,2] = avg
        return image
```

The result of such filtering is as shown in fig(5):

Figure 5: Gray scale image

1.3 Solution for Counting Annual Growth Rings:

The number of rings in the horizontal direction was found to be **31**.
The number of rings in the vertical direction was found to be **31**.
The average number of rings is **31**.
The step by step solution is explained below:

1.3.1 Step 1: Gaussian Blur

The given image is filtered using 'Gaussian blur' function from the cv2 package with kernel 3x3, which is a Gaussian filter. This is done in order to blur out pixel irregularities in the given image. This enables us to get a slightly better resolution of the rings in the wood profile by Laplace filtering. The same is explained in the reference[2] and is referred to as 'Laplace of Gaussian'. The filtering is done as given by the code below:

```
im5=deepcopy(im)
gauss = cv2.GaussianBlur(im5,(3,3),0)
misc.toimage(gauss).save('Gauss.bmp')
```

The filtered image after Gaussian blur is as given in figure(6)

Figure 6: Gaussian blur image

1.3.2 Step 2: Gray scale filtering

The gaussian filtered image is then converted to a gray scale image using the 'Gray' function from section 1.2.4.

1.3.3 Step 3: Laplace fitering

The gray scale image is then laplace filtered. Using the 'Laplacian' function from the cv2 package. the code for the same is given below:

```
im = plt.imread('Gray2.bmp')
im5 = deepcopy(im)
laplacian = cv2.Laplacian(im5,cv2.CV_64F)
misc.toimage(laplacian).save('lap_fil.bmp')
```

The filtered image after Laplacian is as given in figure(7)

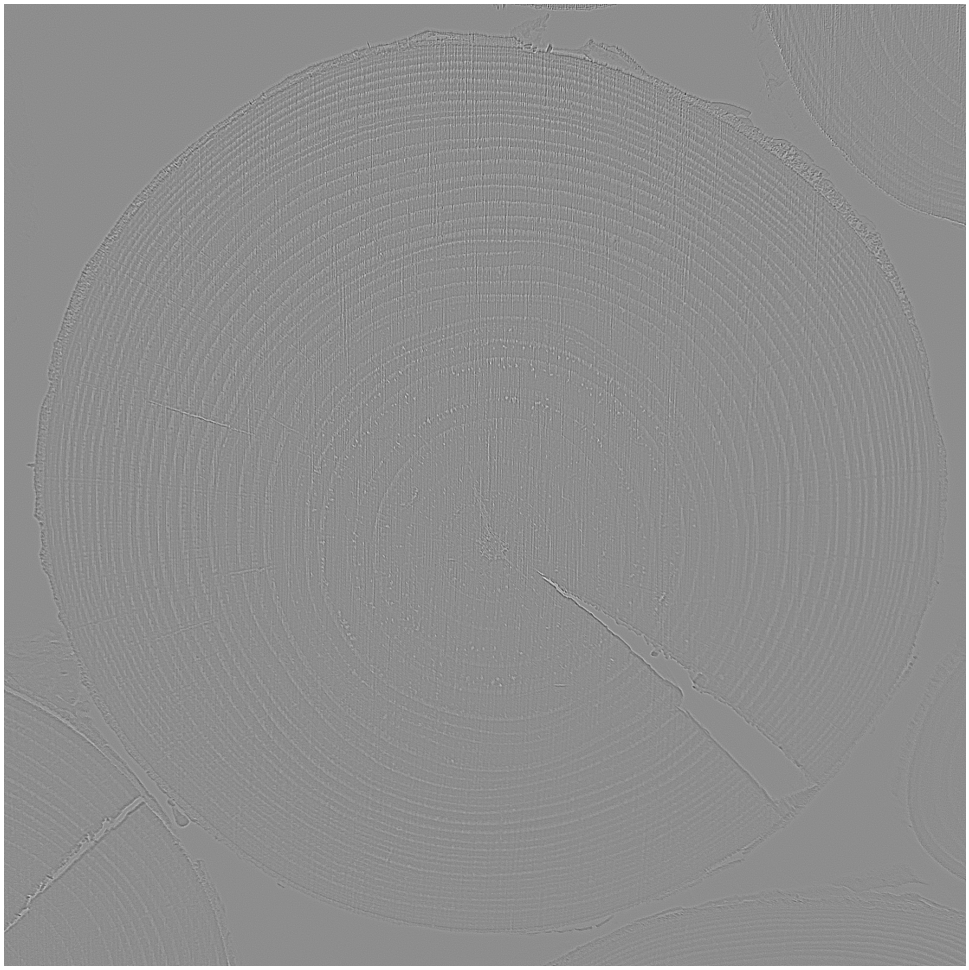


Figure 7: Laplace filtered image

1.3.4 Step 4: Finding the center

First the centre of the wood profile is determined. This was done by creating a crosshair with veritcal and horizontal lines . The following is the code for the same:

```
imy = cv2.imread('wood_2.bmp',0)
centering = deepcopy(imy)
centering[1020,:]= 0 #horizontal black crosshair
centering[:,900] = 0 #vertical black crosshair
misc.toimage(centering).save('centering.bmp')
```

The image with the cross hairs is exagerrated in the given figure (8).The centre was found to be the [1020, 900] pixel, measured from the left corner in positive-x and negative-y directions. The cross hairs, originallly black are shown in red in figure(8) for better readability.

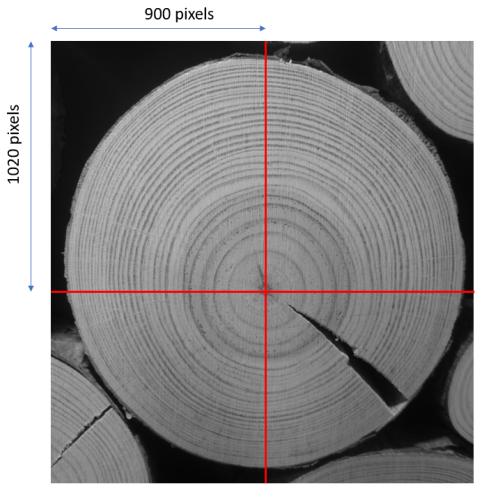


Figure 8: Center of the wood profile

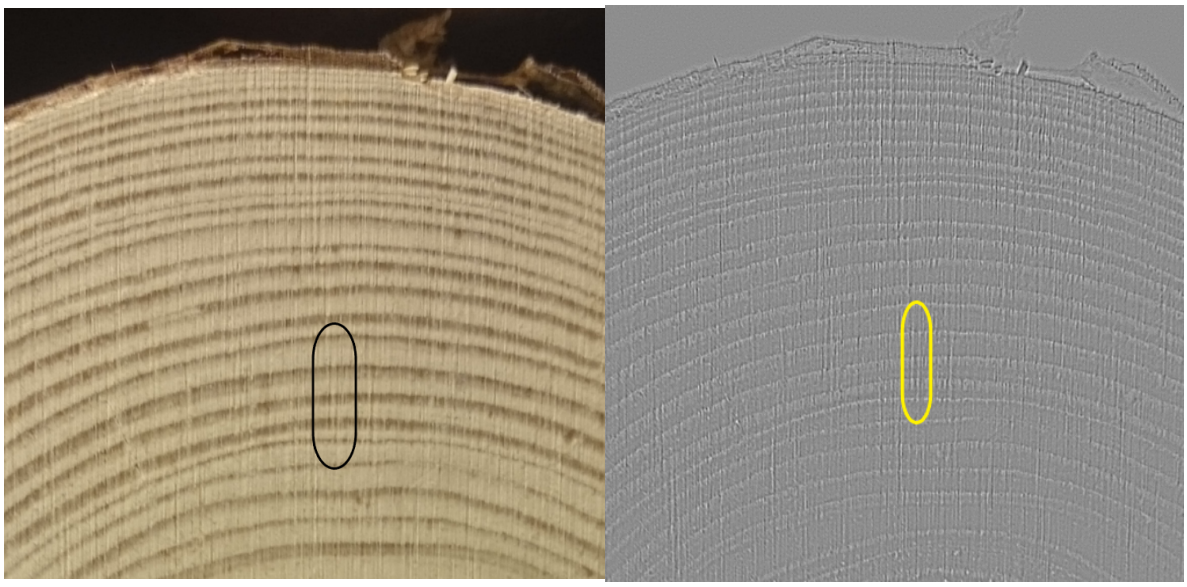
1.3.5 Step 5: Counting the rings in the horizonatal direction

The number of rings in the horizontal direction is counted by traversing from the left end of the image to only the center of the wood profile, in order to reduce the possible additional errors due to traversing the enitre image width.
The laplace filtered image shown in fig(7) is read as a 2D matrix with respective intensities at each pixel location. As visible in the figure, the edges of the image after laplace filtering are displayed with a higher intensity than the background.
Hence the criteria for counting the rings is to check for pixels with intensity higher than a specific threshold intensity. A threshold pixel intensity of **158** (0- black and 255-white) seems to be suitable to differentaite the edges from the background, and in this case it was determined by trial and error.
The 1020 row of the image matrix is checked for occurences of such high pixel intensities (>158) to count as a ring. The code for the same is given below:

```
imx = cv2.imread('lap_fil.bmp',0)
c=0
for i in range (900):
        if (imx[1020,i]>158): #checking for high intensities
                c=c+1                #variable for ring count
print('The number of rings in horizontal direction from the
   left end to the centre of the wood profile is',c)
```

1.3.6 Step 6: Counting the rings in the vertical direction

In counting the ring in vertical direction the image is traversed from the top to the center of the wood profile, in order to reduce the possible additional errors due to traversing the enitre image hieght.
The threshold pixel intensity to count the rings is **158**, the same as in section 1.3.5 while counting the rings in horizontal direction.
However , there is an addtional error source in the image particularly in counting in vertical direction:
Error source: On observing the image closely we see vertically oriented miniature edges in the wood profile which are a result of the wood cutting along that direction. the same is shown and highlighted in figure(9a). The edges seem to be identified by the laplacian filter even after using 'Lapcian of Gaussian' as shown and highlighted in figure(9b)



(a) Vertical eges in original image (b) Vertical edges in laplace image

Figure 9: Miniature vertical edges :error source

The existence of this error source is confirmed when we look at the pixel locations (variable list 'loc') with high intensities, dtermined by the following code for the threshold 158:

```
loc=[]
for i in range (1020):
        if (imx[i,900]>158):
                loc.append(i)
```

The list 'loc' from the code snippet holds the following high intensity pixel locations:

```
[55, 63, 142, 143, 144, 146, 157, 158, 165, 166, 180, 198,
   203, 335, 336, 338, 339, 340, 370, 371, 372, 373, 374,
   375, 376, 377, 378, 379, 380, 381, 398, 399, 400, 401,
   402, 403, 404, 405, 406, 407, 422, 423, 424, 425, 431,
   442, 443, 444, 467, 468, 469, 470, 555, 556, 752, 797,
   813, 829, 895, 912, 973, 991, 992, 1000, 1018, 1019]
```

From the values we see the existence of consecutive pixels with high intensities in the vertical direction, for example:locations 375,376..381. These consecutive pixel locations

cannot be counted as consecutive rings but rather have to be accounted for as vertical edges and these vertical edges also intersect with the multiple rings.
Mitigating error source: In order to tackle this problem only the first high intensity pixel of such consecutive pixels is counted as a ring.
And for more than 9 consecutive pixels a new ring count is added, because there is an average space of 9 pixels between each ring and it is highly probable that the vertical edge is intersecting a new ring.
The counting as described above is made possible by using the 'continue' statement within the loop corresponding to the required conditioons. The code is as given below:

```
for i in range(len(loc)):
        if(loc[i]-loc[i-1]<=1):
                r=r+1            #variable to count consective
                    pixels
                if(r%9==0):
                        d=d+1
                else:
                        continue
        d=d+1                # variable for ring count
        r=0
print('The number of rings in the vertical dirtection from
   the top to the centre of the wood profile is',d)
print('-----------------------------------')
```

References

[1] Grayscale - Wikipedia
https://en.wikipedia.org/wiki/Grayscale

[2] Laplace of Gaussian
https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[3] OpenCV Gaussian Blur
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

[4] Opencv Laplacian filter
https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html

Python code for the project:

```
"""
Created on Sun Sep  8 12:18:39 2019

Author: Vivek Chandrasekaran
Title : Scientific Visualization Project 1
Description: Wood_2 image filtering and counting the rings in
    the wood pattern
"""

import cv2
import matplotlib.pyplot as plt
import numpy as np
from copy import deepcopy
import scipy.misc as misc

def Red(image):         # Variables received as parameters
        image[:,:,1] = 0   # Making green content 0
        image[:,:,2] = 0   # Making blue content 0
return image        # Returning the image

# function for creating green content image
def Green(image):
        image[:,:,0] = 0   # Making red content 0
        image[:,:,2] = 0   # Making blue content 0
return image

# function for creating blue content image
def Blue(image):
        image[:,:,0] = 0   # Making red content 0
        image[:,:,1] = 0   # Making green content 0
return image

# function for creating gray image
def Gray(image):
# Luminosity average for getting agray image
        avg = np.dot(image[...,:3], [0.299, 0.587, 0.114])
# Storing the average 2-d in all 3 axes
        image[:,:,0] = avg
        image[:,:,1] = avg
        image[:,:,2] = avg
return image


im = plt.imread('wood_2.bmp')

# Converting to red
```

```
# Creating a copy in another variable without passing the
   pointer
im1 = deepcopy(im)
red = Red(im1)       # Calling the Red function
misc.toimage(red).save('Red.bmp') # Saving red content image
plt.imshow(red)

# Converting to blue
im2 = deepcopy(im)
blue = Blue(im2)     # Calling the Blue function
misc.toimage(blue).save('Blue.bmp') # Saving blue content
   image
plt.figure()         # For creating another figure window
plt.imshow(blue)

im3 = deepcopy(im)
green = Green(im3)     # Calling the Blue function
misc.toimage(green).save('Green.bmp') # Saving green content
   image
plt.figure()         # For creating another figure window
plt.imshow(green)


# Converting to gray
im4 = deepcopy(im)
gray = Gray(im4)     # Calling the Gray function
misc.toimage(gray).save('Gray.bmp') # Saving gray content
   image
plt.figure()
plt.imshow(gray)


#Gaussian filtering to smoothen disturbance
im5=deepcopy(im)
gauss = cv2.GaussianBlur(im5,(3,3),0)
misc.toimage(gauss).save('Gauss.bmp') # Saving gaussian
   filtered content image

#Caonsidering Gaussian blur image to convert to gray scale
   and laplace filter
im = plt.imread('Gauss.bmp')

# Converting to gray
im4 = deepcopy(im)
gray = Gray(im4)     # Calling the Gray function
misc.toimage(gray).save('Gray2.bmp') # Saving gray content
   image
```

```
#laplace filtering
im = plt.imread('Gray2.bmp')
im5 = deepcopy(im)
laplacian = cv2.Laplacian(im5,cv2.CV_64F)
misc.toimage(laplacian).save('lap_fil.bmp') # Saving blue
   content image
#plt.figure()          # For creating another figure window
#plt.imshow(laplacian)

#reading the filtered image to find the size in pixels
imx = cv2.imread('lap_fil.bmp',0)
size = np.ones(2)
size[0] = len(imx);
size[1] = len(imx[0])
print('-----------------------------------')
print('RESULTS')
print('-----------------------------------')
print('Size of the image in pixels is') #Displaying the size
print('Rows = ',size[0])
print('Columns = ',size[1])
print('-----------------------------------')

##Centering
#imy = cv2.imread('wood_2.bmp',0)
#centering = deepcopy(imy)
#centering[1020,:]= 0 #horizontal black crosshair
#centering[:,900] = 0 #vertical black crosshair
#misc.toimage(centering).save('centering.bmp')

#counting the number of rings in horizontal direction
c=0
for i in range (900):
        if (imx[1020,i]>158): #checking for high intensities
                c=c+1                #variable for ring count
print('The number of rings in horizontal direction from the
   left end to the centre of the wood profile is',c)

#counting the number of rings in the vertical direction
#recording location of the high intensities
d=0
loc=[]
for i in range (1020):
        if (imx[i,900]>158):
                loc.append(i)
#print(loc)

#filtering high intensities for redundancies and errors in
```

```
   vertical direction
r=0
for i in range(len(loc)):
        if(loc[i]-loc[i-1]<=1):
                r=r+1           #variable to count consective
                    pixels
                if(r%9==0):
                        d=d+1
                else:
                        continue
        d=d+1               # variable for ring count
        r=0
print('The number of rings in the vertical dirtection from
   the top to the centre of the wood profile is',d)
print('-----------------------------------')

# the average number of rings
avg= (d+c)/2
print('The average number of rings in the wwod profile is:')
print(avg)
print('-----------------------------------')
```