CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

The first step, the users will be asked to upload the USG images obtained from the source. After that, change the images into grayscale and then process it into filtering. The next step is, the images will be returned from the pixel array into JPG. And then calculate the images by MSE to get the PSNR value for more accurate result.

```
1. import cv2
2. import numpy as np
3. import math
4. ColorImg=cv2.imread('/home/piter/Deni/Bimbingan/FotoUSG/
   bayi54.jpg')
5. GrayImg = cv2.cvtColor(ColorImg, cv2.COLOR_BGR2GRAY)
6. cv2.imshow('Gambar Asli', ColorImg)
7. cv2.imshow('Gambar Gray', GrayImg)
8. cv2.imwrite('Grayscale.jpg',GrayImg)
9. cv2.waitKey(0)
10.cv2.destroyAllWindows()
```

Line 1 to 3 is for the import of library opencv, numpy, and math. Line 4 is for take the USG images from the folder, while line 5 is for change it into grayscale. Line 6 to 10 is to show the result of original images and the grayscale one, also to save the grayscale one so that the image will appear when the program is started.

```
11.   for a in range(Tinggi - AreaPix):
12.       for b in range(Lebar - AreaPix):
13.           x = 0
14.           for i in range(Baris, Baris + Kernel):
15.               y = 0
16.               for j in range(Kolom, Kolom + Kernel):
17.                   MaksKernel[x][y] = GrayImg[i,j]
18.                   if x == 1 and y == 1:
19.                       data = {
20.                                  "x": i, "y": j, "value":
GrayImg[i,j]}
21.                   y +=1
22.               x +=1
23.           NilaiMin = np.min(MaksKernel) #Sort Pixel Min
```

```
24.            NilaiMaks = np.max(MaksKernel) #Sort Pixel Max
25.            Hasil = (int(NilaiMin) + int(NilaiMaks))/2
```

Line 11 to 22 is maximum kernel to determine the position of the pixels to be filtered. The writer uses 3x3, 5x5, 7x7, and 9x9 maximum kernel; the larger the kernel, the wider the calculation area. Line 23 to 25 is calculating the midpoint filter algorithm by finding the minimum and maximum values in the kernel, and then it is divided by two.

```
26.      from copy import copy, deepcopy
27.      CopyImg = deepcopy(GrayImg)
28.      for i in index:
29.          temp =  int(i['value'])
30.          CopyImg[i['x']][i['y']] = temp
31.          print(CopyImg)
32.      cv2.imwrite('/home/piter/Deni/Bimbingan/MidpointK3/
  bayi1.jpg',CopyImg)
33.      cv2.imshow('Gambar Gray', GrayImg)
34.      cv2.imshow('Midpoint.jpg',CopyImg)
35.      cv2.waitKey(0)
36.      cv2.destroyAllWindows()
```

Line 26 is for deep-copying. Line 27 is for initializing the USG grayscale images. Line 28 to 31 is looping process for creating the images that have been filtered. Line 32 is storing the filtered images in JPG format. Line 33 to 36 is displaying the grayscale images and displaying the filtered images. The images will appear once the program is started.

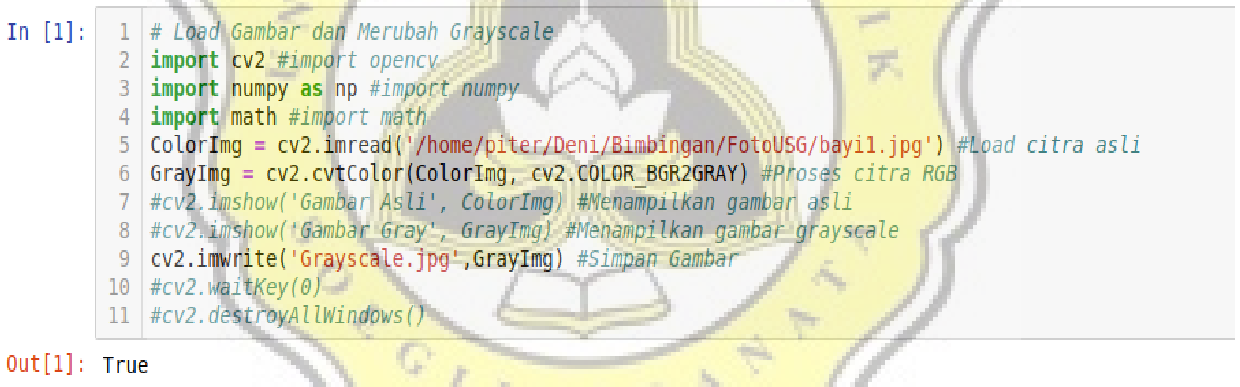
```
37.      for a in range(Tinggi - AreaPix):
38.          for b in range(Lebar - AreaPix):
39.              x = 0
40.              Hasil = 0
41.              for i in range(Baris, Baris + Kernel):
42.                  y = 0
43.                  for j in range(Kolom, Kolom + Kernel):
44.                      MaksKernel[x][y] = GrayImg[i,j]
45.                      Hasil += (1.0/MaksKernel[x][y])
46.                      if x == 1 and y == 1:
47.                          data = {
48.                                       "x": i, "y": j, "value":
  GrayImg[i,j]}
49.                      y +=1
50.                  x +=1
51.
52.              NewPixel[1][1] = math.ceil(JmlhPix/Hasil)
53.              data['value'] = math.ceil(JmlhPix/Hasil)
```

Line 37 to 50 is setting the pixel position that will be filtered by maximum kernel. The writer uses 3x3, 5x5, 7x7, and 9x9 maximum kernel; the larger the kernel, the wider the calculation area. The pixels will be divided by one first in the line 45. Line 52 to 53 is to calculating the harmonic mean filter algorithm, which is the kernel value is divided by the average of pixels.

```
54.     mse = np.square(np.subtract(CopyImg,GrayImg)).mean()
55.     print mse
56.     pnsr = 10 * math.log10(255*255 / mse)
57.     print pnsr
```

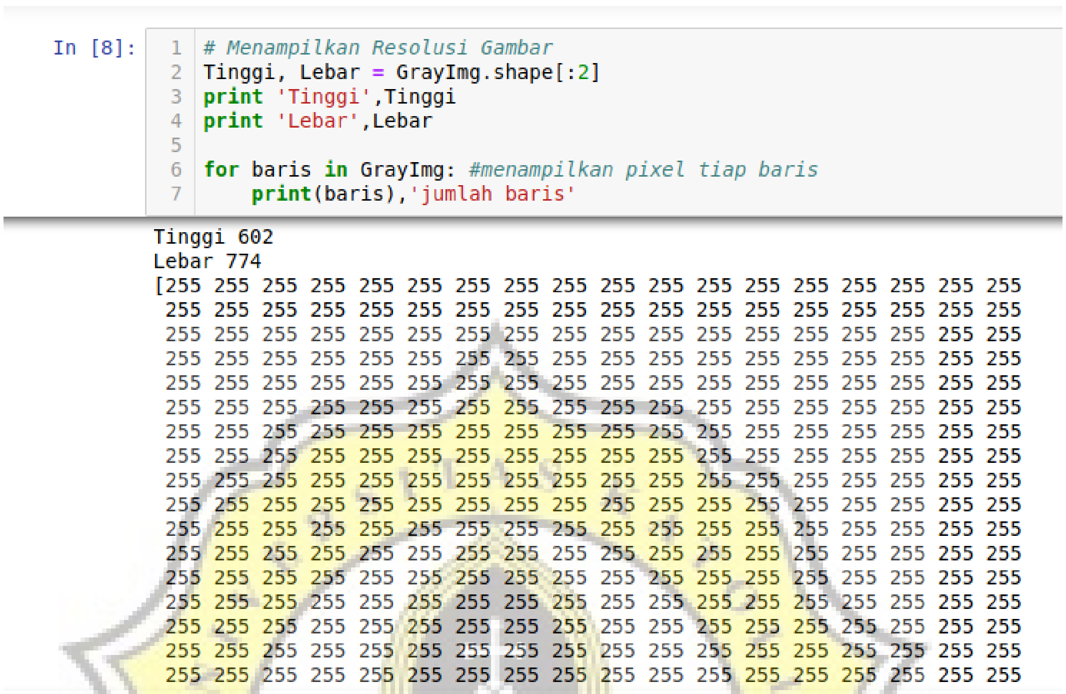
Line 54 to 55 is calculating the average of the filtered images. Line 56 to 57 is calculating the PSNR for the more accurate level of the filtered images.

5.2 Testing



```
In [1]:
# Load Gambar dan Merubah Grayscale
import cv2 #import opencv
import numpy as np #import numpy
import math #import math
ColorImg = cv2.imread('/home/piter/Deni/Bimbingan/FotoUSG/bayi1.jpg') #Load citra asli
GrayImg = cv2.cvtColor(ColorImg, cv2.COLOR_BGR2GRAY) #Proses citra RGB
#cv2.imshow('Gambar Asli', ColorImg) #Menampilkan gambar asli
#cv2.imshow('Gambar Gray', GrayImg) #Menampilkan gambar grayscale
cv2.imwrite('Grayscale.jpg',GrayImg) #Simpan Gambar
#cv2.waitKey(0)
#cv2.destroyAllWindows()

Out[1]: True
```

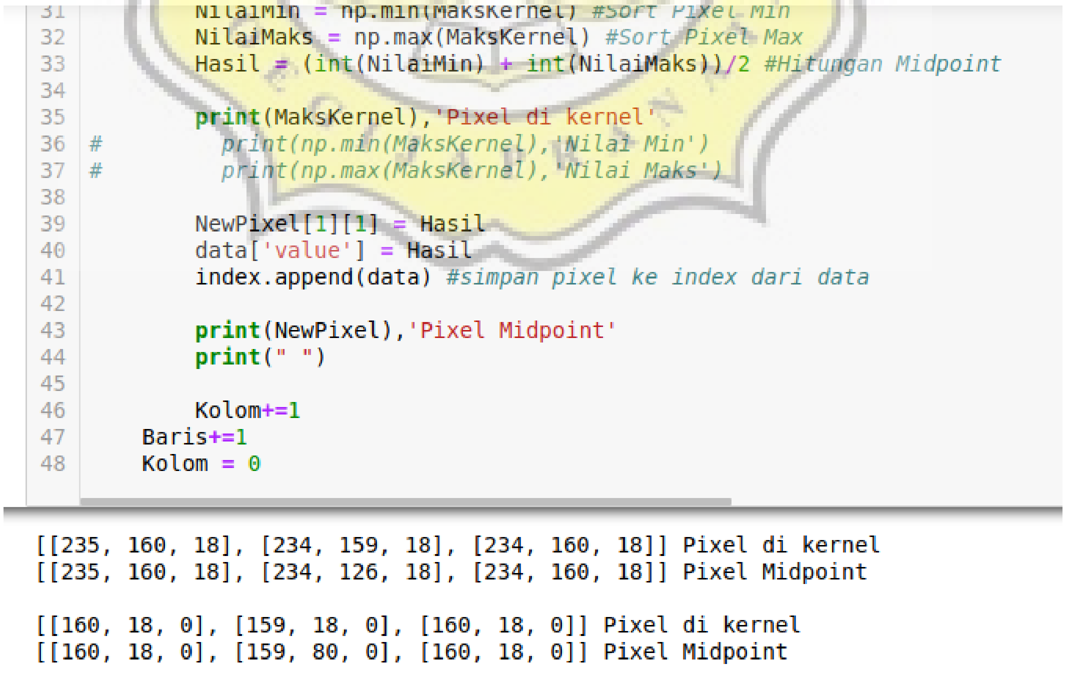
The first test is inserting the USG images file. Second is changing into grayscale to facilitate the filtering process. If “true” word appears in the output, it means the process is success.



```
In [8]:
# Menampilkan Resolusi Gambar
Tinggi, Lebar = GrayImg.shape[:2]
print 'Tinggi',Tinggi
print 'Lebar',Lebar

for baris in GrayImg: #menampilkan pixel tiap baris
    print(baris),'jumlah baris'
```

```
Tinggi 602
Lebar 774
[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
```

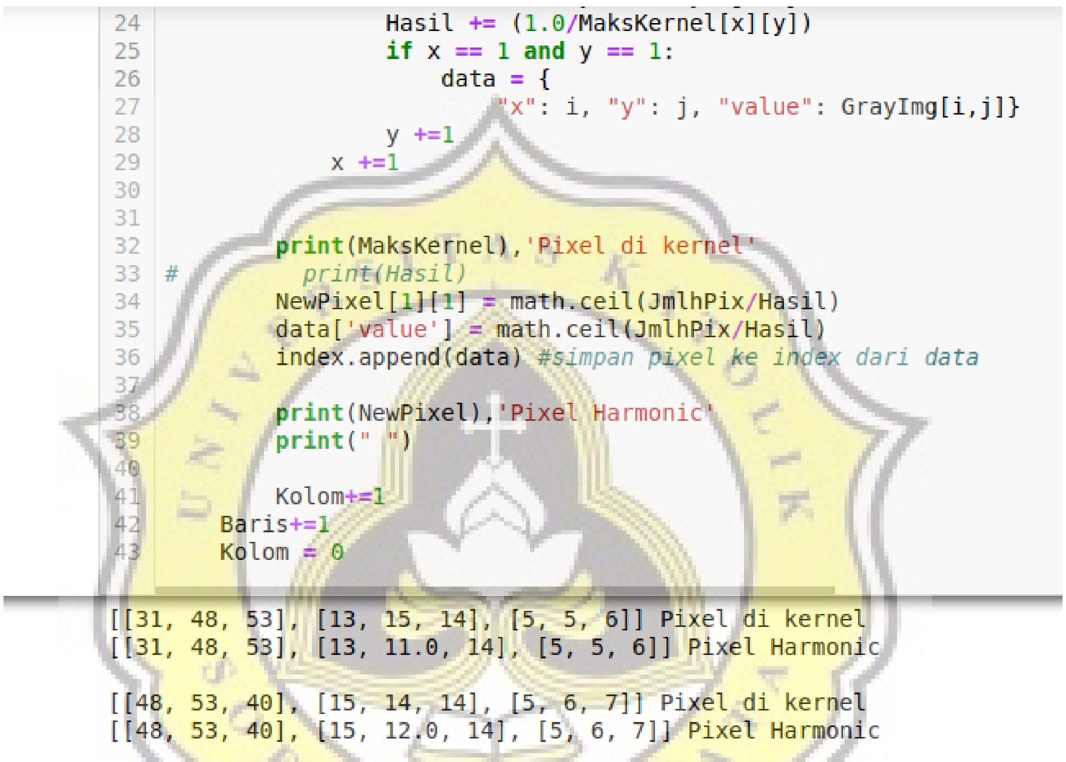
The code above shows the size of USG images resolution that have been in grayscale, and shows the pixel value of the images.



```
            NilaiMin = np.min(MaksKernel) #Sort Pixel Min
            NilaiMaks = np.max(MaksKernel) #Sort Pixel Max
            Hasil = (int(NilaiMin) + int(NilaiMaks))/2 #Hitungan Midpoint

            print(MaksKernel),'Pixel di kernel'
#             print(np.min(MaksKernel),'Nilai Min')
#             print(np.max(MaksKernel),'Nilai Maks')

            NewPixel[1][1] = Hasil
            data['value'] = Hasil
            index.append(data) #simpan pixel ke index dari data

            print(NewPixel),'Pixel Midpoint'
            print(" ")

            Kolom+=1
        Baris+=1
        Kolom = 0
```

```
[[235, 160, 18], [234, 159, 18], [234, 160, 18]] Pixel di kernel
[[235, 160, 18], [234, 126, 18], [234, 160, 18]] Pixel Midpoint

[[160, 18, 0], [159, 18, 0], [160, 18, 0]] Pixel di kernel
[[160, 18, 0], [159, 80, 0], [160, 18, 0]] Pixel Midpoint
```

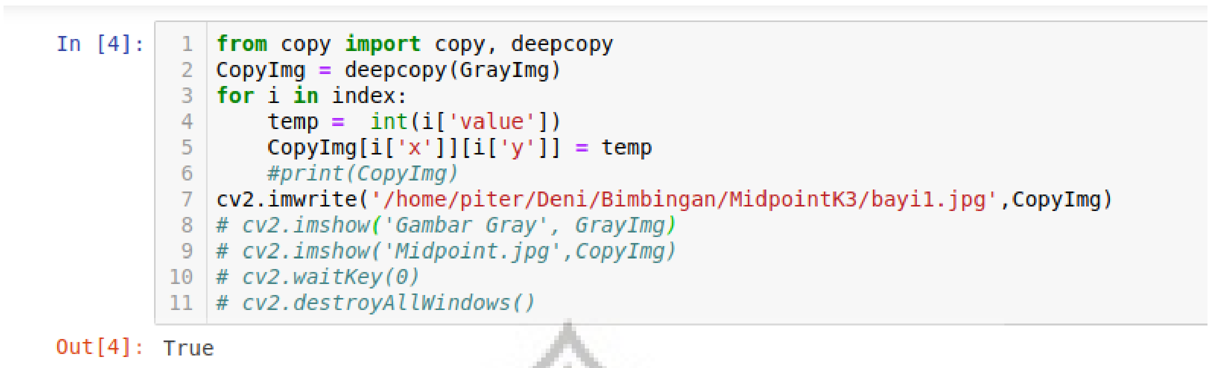
The code above is for the midpoint filter algorithm calculation. Kernel above uses 3x3. If the code is correct, the appeared output is the pixel value which has been already in the 3x3 kernel, and the result is from the midpoint filter calculation.



```
        Hasil += (1.0/MaksKernel[x][y])
        if x == 1 and y == 1:
          data = {
            "x": i, "y": j, "value": GrayImg[i,j]}
        y +=1
      x +=1


    print(MaksKernel),'Pixel di kernel'
#    print(Hasil)
    NewPixel[1][1] = math.ceil(JmlhPix/Hasil)
    data['value'] = math.ceil(JmlhPix/Hasil)
    index.append(data) #simpan pixel ke index dari data

    print(NewPixel),'Pixel Harmonic'
    print(" ")

    Kolom+=1
  Baris+=1
  Kolom = 0
```

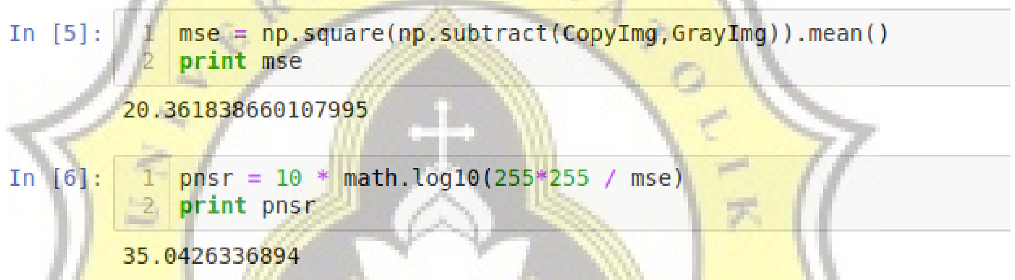
```
[[31, 48, 53], [13, 15, 14], [5, 5, 6]] Pixel di kernel
[[31, 48, 53], [13, 11.0, 14], [5, 5, 6]] Pixel Harmonic

[[48, 53, 40], [15, 14, 14], [5, 6, 7]] Pixel di kernel
[[48, 53, 40], [15, 12.0, 14], [5, 6, 7]] Pixel Harmonic
```

The code above is for harmonic mean filter algorithm calculation. The kernel from the picture above uses 3x3 kernel. If the code is correct, then the appeared output is the pixel value which has been already in the 3x3 kernel, and the result is from the harmonic mean filter calculation.



```
In [4]:  from copy import copy, deepcopy
         CopyImg = deepcopy(GrayImg)
         for i in index:
             temp =  int(i['value'])
             CopyImg[i['x']][i['y']] = temp
             #print(CopyImg)
         cv2.imwrite('/home/piter/Deni/Bimbingan/MidpointK3/bayi1.jpg',CopyImg)
         # cv2.imshow('Gambar Gray', GrayImg)
         # cv2.imshow('Midpoint.jpg',CopyImg)
         # cv2.waitKey(0)
         # cv2.destroyAllWindows()

Out[4]:  True
```

The code above is the process of making pixel that has been filtered and stored in the index, to be used as an image of the JPG format.



```
In [5]:  mse = np.square(np.subtract(CopyImg,GrayImg)).mean()
         print mse

         20.361838660107995

In [6]:  pnsr = 10 * math.log10(255*255 / mse)
         print pnsr

         35.0426336894
```

The code above is for the process of calculating the MSE process to calculate the average images that have been filtered. The PSNR functions is to measure the accuracy level of filtering of the images.