

CS282 PA1 Report

OpenCV Exercises

Image Enhancement

Marwin B. Alejo (2020-20221)

Date of Submission: Mar 07, 2022
Due Date: Mar 15, 2022

1. INTRODUCTION

Image Enhancement is a subfield of Computer Vision that focuses on the practical usage of algorithms to improve the quality of the information in an image format before processing. Common of these image enhancement algorithms are performed either in the spatial or frequency space. By extension, practitioners often use the OpenCV library in implementing these algorithms onto several applications. Similarly, this activity focuses on the utilization of OpenCV for simple to intermediate Image Enhancement tasks and on understanding its fundamental concepts. Furthermore, this activity accomplished the following:

1. Write an OpenCV program that perform the following computations: a. Select a negative floating-point number. Take its absolute value, round it, and then take its ceiling and floor. Print the results to screen; b. Generate a 3x4 matrix whose elements are random integers. Print the matrix; c. Declare matrix variables A,B and integer variable c within OpenCV. (see activity sheet for matrix info); d. Create a 2D matrix with three channels of type byte and size 100-by-100 and set all values to zero. Draw a red rectangle between with corners (30, 10) and (60, 40). Display this image; e. Copy the red channel in (d) above into another matrix for display as grayscale image.
2. Reads and displays a video and is controlled by a slider and button. The slider controls the position within the video from star to end in 10 increments and the button controls pause/unpause. Label them both. The program will prompt the user to type in the filename of the video.
3. Write code that reads in and displays an image. When the user clicks on the image, display the mouse pointer coordinates and the corresponding pixel (BGR) values and write those values as text to the screen at the mouse location.
4. Write code that continuously reads frames from webcam, turns the result to grayscale, and performs Canny edge detection on the image. Display all three stages of the processing as three different images in one window.
5. Write code that performs image enhancement on (a) dental.gif, (b) cells27.jpg, (c) butterfly.gif, and (d) momandkids.jpg.

6. Write code that does “Unsharp Masking” on building.tif.
7. Verify the properties of Fourier spectrum on camera-man images.

2. MATERIALS

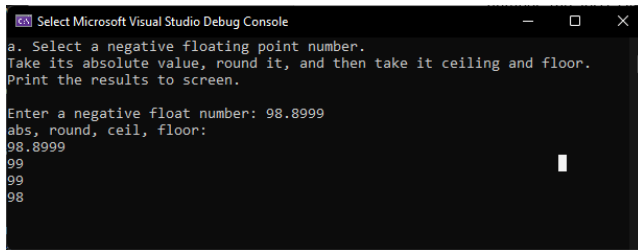
This work implemented the OpenCV (version 4.5.5) on both local Windows 11 machine and Google Colab through either C++ (MSVC++ on VS2022) or Python programming language.

3. BRIEF REPORT

3.1 Matrix Operations through OpenCV

Since OpenCV matrix operations are limited only on C++ environment, this work uses Microsoft Visual Studio with MSVC++ on a local machine with Windows 11.

1. To accomplish item 1.a., the standard `cin()` was used to ask the user to enter a negative float number, the `abs()`, `cvRound()`, `cvCeil()`, and `cvFloor()` functions of OpenCV was used to generate the absolute, round, ceil, and floor values of the entered negative float number. The (1) `abs()` function of OpenCV returns the positive equivalent of any constant or variable, the (2) `cvRound()` function of OpenCV returns a rounded value of the constant or variable to limit decimal places, the (3) `cvCeil()` function of the OpenCV returns a rounded decimal value to higher whole digit number, and the (4) `cvFloor()` function returns a rounded decimal value to lower digit number. These OpenCV function descriptions suffices the output of the written code of this section. Figure 1 shows this sample output.
2. To accomplish item 1.b., this paper uses the same setup in 1.a. This work uses the `Mat()` function of OpenCV with input parameters of (4, 3, CV_8UC1) to create a 4x3 matrix whose elements are one channel 8-bit integer. To generate random numbers as element of the matrix, this work uses the `randu()` function. Figure 2 shows the sample output for this section.
3. To accomplish the several matrix computations as required in section c, this work uses the `Mat()` OpenCV function to create the required matrices, `multiply()`, `add()`, and `PCA()` for processing. Figure 3 shows the sample output of the code written for section c.



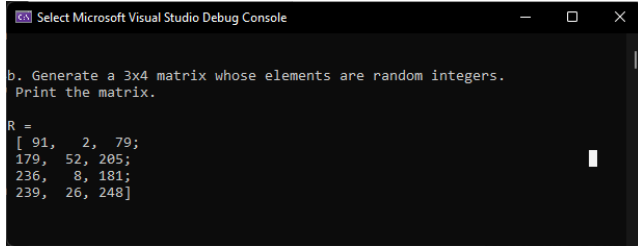
```

Select Microsoft Visual Studio Debug Console
a. Select a negative floating point number.
Take its absolute value, round it, and then take it ceiling and floor.
Print the results to screen.

Enter a negative float number: 98.8999
abs, round, ceil, floor:
98.8999
99
99
98

```

Figure 1: Output of Item 1.a.



```

Select Microsoft Visual Studio Debug Console
b. Generate a 3x4 matrix whose elements are random integers.
Print the matrix.

R =
[ 91, 2, 79;
179, 52, 205;
236, 8, 181;
239, 26, 248]

```

Figure 2: Output of Item 1.b.

- To accomplish the tasks in section d and e, this work uses the `Mat()`, `Point()`, and `Rectangle()` OpenCV functions to draw the required shaped on screen, given the point coordinates. Figures 4 and 5 show the sample output for this section. Furthermore, the code for this item is in PA01.1 folder – which contain C++ source files.

3.2 OpenCV Video Player

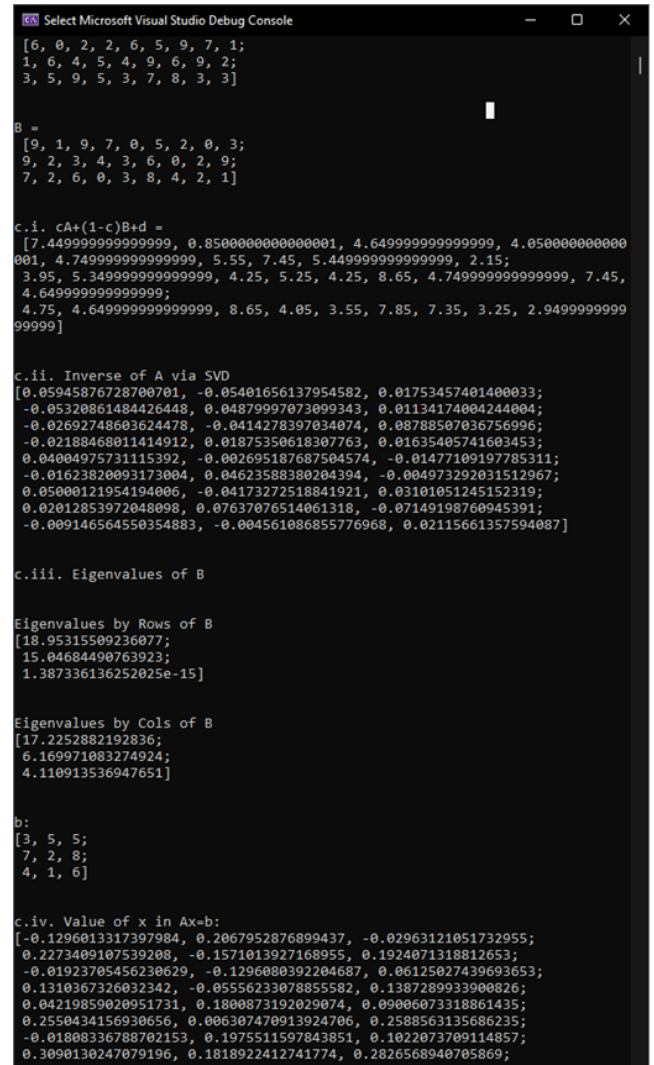
To accomplish the task 2 of this activity, this work used the OpenCV functions of its `imageproc`, `highgui`, and `videoio` modules. This work uses if-else to implement the slider with 10 increments on the “UP naming Mahal” video of “UP Engineering Choir” from their YouTube channel. Moreover, this work uses the keyboard button “p” to implement the pause/play function requirement of item 2. Figure 6 shows the sample screenshot of the developed simple video player. The C++ source file of this item is in PA01.2 folder.

3.3 Mouse Location and Pixel Values

For the item 3 of this activity, this work implemented the OpenCV with Python programming on a Window 11 local machine. This work uses the `putText()` and `setMouseCallback()` OpenCV functions to accomplish this task on ‘regalia.png’. Figure 7 shows a screenshot of the output of this task. The complete code of this task is in PA01.3.py.

3.4 Live Feed Camera Filter

For the item 4 of this activity, this work uses the same set-up as in item 3. This work also uses the `VideoCapture()`, `cvtColor()`, `Canny()`, and `hconcat/vconcat()` functions of OpenCV. The output of this task is a windows with three camera feeds whose output are (1) no filter, (2) grayscale, and (30) and canny edge as shown in figure 8. The complete code of this task is in PA01.4.py.



```

Select Microsoft Visual Studio Debug Console

[6, 0, 2, 2, 6, 5, 9, 7, 1;
1, 6, 4, 5, 4, 9, 6, 9, 2;
3, 5, 9, 5, 3, 7, 8, 3, 3]

B =
[9, 1, 9, 7, 0, 5, 2, 0, 3;
9, 2, 3, 4, 3, 6, 0, 2, 9;
7, 2, 6, 0, 3, 8, 4, 2, 1]

c.i. cA+(1-c)B+d =
[7.449999999999999, 0.8500000000000001, 4.649999999999999, 4.050000000000000,
0.01, 4.749999999999999, 5.55, 7.45, 5.449999999999999, 2.15;
3.95, 5.349999999999999, 4.25, 5.25, 4.25, 8.65, 4.749999999999999, 7.45,
4.649999999999999;
4.75, 4.649999999999999, 8.65, 4.05, 3.55, 7.85, 7.35, 3.25, 2.949999999999999]

c.ii. Inverse of A via SVD
[0.05945876728700701, -0.05401656137954582, 0.01753457401400033;
-0.05320861484426448, 0.04879997073099343, 0.01134174004244004;
-0.02692748603624478, -0.0414278397034074, 0.08788507036756996;
-0.02188468011414912, 0.01875350618307763, 0.01635405741603453;
0.04004975731115392, -0.002695187687504574, -0.01477109197785311;
-0.01623820093173004, 0.04623588380204394, -0.004973292031512967;
0.05000121954194006, -0.04173272518841921, 0.03101051245152319;
0.02012853972048098, 0.07637076514061318, -0.07149198760945391;
-0.0091465456550354883, -0.004561086855776968, 0.02115661357594087]

c.iii. Eigenvalues of B

Eigenvalues by Rows of B
[18.95315500236077;
15.04684490763923;
1.387336136252025e-15]

Eigenvalues by Cols of B
[17.2252882192836;
6.169971083274924;
4.110913536947651]

b:
[3, 5, 5;
7, 2, 8;
4, 1, 6]

c.iv. Value of x in Ax=b:
[-0.1296013317397984, 0.2067952876899437, -0.02963121051732955;
0.2273409107539208, -0.1571013927168955, 0.1924071318812653;
-0.01923705456230629, -0.1296080392204687, 0.06125027439693653;
0.1310367326032342, -0.05556233078855582, 0.1387289933900826;
0.04219859020951731, 0.1800873192029074, 0.09006073318861435;
0.2550434156930656, 0.006307470913924706, 0.2588563135686235;
-0.01808336788702153, 0.1975511597843851, 0.1022073709114857;
0.3090130247079196, 0.1818922412741774, 0.2826568940705869;

```

Figure 3: Output of Item 1.c.

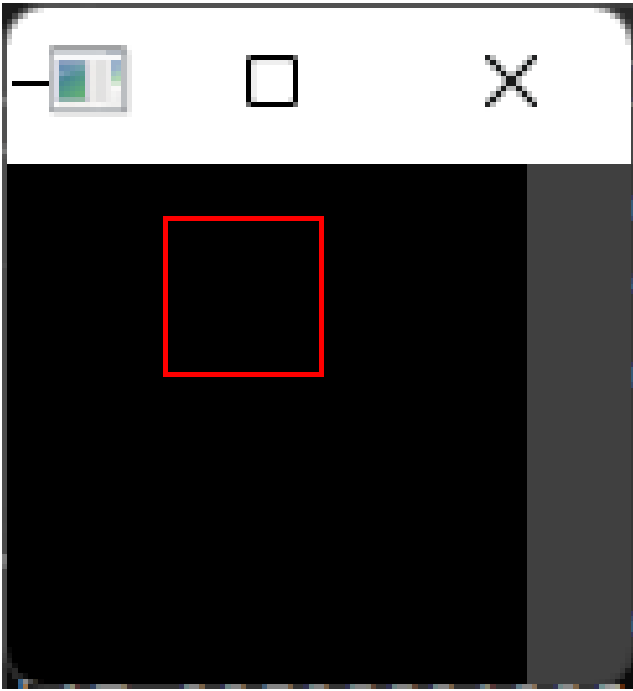


Figure 4: Output of Item 1.d.

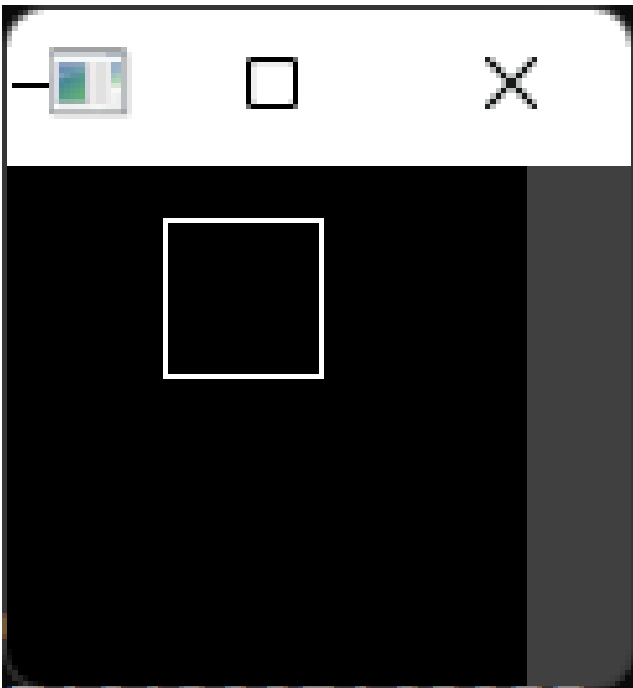


Figure 5: Output of Item 1.e.

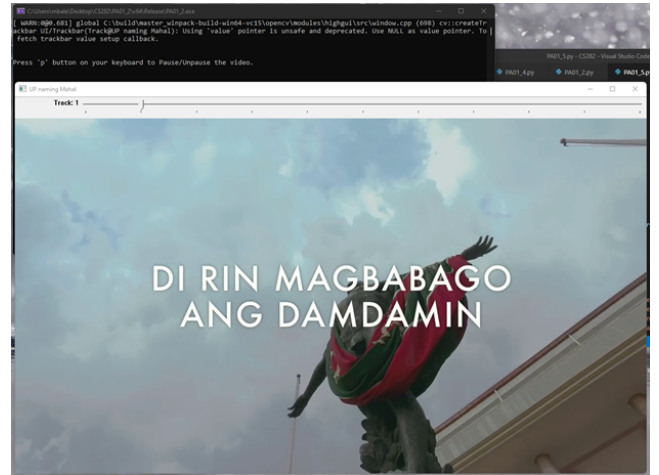


Figure 6: Output of Item 2.



Figure 7: Output of Item 3.

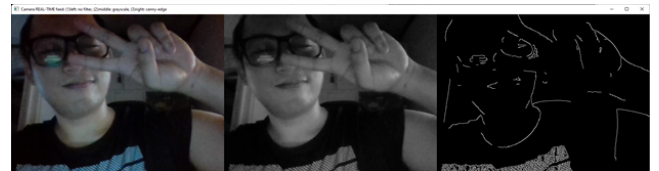


Figure 8: Output of Item 4.

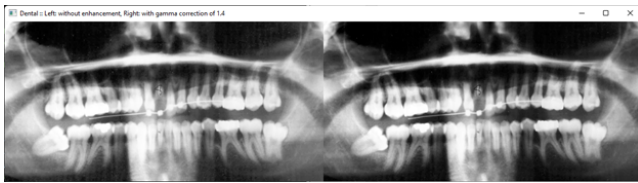


Figure 9: Output of Item 5.a.

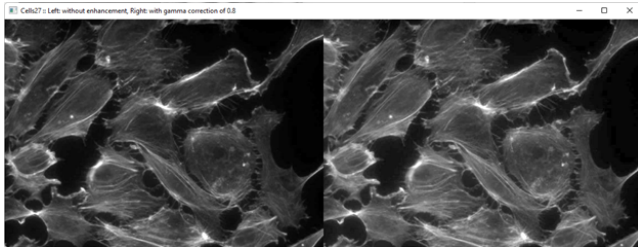


Figure 10: Output of Item 5.b.

3.5 Image Enhancement on Images

For the fifth item of this activity, several image enhancement algorithms were used to accomplish this task while considering the same programming set-up as used in items 3 and 4. Since (a) dental.gif and (b) cells27.jpg are (a) brighter and (b) darker, “gamma correction with (a) 1.4 and (b) 0.8 values” were used to correct the pixel values of each image and increase the required information of both images as they seemed to be. For the (c) butterfly.gif image, this work uses “histogram equalization” to balance the color statistics of the image given that it is original darker than it seemed to be. For the (d) momandkids.jpg image that is affected by the “pepper and salt noise”, this work applied the “median filtering”. Figures 9-12 show the output of these items. The complete code of this task is in PA01_5.py.

3.6 Unsharp Masking

For the sixth item of this activity, this work used the OpenCV functions with Python programming language on Google Colab. To accomplish this task, this work used the `median_filter()` and `Laplacian()` filters. Figure 13 show the

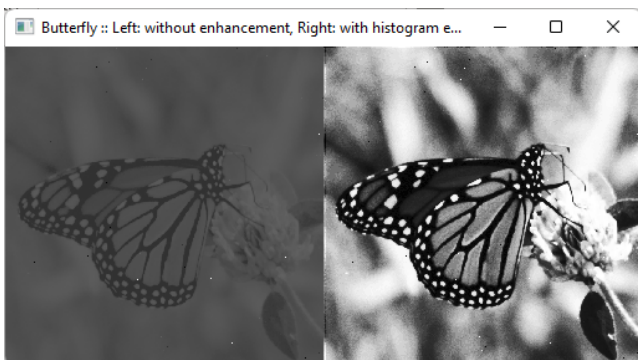


Figure 11: Output of Item 5.c.



Figure 12: Output of Item 5.d.

sample output of the code developed for this item. Notable with the output image is its crispiness over the original image. These artefacts are already in the original however are not highlighted as texture while the output image provides a “texture” on it by adding the masked outline of the grayscale image. The complete code of this task is in PA01_5.ipynb.

3.7 Fourier Transform Translation and Rotation Properties

Similar to item 6, this work accomplished item 7 using OpenCV and Numpy-FFT functions with Python programming language on Google Colab to verify the rotation properties of Fourier Transform using the provided cameraman images. Sample outputs of this item are portrayed by figures 14-17 with code snippet 7 shows the used code for this item.

By inspection of these outputs, all these “cameraman” images possess the same magnitude despite their orientation which is the opposite of their angle values that goes along with their orientation. These simple observation prove that the translation and rotation properties of Fourier transform, as discussed in the activity sheet, are correct.

4. GENERALIZATION

Overall, this report shows the output of each performed item of the activity and OpenCV programming with C++ and Python on both local machine and the cloud. For item 1, matrix computations were demonstrated through OpenCV functions on C++ alone. For item 2, development of a simple video player using C++ and OpenCV was performed. For item 3, putting the actual coordinates and RGB values per pixel of an image was explored. For item 4 (the most fun one), development of a real-time frame capturer with filter was exercised using Python and OpenCV functions. For item 5, appropriate image enhancement computations were performed on provided images to correct their values as they seemed to be. For item 6, unsharp masking was performed on a TIFF file using Python, Scikit-Image, and OpenCV. For item 7, verification of Fourier Transform Rotation and Translation Properties were confirmed with the analysis of the magnitude and angle vectors of “cameraman” images. With these actions, the activity is said to be accomplished.

Before Unsharp Masking



After Unsharp Masking



Figure 13: Output of Item 6.

Camerman 1:

```
Magnitude: [[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
...
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]]
```

```
Angle: [[-3.14159265 2.10890294 -1.93857352 ... 0.43951421 1.93857352
-2.10890294]
[ 1.66974353 -2.45066087 -0.90769046 ... 1.68056331 2.95759358
-1.50915254]
[-2.82320852 -2.65104844 0.31868674 ... -1.00905838 2.86002229
0.09759344]
...
[ 2.1730782 -1.93333526 0.00557562 ... -0.43184036 2.94591224
0.25074855]
[ 2.82320852 -0.09759344 -2.86002229 ... -2.98133123 -0.31868674
2.65104844]
[-1.66974353 1.50915254 -2.95759358 ... -0.10524522 0.90769046
2.45066087]]
```

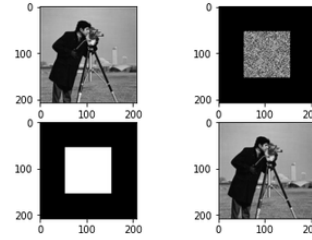


Figure 14: Output of Item 7-cameraman1.

Camerman 2:

```
Magnitude: [[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
...
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]]
```

```
Angle: [[-1.53039974e-14 -6.70292437e-01 -8.91592387e-01 ... -2.52982321e+00
8.91592387e-01 6.70292437e-01]
[-1.23928319e+00 -2.31163417e+00 2.05556033e+00 ... 1.43263299e+00
3.05885214e-01 -1.02430656e+00]
[-2.98196853e+00 4.76566424e-01 4.75107366e-01 ... 2.07519415e+00
3.07319290e+00 3.04907472e+00]
...
[-8.54076839e-01 -1.80612023e+00 -2.69971706e+00 ... -5.42770553e-01
-1.84776919e-01 1.22135909e-01]
[ 2.98196853e+00 -3.04907472e+00 -3.07319290e+00 ... 4.51464709e-02
-4.75107366e-01 -4.76566424e-01]
[ 1.23928319e+00 1.02430656e+00 -3.05885214e-01 ... -1.03809647e-01
-2.05556033e+00 2.31163417e+00]]
```

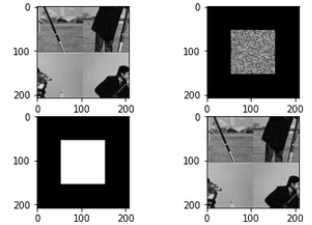


Figure 15: Output of Item 7-cameraman2.

Camerman 3:

```

Magnitude: [[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
...
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]]

Angle: [[ 3.14159265 -1.56000132  0.52598897 ... -1.41122684 -0.52598897
 1.56000132]
[ 0.14383615  0.8592954  1.84599921 ... -2.10072593 -3.06614493
-1.57726339]
[ 1.87294275 -2.34209823 -0.12526724 ... -0.29655969  0.64094524
 1.02626127]
...
[ 0.80066852  1.64701757  2.90380135 ... -2.23979145 -1.07576492
-0.20770137]
[-1.87294275 -1.02626127 -0.64094524 ... -1.11826942  0.12526724
 2.34209823]
[-0.14383615  1.57726339  3.06614493 ... -1.49354712 -1.84599921
-0.8592954 ]]
```

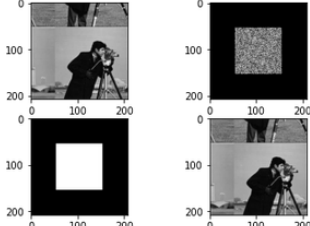


Figure 16: Output of Item 7-cameraman3.

Camerman 4:

```

Magnitude: [[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
...
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]
[0. 0. 0. ... 0. 0. 0.]]

Angle: [[ 4.14483263e-16 -1.34678139e+00  6.11151096e-01 ... -1.00547948e+00
-6.11151096e-01  1.34678139e+00]
[ 4.44381727e-01  2.11609715e+00  2.98170947e+00 ... -3.05616998e+00
-6.90132964e-01 -8.69083781e-01]
[-1.30970272e+00 -9.69506808e-02 -1.28400065e+00 ... -1.05533400e-01
 2.77720689e+00 -2.25142843e+00]
...
[ 2.39196704e+00 -3.02498729e+00  2.50185039e-01 ... -9.02049273e-01
-1.92769973e+00  1.71862321e+00]
[ 1.30970272e+00  2.25142843e+00 -2.77720689e+00 ...  2.91609276e+00
 1.28400065e+00  9.69506808e-02]
[-4.44381727e-01  8.69083781e-01  6.90132964e-01 ...  9.83490114e-01
-2.98170947e+00 -2.11609715e+00]]
```

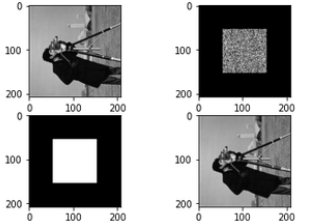


Figure 17: Output of Item 7-cameraman4.