Reflection A6

D V Anantha Padmanabh
ME23B012

May 2024

Task 1

Treasure Hunt II

1.1 Running the script

When the script `script.sh.x` is run, the repository is initialised. A file `question_4.cpp` is made. The file initially contained:

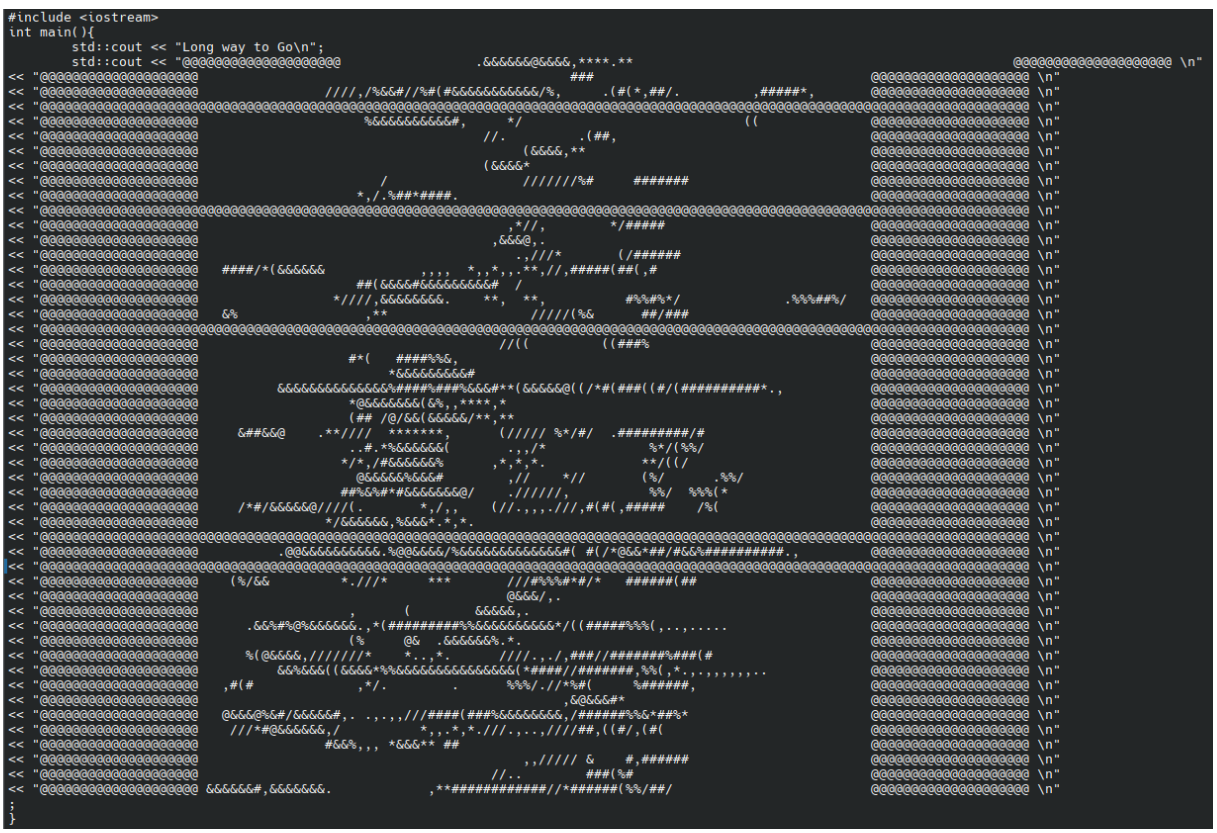
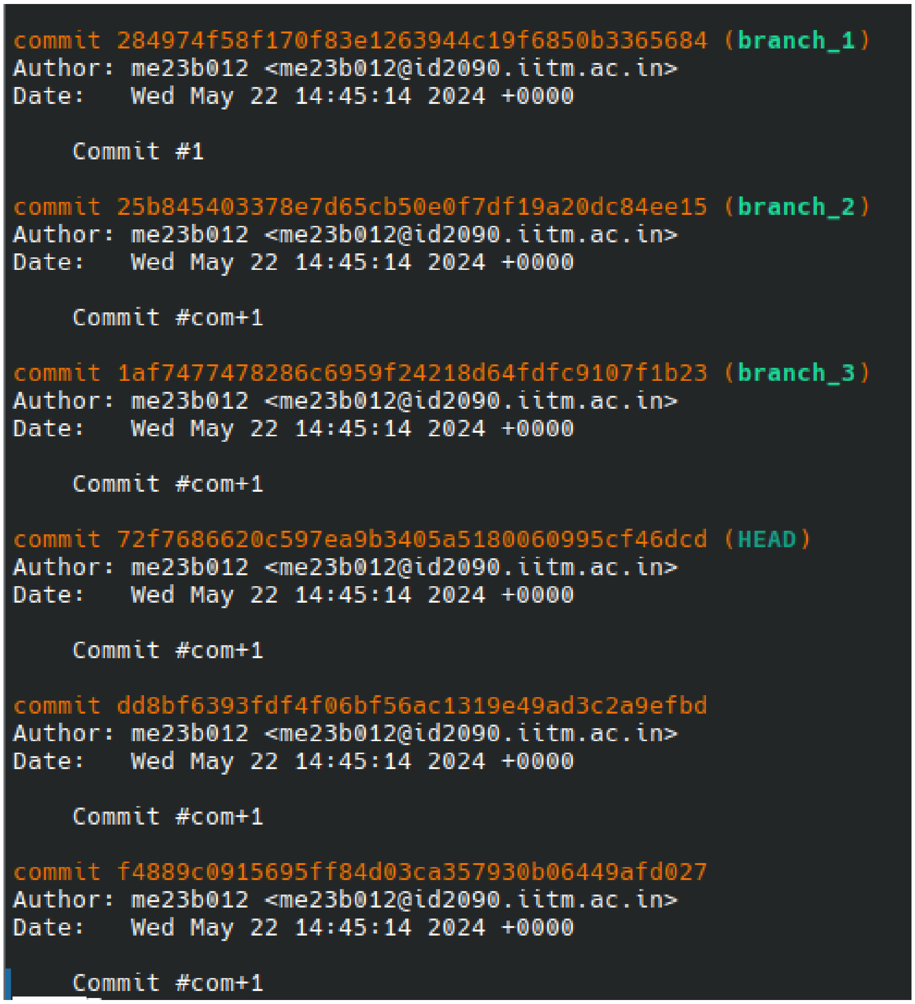


Figure 1.1: question_4.cpp initially

1.2 Finding the commit

We can use `git log --all` to see all commits as shown:



```
commit 284974f58f170f83e1263944c19f6850b3365684 (branch_1)
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #1

commit 25b845403378e7d65cb50e0f7df19a20dc84ee15 (branch_2)
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #com+1

commit 1af7477478286c6959f24218d64fdfc9107f1b23 (branch_3)
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #com+1

commit 72f7686620c597ea9b3405a5180060995cf46dcd (HEAD)
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #com+1

commit dd8bf6393fdf4f06bf56ac1319e49ad3c2a9efbd
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #com+1

commit f4889c0915695ff84d03ca357930b06449afd027
Author: me23b012 <me23b012@id2090.iitm.ac.in>
Date:   Wed May 22 14:45:14 2024 +0000

    Commit #com+1
```

Figure 1.2: Git Log

Then we can checkout to each commit and examine `question_4.cpp` using `git checkout [COMMIT HASH]`

```
→  loss git:(branch_3) ✗ git log --all
→  loss git:(branch_3) ✗ git checkout 284974f58f170f83e1263944c19f6850b3365684
Note: switching to '284974f58f170f83e1263944c19f6850b3365684'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 284974f Commit #1
→  loss git:(284974f) ✗ less question_4.cpp
→  loss git:(284974f) ✗ git log --all
→  loss git:(284974f) ✗ git checkout 25b845403378e7d65cb50e0f7df19a20dc84ee15
Previous HEAD position was 284974f Commit #1
HEAD is now at 25b8454 Commit #com+1
→  loss git:(25b8454) ✗ less question_4.cpp
→  loss git:(25b8454) ✗ git log --all
→  loss git:(25b8454) ✗ git checkout 1af7477478286c6959f24218d64fdfc9107f1b23
Previous HEAD position was 25b8454 Commit #com+1
HEAD is now at 1af7477 Commit #com+1
→  loss git:(1af7477) ✗ less question_4.cpp
→  loss git:(1af7477) ✗ git log --all
→  loss git:(1af7477) ✗ git checkout dd8bf6393fdf4f06bf56ac1319e49ad3c2a9efbd
Previous HEAD position was 1af7477 Commit #com+1
HEAD is now at dd8bf63 Commit #com+1
→  loss git:(dd8bf63) ✗ less question_4.cpp
→  loss git:(dd8bf63) ✗ git log --all
→  loss git:(dd8bf63) ✗ git checkout 72f7686620c597ea9b3405a5180060995cf46dcd
Previous HEAD position was dd8bf63 Commit #com+1
HEAD is now at 72f7686 Commit #com+1
→  loss git:(72f7686) ✗ less question_4.cpp
→  loss git:(72f7686) ✗
```

Figure 1.3: Git Checkout

The correct file was in commit `72f7686620c597ea9b3405a5180060995cf46dcd` therefore we checkout there by doing `git checkout 72f7686620c597ea9b3405a5180060995cf46dcd`.

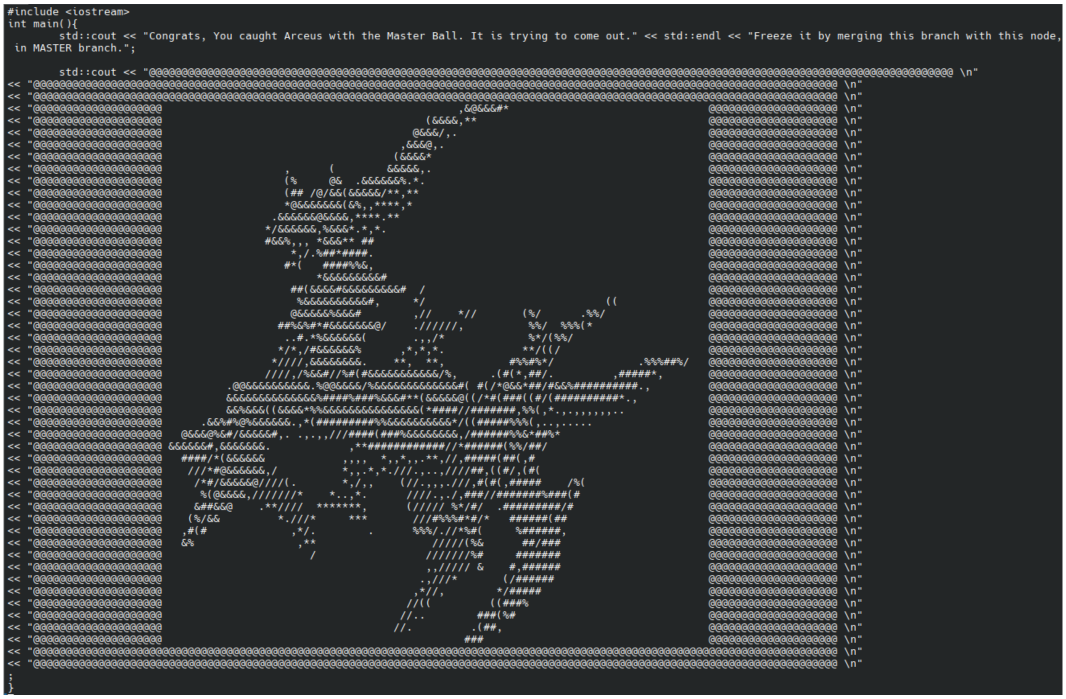


Figure 1.4: Correct question_4.cpp

1.3 Merging to master

By using `git branch`, we see there are just three branches and no master branch:

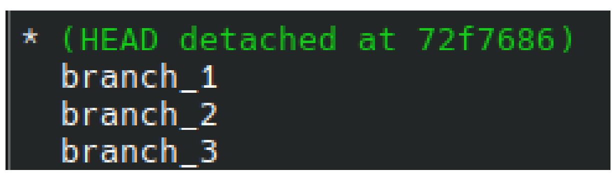


Figure 1.5: Existing Branches

So we add a branch master using `git branch master` then checkout to it by `git checkout master`.

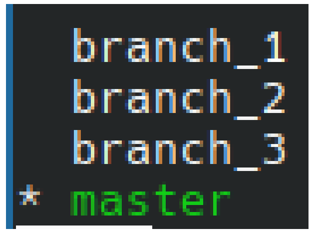


Figure 1.6: Master Branch

Finally we merge the commit to master by doing `git merge 72f7686620c597ea9b3405a5180060995cf46dcd` (make sure you are in master branch).

Task 2

Image Processing

2.1 Introduction

2.1.1 Kernel Operations

Consider a 3×3 kernel which is a matrix A_{ij} that will operate on an image matrix as follows,

For a pixel located at (m, n) the value of the pixel $P_{m,n}$ in the image, the new value is given by the following formula:

$$P_{m,n}^{new} = \sum_{i=0,j=0}^{i=2,j=2} A_{ij} \cdot P_{m-1+i,n-1+j}^{old}$$

This is known as convolution operation.

2.1.2 Edge Detection

The edge detection class of kernels are used for detecting the boundaries within images, they highlight the places where the intensity of the pixels sharply change to do this. Examples of few such kernels are as follows:

- **Sobel-Feldman Operator**: it uses the gradient of pixels to find the sharpest intesnity change

$$\text{Sobel X: } \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{Sobel Y: } \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Laplacian Operator**: it measures the second derivative to detect edges. It is sensitive to noise, therefore the image should be smoothed before applying it. Some commonly used Laplacian kernels are:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \text{OpenCV's cv2.Laplacian() uses: } \begin{bmatrix} 2 & 0 & 2 \\ 0 & -8 & 0 \\ 2 & 0 & 2 \end{bmatrix}$$

2.1.3 Brightness

The brightness of an image is basically the intensity of the . Therefore we can equally scale all the pixels to alter brightness.

We can use identity kernel and scale it to do brightness altering.

$$b \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where b is the brightness adjustment factor. When $b > 1$, it makes the image brighter, and when $b < 1$, it makes the image dimmer.

2.1.4 Blurring

Blurring is used to reduce noise/details in an image, creating a smoother and more uniform appearance. It is often a prerequisite step for edge detection. Blurring can be done by applying a kernel that averages the pixel values in a local neighborhood.

Box Blur

The box blur kernel is a simple and efficient way to blur an image. It assigns equal weight to all pixels in a square neighborhood, effectively averaging their values. The size of the neighborhood determines the amount of blurring. A larger kernel size results in more blurring.

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian Blur

Gaussian blur is a more sophisticated blurring technique that uses a Gaussian function to assign weights to the pixels in a neighborhood. The Gaussian function assigns higher weights to pixels closer to the center of the kernel and lower weights to pixels farther away. This results in a smoother and more natural-looking blur.

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Both box blur and Gaussian blur can be applied iteratively to achieve a stronger blurring effect. However, it is important to note that excessive blurring can lead to loss of important details and information in the image.

2.2 Implementation

2.2.1 Part (a): Kernel Operation

Edge Detection

We combine the *Sobel X* and *Sobel Y* operators by taking the magnitude of the gradient,

```
img = imread('Lena.png');
img = double(img);

sobel_x = [ -1  0  1;
            -2  0  2;
            -1  0  1 ];
sobel_y = [ -1 -2 -1;
             0  0  0;
             1  2  1 ];

Gx = conv2(img, sobel_x, 'same');
Gy = conv2(img, sobel_y, 'same');
G = sqrt(Gx.^2 + Gy.^2);
G = G / max(G(:));

imwrite(G, 'sobel_lena.png');
```

(a) Original (b) Sobel Operation

Figure 2.1: Edge Detection

Brightness

Using the idenitity kernel we can change all the pixels by a fixed factor, here I have given a factor of 0.25 which significantly reduces the brightness(1.5 to increase brightness).

```
img = imread('Lena.png');
img = double(img);

k = [0, 0, 0; 0, 0.25, 0; 0, 0, 0];

bright_img = conv2( img, k, 'same');

bright_img = uint8(bright_img);

imwrite(bright_img, 'dim_lena.png')
```

(a) Dim (b) Original (c) Bright

Figure 2.2: Brightness

2.2.2 Part (b): Noise Reduction

We can reduce the noise using the gaussian blur kernel, it reduces noise by computing a weighted average of each pixel with its neighbors, with the weights determined by a Gaussian function. This blurs the image and smoothenes high frequency noise components while preserving low frequency details like edges.

```
img = imread('Noisy_Lena.png');
img = double(img);

k = [0.0625, 0.125, 0.0625; 0.125, 0.25, 0.125; 0.0625, 0.125, 0.0625];

blur_img = conv2(img, k, 'same');
blur_img = uint8(blur_img);

imwrite(blur_img, 'blur_lena.png'));
```

(a) Original (b) Noise Reduced

Figure 2.3: Noise Reduction

2.2.3 Part (c): Fourier Transform

We do Fast Fourier Transform (FFT) to convert to frequency domain and create a low pass filter so that the high frequency noise is reduced.

```
img2 = imread('Lena.png');
F = fft2(img2);

[M, N] = size(img2);
[u, v] = meshgrid(1:N, 1:M);

D0 = 50;

H = exp(-((u - N/2).^2 + (v - M/2).^2) / (2 * D0^2));
F_filtered = F .* fftshift(H);

img_filtered = ifft2(F_filtered);
img_filtered = real(img_filtered);
```

(a) Original

(b) FFT Noise Reduction

Figure 2.4: Noise Reduction using fourier transform

2.2.4 The Complete Script

The script is named `question_2.sh` in the `assignment_6` directory. It needs the files *Lena.png* and *Noisy_Lena.png* to be present in the same directory.

It creates a directory named `me23b012_output_images` which contains all the images after the image processing.

```
#!/usr/bin/octave

% INFO: Script to perform image processing operations
% Usage: ./question_2.sh

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part (a)

% Load image
img1 = imread('Lena.png');
img1 = double(img1);

% Sobel Operator
sobel_x = [-1, 0, 1;
           -2, 0, 2;
           -1, 0, 1];
sobel_y = [-1, -2, -1;
            0,  0,  0;
            1,  2,  1];

% Apply Sobel Operator to image

Gx = conv2(img1, sobel_x, 'same');
Gy = conv2(img1, sobel_y, 'same');
sobel_result = sqrt(Gx.^2 + Gy.^2);
sobel_img = sobel_result / max(sobel_result(:));
```

```
% Apply brightness filter to image

% Brighten the image
k_bright = [0, 0, 0;
            0, 1.5, 0;
            0, 0, 0];
bright_img = conv2(img1, k_bright, 'same');
bright_img = uint8(bright_img);

% Dim the image

k_dim = [0, 0, 0;
         0, 0.25, 0;
         0, 0, 0];
dim_img = conv2(img1, k_dim, 'same');
dim_img = uint8(dim_img);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part(b)

% Apply blur filter to noisy image to smoothen it

img2 = imread('Noisy_Lena.png');
img2 = double(img2);

% Gaussian Blur

k_gaussian_blur = [0.0625, 0.125, 0.0625;
                   0.125, 0.25, 0.125;
                   0.0625, 0.125, 0.0625];

% Apply Gaussian Blur to image

blur_img = conv2(img2, k_gaussian_blur, 'same');
blur_img = uint8(blur_img);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part(c)

% Doing Fast Fourier Transform (FFT)
F = fft2(img2);

% Create a meshgrid

[M, N] = size(img2);
[u, v] = meshgrid(1:N, 1:M);

% Cutoff frequency

D0 = 50;

% Create a filter
```

```
H = exp(-((u - N/2).^2 + (v - M/2).^2) / (2 * D0^2));
F_filtered = F .* fftshift(H);

% Inverse FFT

img_filtered = ifft2(F_filtered);
img_filtered = real(img_filtered);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Create the output directory

mkdir('me23b012_output_images');

% Save the images to the output directory

imwrite(uint8(img1), 'me23b012_output_images/original_image.png');
imwrite(sobel_img, 'me23b012_output_images/edge_detection.png');
imwrite(bright_img, 'me23b012_output_images/brighter.png');
imwrite(dim_img, 'me23b012_output_images/dimmer.png');
imwrite(uint8(img2), 'me23b012_output_images/noisy_image.png');
imwrite(blur_img, 'me23b012_output_images/blur_noise_reduction.png');
imwrite(uint8(img_filtered), 'me23b012_output_images/fft_noise_reduction.png');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.3 Observation and Conclusion

- Different kernels are used for different image processing operations. These kernels have a mathematical logic to the corresponding operations such as scaling intensity of pixels to adjust brightness.
- The `conv2()` function handles border pixels and does necessary padding.
- Blurring using kernels may not be very effective because cant distinguish between edge details and noise therefore some of the details may be lost while smoothing the image.
- The Fourier transform method on the other hand specifically targets the noise which is generally high frequency and caps it to a good extent. **The Fourier transform method works exceptionally in the case of periodic noise**.
- We see that decreasing the cutoff frequency D_0 results in loss of image details and image is more blurred, while increasing it further accepts in more of the noise and image remains noisy. We can find the sweet spot using a frequency analysis but this implementation takes does not cover and an arbitrary value of $D_0 = 50$ is chosen

(a) $D_0 = 20$ (b) $D_0 = 90$

Figure 2.5