HCMUS - VNUHCM / FIT / Computer Vision & Cognitive Cybernetics Department

Digital image & video processing - LQN

**Student ID:** 21127690

**Student name:** Ngo Nguyen Thanh Thanh

**Report: OPENCV – BASIC**

**I. Evaluation summary:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task** | | | **Requirement Met(%)** | **Notes** |
| **Implementation** | Algorithm to transform color | Linear mapping | 100% |  |
| Non-linear mapping | 100% | Logarithmic & Exponential mapping |
| Probability Density Function-based mapping | 100% |  |
| Algorithm to transform geometry | Pixel co-ordinate transformations | 100% | Affine Transformation  and Bilinear transform: |
| Brightness interpolation | 100% | Nearest-neighborhood interpolation and Linear interpolation |
| Algorithm to smooth the image | Averaging filter | 100% |  |
|  | Gaussian filter | 100% |  |
|  | Median filter | 100% |  |
|  | Algorithm to blur the image | Gaussian Blur | 100% |  |
| **Total:** | |  | 100% |  |

1. **List of features:**

**List of Functions:** a summary of the key functions in the program:

1. **read\_image(file\_path)**: Reads an image from the given file path in grayscale using OpenCV.
2. **linear\_mapping(image, a=1.0, b=0)**: Applies a linear transformation (scaling and shifting) to the image.
3. **log\_mapping(image, c=1.0)**: Applies a logarithmic transformation to compress pixel values.
4. **exp\_mapping(image, c=1.0)**: Applies an exponential transformation to expand pixel values.
5. **histogram\_equalization(image)**: Adjusts image contrast by redistributing pixel intensities.
6. **affine\_transform(image, a, b)**: Applies affine transformation to modify pixel coordinates.
7. **nearest\_neighbor\_resize(image, scale\_x, scale\_y)**: Resizes the image using the nearest neighbor method.
8. **linear\_interpolation\_resize(image, scale\_x, scale\_y)**: Resizes the image using linear interpolation.
9. **bilinear\_transform(image, a, b)**: Applies bilinear transformation with interpolation.
10. **averaging\_filter(image, kernel\_size)**: Applies a box filter to smooth the image.
11. **gaussian\_kernel(size, sigma)**: Generates a Gaussian kernel for filtering.
12. **gaussian\_filter(image, kernel)**: Applies a Gaussian filter to smooth the image.
13. **median\_filter(image, kernel\_size)**: Applies a median filter to remove noise.
14. **gaussian\_blur(image, kernel)**: Applies Gaussian blur to the image.

**- Menu Functions:**

1. **menu()**: Displays the main menu with options for different tasks.
2. **color\_transform\_menu(image)**: Displays the color transformation menu.
3. **geometric\_transform\_menu(image)**: Displays the geometric transformation menu.
4. **smoothing\_menu(image)**: Displays the image smoothing menu.

These functions handle the selection of image processing tasks and apply transformations or filters accordingly.

**The program with proof images:**

**A black screen with white text

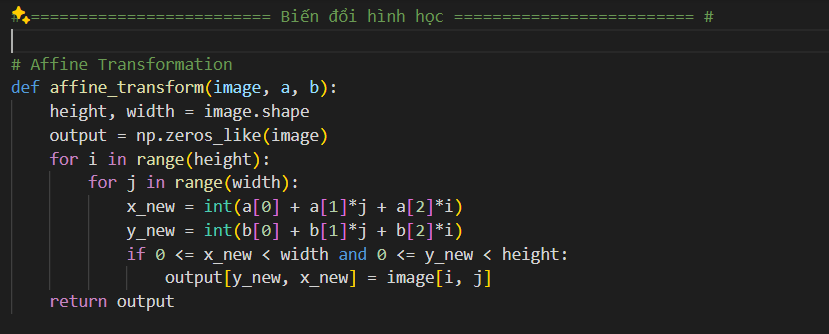
Description automatically generated**

**A computer screen shot of a program

Description automatically generated**

**A screen shot of a computer code

Description automatically generated**

****

**A screen shot of a computer code

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**III. Summarization of the usage**

This program allows users to perform various image processing tasks on the Lenna image, including color transformations, geometric transformations, and image smoothing.

1. **Color Transformations**:
   * Users can apply different transformations to modify the image's pixel values:
     + **Linear Mapping**: Scales and shifts pixel values.
     + **Logarithmic Mapping**: Compresses pixel values to enhance dark areas.
     + **Exponential Mapping**: Expands pixel values to brighten the image.
     + **Histogram Equalization**: Improves contrast by redistributing pixel intensities.
2. **Geometric Transformations**:
   * Users can modify the image's geometry:
     + **Affine Transformation**: Applies linear transformations (rotation, translation, scaling).
     + **Resize (Nearest Neighbor)**: Resizes the image using the nearest neighbor method.
     + **Resize (Linear Interpolation)**: Resizes using linear interpolation for smoother scaling.
     + **Bilinear Transform**: Uses bilinear interpolation to resize the image with more accuracy.
3. **Image Smoothing**:
   * Users can smooth the image using different filters:
     + **Averaging Filter**: Smooths the image using a simple average filter.
     + **Gaussian Filter**: Applies a Gaussian filter to reduce noise and blur the image.
     + **Median Filter**: Uses the median of neighboring pixels to smooth the image.
     + **Gaussian Blur**: Applies a predefined Gaussian blur to soften the image.

The program's user interface allows selecting from a variety of image processing techniques through a menu-driven system. The processed image is displayed after each operation, providing visual feedback for users to explore different transformations and filters.

**Function usage sumarization:**

1. **read\_image(file\_path)**: Reads an image from the given file path in grayscale using OpenCV.
2. **linear\_mapping(image, a=1.0, b=0)**: Applies a linear transformation to the image by scaling and shifting pixel values.
3. **log\_mapping(image, c=1.0)**: Applies a logarithmic transformation to the image to compress the pixel values.
4. **exp\_mapping(image, c=1.0)**: Applies an exponential transformation to the image to expand the pixel values.
5. **histogram\_equalization(image)**: Performs histogram equalization to adjust the image contrast by redistributing pixel intensities.
6. **affine\_transform(image, a, b)**: Applies an affine transformation to the image by modifying the coordinates of the pixels based on a set of parameters.
7. **nearest\_neighbor\_resize(image, scale\_x, scale\_y)**: Resizes the image using the nearest neighbor interpolation method, which selects the closest pixel for resizing.
8. **linear\_interpolation\_resize(image, scale\_x, scale\_y)**: Resizes the image using linear interpolation, where pixel values are computed by averaging the neighboring pixel values.
9. **bilinear\_transform(image, a, b)**: Applies a bilinear transformation to the image by interpolating pixel values using a set of parameters.
10. **averaging\_filter(image, kernel\_size)**: Applies an averaging filter (box filter) to the image to smooth it by averaging the pixels within a given kernel size.
11. **gaussian\_kernel(size, sigma)**: Generates a Gaussian kernel, which is used to apply Gaussian filtering to smooth the image.
12. **gaussian\_filter(image, kernel)**: Applies a Gaussian filter to the image using the specified Gaussian kernel for smoothing.
13. **median\_filter(image, kernel\_size)**: Applies a median filter to the image to remove noise by replacing each pixel with the median of its neighbors.
14. **gaussian\_blur(image, kernel)**: Applies Gaussian blur to the image using a Gaussian kernel to create a blur effect.

**- Menu Functions:**

These are the functions used to display menus and handle user choices for different image processing tasks:

1. **menu()**: Displays the main menu with options to select different image processing tasks. It gives the user four options: Color Transformations, Geometric Transformations, Image Smoothing, and Exit.
2. **color\_transform\_menu(image)**: Displays a sub-menu for color transformations where the user can choose from linear mapping, logarithmic mapping, exponential mapping, and histogram equalization.
3. **geometric\_transform\_menu(image)**: Displays a sub-menu for geometric transformations where the user can choose from affine transformation, resizing using nearest neighbor or linear interpolation, and bilinear transformation.
4. **smoothing\_menu(image)**: Displays a sub-menu for image smoothing where the user can choose from averaging filter, Gaussian filter (custom kernel), median filter, and Gaussian blur (predefined kernel).

These functions handle user input, call the respective transformation or filtering function, and display the results of the image manipulation.

**IV. Implementation**:

**Description of Methods and Pseudo code**

**1. linear\_mapping(image, a=1.0, b=0)**

**Purpose**: Applies a linear transformation to the image by scaling (multiplying by a) and shifting (adding b) pixel values.

**Details**:

* For each pixel, calculate new\_pixel = a \* old\_pixel + b.
* The result is clipped to ensure values remain within the valid pixel range [0, 255].

**Pseudo code:**

For each pixel (i, j) in the image:

output[i, j] = a \* image[i, j] + b

Clip output to the range [0, 255]

Return output

**2. log\_mapping(image, c=1.0)**

**Purpose:** Applies logarithmic mapping to enhance the darker regions of the image.

**Details:**

* For each pixel, calculate new\_pixel = c \* log(1 + old\_pixel).
* The result is clipped to ensure values remain within the valid pixel range [0, 255].

**Pseudo code:**

For each pixel (i, j) in the image:

output[i, j] = c \* log(1 + image[i, j])

Clip output to the range [0, 255]

Return output

**3. exp\_mapping(image, c=1.0)**

**Purpose**: Applies exponential mapping to brighten the image, especially the darker regions.  
**Details**:

* For each pixel, calculate new\_pixel = c \* (exp(old\_pixel / 255.0) - 1) \* 255.
* The result is clipped to ensure values remain within the valid pixel range [0, 255].

**Pseudo code:**

For each pixel (i, j) in the image:

output[i, j] = c \* (exp(image[i, j] / 255.0) - 1) \* 255

Clip output to the range [0, 255]

Return output

**4. histogram\_equalization(image)**

**Purpose**: Equalizes the histogram of the image to improve its contrast.  
**Details**:

* Calculate the histogram of the image.
* Derive the cumulative distribution function (CDF) and scale it to the [0, 255] range.
* Map each pixel value to its corresponding value in the equalized CDF.

**Pseudo code:**

Calculate histogram of the image

Calculate cumulative distribution function (CDF) of the histogram

Scale CDF to range [0, 255]

For each pixel (i, j) in the image:

output[i, j] = CDF[image[i, j]]

Return output

**5. affine\_transform(image, a, b)**

**Purpose**: Applies an affine transformation (rotation, scaling, or translation) to the image.  
**Details**:

* For each pixel, calculate the new coordinates using the formula x\_new = a[0] + a[1] \* x + a[2] \* y and y\_new = b[0] + b[1] \* x + b[2] \* y.
* Assign the pixel value to the new coordinates.

**Pseudo code:**

For each pixel (i, j) in the image:

x\_new = a[0] + a[1] \* j + a[2] \* i

y\_new = b[0] + b[1] \* j + b[2] \* i

If x\_new and y\_new are within image boundaries:

output[y\_new, x\_new] = image[i, j]

Return output

**6. nearest\_neighbor\_resize(image, scale\_x, scale\_y)**

**Purpose**: Resizes the image using the nearest neighbor interpolation method.  
**Details**:

* Calculate the new image dimensions as new\_width = scale\_x \* old\_width and new\_height = scale\_y \* old\_height.
* For each pixel in the resized image, find the nearest pixel in the original image and assign its value.

**Pseudo code:**

Calculate new height = height \* scale\_y

Calculate new width = width \* scale\_x

For each pixel (i, j) in the resized image:

x = floor(j / scale\_x)

y = floor(i / scale\_y)

output[i, j] = image[y, x]

Return output

**7. linear\_interpolation\_resize(image, scale\_x, scale\_y)**

**Purpose**: Resizes the image using linear interpolation to produce smoother results.  
**Details**:

* For each pixel in the resized image, calculate its position in the original image.
* Interpolate the surrounding pixels along both axes (x and y) using linear interpolation

**Pseudo code:**

Calculate new height = height \* scale\_y

Calculate new width = width \* scale\_x

For each pixel (i, j) in the resized image:

x = j / scale\_x

y = i / scale\_y

x0 = floor(x)

x1 = min(x0 + 1, width - 1)

y0 = floor(y)

y1 = min(y0 + 1, height - 1)

alpha = x - x0

beta = y - y0

output[i, j] = (1 - alpha) \* (1 - beta) \* image[y0, x0]

+ alpha \* (1 - beta) \* image[y0, x1]

+ (1 - alpha) \* beta \* image[y1, x0]

+ alpha \* beta \* image[y1, x1]

Return output

**8. bilinear\_transform(image, a, b)**

**Purpose**: Performs bilinear interpolation for resizing or transforming the image.  
**Details**:

* Calculate the new pixel position using the formula x\_new = a[0] + a[1] \* j + a[2] \* i + a[3] \* j \* i and y\_new = b[0] + b[1] \* j + b[2] \* i + b[3] \* j \* i.
* Interpolate the pixel value based on the surrounding pixels.

**Pseudo code:**

For each pixel (i, j) in the image:

x\_new = a[0] + a[1] \* j + a[2] \* i + a[3] \* j \* i

y\_new = b[0] + b[1] \* j + b[2] \* i + b[3] \* j \* i

If x\_new and y\_new are within image boundaries:

x\_new\_int = round(x\_new)

y\_new\_int = round(y\_new)

output[i, j] = image[y\_new\_int, x\_new\_int]

Return output

**9. averaging\_filter(image, kernel\_size)**

**Purpose**: Applies an averaging filter (box blur) to smooth the image.  
**Details**:

* For each pixel, calculate the average value of the surrounding pixels using a square kernel of size kernel\_size.
* The result is applied to each pixel in the image.

**Pseudo code:**

For each pixel (i, j) in the image:

Define a region around (i, j) of size kernel\_size x kernel\_size

output[i, j] = sum(region) / (kernel\_size \* kernel\_size)

Return output

**10. gaussian\_kernel(size, sigma)**

**Purpose**: Generates a Gaussian kernel for use in Gaussian filtering.  
**Details**:

* Calculate the Gaussian function for each pixel in the kernel based on the size (size) and standard deviation (sigma).
* Normalize the kernel so that the sum of all values equals 1.

**Pseudo code:**

For each (i, j) in the kernel:

x = i - size // 2

y = j - size // 2

kernel[i, j] = exp(-(x^2 + y^2) / (2 \* sigma^2))

Normalize kernel so that sum(kernel) = 1

Return kernel

11. **gaussian\_filter(image, kernel)**

**Purpose**: Applies a Gaussian filter to the image to blur it.  
**Details**:

* For each pixel, calculate the weighted sum of the surrounding pixels using the Gaussian kernel.
* The result is applied to each pixel in the image.

**Pseudo code:**

For each pixel (i, j) in the image:

Define a region around (i, j) of size equal to the kernel

output[i, j] = sum(region \* kernel)

Return output

**12. median\_filter(image, kernel\_size)**

**Purpose**: Applies a median filter to smooth the image and reduce noise.  
**Details**:

* For each pixel, replace its value with the median of the surrounding pixel values within a kernel of size kernel\_size.

**Pseudo code:**

For each pixel (i, j) in the image:

Define a region around (i, j) of size kernel\_size x kernel\_size

output[i, j] = median(region)

Return output

13. **gaussian\_blur(image, kernel)**  
**Purpose**: Applies Gaussian blur to the image using a predefined Gaussian kernel.  
**Details**:

* For each pixel, calculate the weighted sum of neighboring pixels using the Gaussian kernel.
* The result is applied to each pixel in the image

**Pseudo code:**

For each pixel (i, j) in the image:

Define a region around (i, j) of size equal to the kernel

output[i, j] = sum(region \* kernel)

Return output