University of Moratuwa

Department of Electronic and Telecommunication Engineering

EN3160 - Image Processing and Machine Vision

Intensity Transformations and Neighborhood Filtering

Balasooriya B.A.P.I.
220054N

August 8, 2025

1 Question 1: Basic Intensity Transformation

1.1 Implementation

Description of the intensity transformation implementation based on Fig. 1a.

```
# Include relevant code snippet
def intensity_transform(image):
    # Your implementation here
    pass
```

1.2 Results

(a) Original Image

(b) Transformed Image

Figure 1: Intensity transformation results

1.3 Discussion

Analysis of the transformation effects and visual changes observed.

2 Question 2: Brain Image Enhancement

2.1 White Matter Accentuation

(a) Original

(b) White Matter Enhanced

(c) Transformation Plot

Figure 2: White matter enhancement

2.2 Gray Matter Accentuation

(a) Original (b) Gray Matter Enhanced (c) Transformation Plot

Figure 3: Gray matter enhancement

3 Question 3: Gamma Correction

3.1 L*a*b* Color Space Conversion

Explanation of color space conversion and gamma correction application.

3.2 Gamma Value Selection

Report the chosen γ value and justification.

3.3 Histogram Analysis

(a) Original Histogram (b) Gamma Corrected Histogram

Figure 4: Histogram comparison before and after gamma correction

4 Question 4: Vibrance Enhancement

4.1 HSV Decomposition

(a) Hue

(b) Saturation

(c) Value

Figure 5: HSV plane decomposition

4.2 Vibrance Transformation

Implementation of the given formula:

$$f(x) = \min\left(\mu x + a \times 128 e^{-\frac{(x-128)^2}{2\sigma^2}}, 255\right)$$

where $a \in [0, 1]$ and $\sigma = 70$.

4.3 Parameter Selection

Report the chosen value of parameter a and justification.

4.4 Results Comparison

Show original, enhanced, and transformation curve.

5 Question 5: Histogram Equalization

5.1 Custom Implementation

Python Implementation:

```python
def histogram_equalization(image):
    # Your custom Python implementation
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])
    cdf = hist.cumsum()
    # Normalize and apply transformation
    pass
```

C++ Implementation (Alternative):

```cpp
Mat histogramEqualization(const Mat& image) {
    Mat result;
    // Calculate histogram
    vector<int> hist(256, 0);
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            hist[image.at<uchar>(i, j)]++;
        }
    }
    // Your custom C++ implementation
    return result;
}
```

5.2 Results and Analysis

Compare histograms before and after equalization.

6 Question 6: Foreground Histogram Equalization

6.1 HSV Analysis and Masking

Process of selecting appropriate plane for thresholding.

6.2 Foreground Extraction

Use of `cv.bitwise_and` for foreground isolation.

6.3 Selective Histogram Equalization

Application of histogram equalization only to foreground.

6.4 Results

Show all intermediate steps and final result.

7 Question 7: Sobel Filtering

7.1 Method 1: Using filter2D

Implementation using existing OpenCV functions.

Python:

```python
# Sobel kernels
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.
    float32)
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.
    float32)

grad_x = cv2.filter2D(image, cv2.CV_32F, sobel_x)
grad_y = cv2.filter2D(image, cv2.CV_32F, sobel_y)
```

C++:

```
// Sobel kernels
Mat sobel_x = (Mat_<float>(3,3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);
Mat sobel_y = (Mat_<float>(3,3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

Mat grad_x, grad_y;
filter2D(image, grad_x, CV_32F, sobel_x);
filter2D(image, grad_y, CV_32F, sobel_y);
```

7.2 Method 2: Custom Implementation

Your own Sobel filter implementation.

7.3 Method 3: Separable Filters

Using the property:

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

7.4 Results Comparison

Compare outputs from all three methods.

8 Question 8: Image Zooming

8.1 Interpolation Methods

Implementation of both nearest-neighbor and bilinear interpolation.

```
def zoom_image(image, scale_factor, method='bilinear'):
    # Your implementation
    pass
```

8.2 Performance Evaluation

Normalized Sum of Squared Differences (SSD) calculations:

$$\text{Normalized SSD} = \frac{\sum_{i,j}(I_1(i,j) - I_2(i,j))^2}{\sum_{i,j} I_1(i,j)^2}$$

8.3 Results Analysis

Compare visual quality and SSD values.

9 References

- OpenCV Documentation: `https://opencv.org/`
- Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
- Stack Overflow and OpenCV forums for implementation help.