TP5: Image Enhancement Techniques

Detailed Explanation of Image Enhancement Methods

April 15, 2025

Contents

1 Introduction to Image Enhancement

Image enhancement is a fundamental process in image processing that aims to improve the visual appearance of an image or to convert the image to a form better suited for analysis by humans or machines. The primary objectives of enhancement include:

- Improving the contrast and brightness
- Highlighting features of interest
- Reducing noise and other artifacts
- Making details more visible

In this report, we explore various image enhancement techniques implemented in TP5, focusing primarily on:

1. Point operations (intensity transformations)
2. Histogram-based methods
3. Spatial domain filtering
4. Combined approaches

2 Point Operations (Intensity Transformations)

2.1 Gamma Correction

Gamma correction is a nonlinear operation used to encode and decode luminance in images. It is defined by the power-law expression:

$$V_{out} = V_{in}^{\gamma} \tag{1}$$

where V_{in} is the input intensity value normalized between 0 and 1, and γ is the gamma value.

- $\gamma < 1$: Brightens the image, enhancing details in darker regions
- $\gamma > 1$: Darkens the image, enhancing details in brighter regions
- $\gamma = 1$: No change

2.1.1 Implementation

Our implementation in `gamma_correction.py` applies various gamma values to an image:

```
# Apply gamma correction
gamma_values = [0.3, 0.5, 1.0, 1.5, 2.5]
gamma_corrected = [np.power(img, gamma) for gamma in
    gamma_values]
```

2.2 Contrast Stretching

Contrast stretching (also called normalization) is a simple technique to improve the contrast in an image by stretching the range of intensity values. The formula is:

$$g(x, y) = \frac{f(x, y) - \min(f)}{\max(f) - \min(f)} \times (L - 1) \tag{2}$$

where $f(x, y)$ is the input image, $g(x, y)$ is the output image, and L is the maximum intensity value (e.g., 256 for 8-bit images).

2.2.1 Implementation

In `contrast_stretching.py`, we implement this as:

```
def contrast_stretch(img, low_percentile=2,
    high_percentile=98):
    """
    Stretch the contrast of an image by mapping
    intensities to
    fill the range from 0 to 1
    """
    # Find low and high percentiles to clip histogram
    low, high = np.percentile(img, [low_percentile,
    high_percentile])

    # Apply contrast stretching
    stretched = np.clip((img - low) / (high - low), 0, 1)

    return stretched
```

3 Histogram-Based Enhancement

3.1 Histogram Equalization

Histogram equalization is a method that improves contrast by effectively spreading out the most frequent intensity values. The mathematical expression for histogram equalization is:

$$h(v) = \text{round}\left((L-1)\sum_{i=0}^{v} p(i)\right) \tag{3}$$

where $p(i)$ is the probability of intensity i in the image, and L is the number of possible intensity values.

3.1.1 Implementation

Our implementation in `histogram_equalization.py` includes a custom function:

```python
def custom_histogram_equalization(image):
    """
    Apply histogram equalization to an image
    T(x_k) = L * cdf_I(k)
    """
    # Get image histogram
    hist, bins = np.histogram(image.flatten(), bins=256,
   range=(0, 1))

    # Calculate cumulative distribution function
    cdf = np.cumsum(hist)
    cdf_normalized = cdf / cdf[-1]  # Normalize to [0,1]

    # Apply histogram equalization using CDF as a mapping
    function
    equalized = np.interp(image.flatten(), bins[:-1],
   cdf_normalized)

    return equalized.reshape(image.shape)
```

We also compare this with scikit-image's built-in implementation:

```python
# Using scikit-image's implementation
eq_skimage = exposure.equalize_hist(img)
```

3.2 Histogram Matching

Histogram matching (or specification) transforms an image so that its histogram matches a specified target histogram. The technique follows these steps:

1. Compute histograms and CDFs of both source and target images
2. For each intensity r in the source image, find its corresponding CDF value s
3. Find the intensity z in the target histogram such that the target CDF at z equals s

4. Map the intensity r to z in the output image

3.2.1 Implementation

In `histogram_matching.py`, we implement this method both manually and using scikit-image:

```python
def match_histogram_manual(source, target_hist):
    """Match the histogram of an image to a target histogram."""
    # Get the source histogram and calculate CDFs
    src_values, src_counts = np.unique(source.ravel(), return_counts=True)
    src_quantiles = np.cumsum(src_counts).astype(np.float64)
    src_quantiles /= src_quantiles[-1]

    # Calculate the target CDF
    target_quantiles = np.cumsum(target_hist)

    # Create the mapping
    interp_values = np.interp(src_quantiles, target_quantiles,
                              np.arange(len(target_quantiles)))

    # Map the source values to the new values
    mapping_func = lambda x: interp_values[np.searchsorted(src_values, x)]

    # Apply the mapping to each pixel
    matched = np.vectorize(mapping_func)(source)

    # Normalize to [0, 1]
    matched = (matched - matched.min()) / (matched.max() - matched.min())

    return matched
```

3.2.2 Bimodal Target Distribution

For the Phobos image, we created a bimodal target distribution:

```python
def create_bimodal_distribution(size=256, mode1=40, mode2=170,
                                spread1=15, spread2=25, weight1=0.3):
    """Create a bimodal distribution as target histogram."""
    x = np.arange(size)
```

```python
    y1 = np.exp(-(x - mode1)**2 / (2 * spread1**2))
    y2 = np.exp(-(x - mode2)**2 / (2 * spread2**2))
    y = weight1 * y1 + (1 - weight1) * y2
    return y / np.sum(y)
```

4 Combined Enhancement Approaches

In `combined_enhancement.py`, we demonstrate how different enhancement methods can be combined to achieve better results:

```python
# Apply multiple enhancement steps
img_denoised = gaussian(img, sigma=1)
img_stretched = contrast_stretch(img_denoised, 5, 95)
img_equalized = exposure.equalize_hist(img_stretched)
img_gamma = np.power(img_equalized, 1.2)  # Slight gamma
    adjustment
```

5 Case Study: Enhancing Phobos Image

5.1 Synthetic Generation of Phobos

In `phobos_synthetic.py`, we create a synthetic image of Phobos (Mars' moon) with its characteristic features:

1. Elliptical shape
2. Large Stickney crater
3. Medium and small craters
4. Grooves and furrows across the surface

```python
def create_phobos():
    """Create a synthetic image of Phobos."""
    # Create a base image (256x256 pixels)
    img = np.zeros((256, 256))

    # Create the basic elliptical shape of Phobos
    rr, cc = ellipse(128, 128, 90, 70, img.shape)
    img[rr, cc] = 0.6  # Set base brightness

    # Add large crater (Stickney)
    rr, cc = disk((100, 140), 25, shape=img.shape)
    img[rr, cc] = 0.5

    # Add medium craters and smaller features...
```

5.2 Histogram Analysis and Enhancement

The Phobos image is then used for various enhancement techniques:

1. Histogram equalization to improve overall contrast
2. Histogram matching to a bimodal distribution to highlight specific features
3. Comparison between different methods

6 Results and Analysis

6.1 Gamma Correction Results

Gamma correction provides different enhancement effects depending on the gamma value:

- Low gamma ($\gamma = 0.3, 0.5$): Brightens the image, revealing details in shadows
- High gamma ($\gamma = 1.5, 2.5$): Darkens the image, enhancing details in highlights

6.2 Histogram Equalization vs. Matching

- **Histogram Equalization**: Provides better overall contrast but can over-enhance noise and lose some subtle details
- **Histogram Matching**: Allows targeted enhancement based on the desired histogram shape

The comparison between the original Phobos image, its equalized version, and the version matched to a bimodal histogram shows:

- Equalization enhances small craters and surface details
- Matching to a bimodal distribution accentuates the distinction between the crater regions and the regular surface

7 Conclusion

Image enhancement is a critical preprocessing step for both human interpretation and computer vision applications. The techniques explored in TP5 demonstrate multiple approaches:

- Point operations provide simple but effective contrast enhancement

- Histogram-based methods offer more sophisticated control over the intensity distribution
- Combined approaches can address multiple image quality issues simultaneously

The implementation of these techniques in Python using libraries such as NumPy, Matplotlib, and scikit-image demonstrates both the mathematical foundations and practical applications of image enhancement algorithms.

8 Further Reading

- Digital Image Processing, R. C. Gonzalez and R. E. Woods
- Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, and S. L. Eddins
- scikit-image: Image processing in Python, S. van der Walt et al.