References


Appendix A (Methodology)

Appendix A, Figure 1: High level design diagram of pixel base watermarking system

Original image data

Spatial Analyzer
Spatial information of image

Watermark Generator
Watermark image data

Watermark Generator
Generated watermark data

Watermark Data

Embedder
Row Watermarked Image
\[ F_{wd} = F_{h} + F_{w} \]

Watermarked image data

Watermarked Image
\[ F_{wd} \]

Decoder
Uncompressed image

Uncompressed image

Extractor

Meta Data

Watermarked Image
\[ F_{wd} \]

Original Image
\[ F_{h} \]

Watermark Image
\[ F_{w} \]
Appendix A, Figure 2: High level design diagram of feature base watermarking system
Appendix A, Figure 3: Mexican hat operator with different sigma values. (Source: http://journals.plos.org)
Appendix A, Figure 3: Encoder module. (Source: http://www.eetimes.com/)
Appendix B (Experimental Design)

Dataset Used in Research

Name: lena1.bmp
Format: bit map
Dimension: 640x640 Px
Size: 1.2Mb

Name: lena2.jpg
Format: jpg
Dimension: 512x512 Px
Size: 94.4 Kb

Name: lena3.png
Format: png
Dimension: 512x512 Px
Size: 473.8 Kb

Name: monky.png
Format: png
Dimension: 512x512 Px
Size: 626.9 Kb
Experimental results of pixel based watermark

Experiment 1: Embed using random insertion

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method</td>
<td>Watermarked image</td>
</tr>
<tr>
<td>Technique</td>
<td>Watermarked image</td>
</tr>
<tr>
<td>Original image</td>
<td>Metadata file</td>
</tr>
<tr>
<td>Watermark</td>
<td>Result Lena.jpg</td>
</tr>
<tr>
<td></td>
<td>Result Lena.csv</td>
</tr>
</tbody>
</table>

Compression parameters:
- Block size: 8x8
- Quality: 70%
- JSAMPROW row pointer size: 1

<table>
<thead>
<tr>
<th>Original image</th>
<th>Watermark image</th>
<th>Watermarked image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension: 512x512</td>
<td>Dimension: 48x48</td>
<td>Dimension: 512x512</td>
</tr>
<tr>
<td>Size: 94.4 Kb</td>
<td>Size: 22.8 Kb</td>
<td>Size: 37.9 Kb</td>
</tr>
</tbody>
</table>
Experiment 2: Embed using less sensitive points of human vision

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method</td>
<td>Pixel base</td>
</tr>
<tr>
<td>Technique</td>
<td>less sensitive points of human vision</td>
</tr>
<tr>
<td>Original image</td>
<td>lena.jpg</td>
</tr>
<tr>
<td>Watermark</td>
<td>Watermark 4</td>
</tr>
<tr>
<td>Watermarked image</td>
<td>result_lena.jpg</td>
</tr>
<tr>
<td>Metadata file</td>
<td>result_lena.csv</td>
</tr>
</tbody>
</table>

Compression parameters:
Block size: 8x8
Quality: 70%
JSAMPROW row pointer size: 1

<table>
<thead>
<tr>
<th>Original image</th>
<th>Watermark image</th>
<th>Watermarked image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension: 512x512</td>
<td>Dimension: 48x48</td>
<td>Dimension: 512x512</td>
</tr>
<tr>
<td>Size: 94.4 Kb</td>
<td>Size: 22.8 Kb</td>
<td>Size: 37.9 Kb</td>
</tr>
</tbody>
</table>

Experiment 3: Embed using LSB of macro-block

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method</td>
<td>Pixel base</td>
</tr>
<tr>
<td>Technique</td>
<td>LSB of macro-block</td>
</tr>
<tr>
<td>Original image</td>
<td>lena.jpg</td>
</tr>
<tr>
<td>Watermark</td>
<td>Watermark 4</td>
</tr>
<tr>
<td>Watermarked image</td>
<td>result_lena.jpg</td>
</tr>
<tr>
<td>Metadata file</td>
<td>result_lena.csv</td>
</tr>
</tbody>
</table>

Compression parameters:
Block size: 8x8
Quality: 70%
JSAMPROW row pointer size: 1
Experimental results of pixel based watermark extraction methods

Experiment 4: Extraction using common algorithm

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method: Pixel base</td>
<td>Extracted watermark: wtm.jpg</td>
</tr>
<tr>
<td>Technique: common</td>
<td></td>
</tr>
<tr>
<td>Original image: result_lena.jpg</td>
<td></td>
</tr>
<tr>
<td>Metadata file: result_lena.csv</td>
<td></td>
</tr>
</tbody>
</table>

Compression parameters:
Block size: 8x8
Quality: 70%
JSAMPROW row pointer size: 1
Experimental results of feature based watermark extraction methods

Experiment 5: Embed using Harris corner detector

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method</td>
<td>Feature base</td>
</tr>
<tr>
<td>Possession</td>
<td>Around single corner</td>
</tr>
<tr>
<td>Original image</td>
<td>lena.jpg</td>
</tr>
<tr>
<td>Watermark</td>
<td>Watermark 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Watermarked image</td>
<td>result_lena.jpg</td>
</tr>
<tr>
<td>Metadata file</td>
<td>result_lena.csv</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature detector</td>
<td>Harris operator</td>
</tr>
<tr>
<td>Smoothing</td>
<td>Gaussian filter</td>
</tr>
<tr>
<td>Kernel</td>
<td>5x5</td>
</tr>
<tr>
<td>Sigma</td>
<td>1</td>
</tr>
<tr>
<td>Threshold</td>
<td>120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compression parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>8x8</td>
</tr>
<tr>
<td>Quality</td>
<td>70%</td>
</tr>
<tr>
<td>JSAMPROW row pointer size</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original image</th>
<th>Binary corner image</th>
<th>Watermarked image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension: 512x512</td>
<td>Dimension: 512x512</td>
<td>Dimension: 512x512</td>
</tr>
<tr>
<td>Size: 37.9 Kb</td>
<td>Size: 16.2 Kb</td>
<td>Size: 47.4 Kb</td>
</tr>
</tbody>
</table>

Experiment 6: Embed using novel corner detector

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Base method</td>
<td>Feature base</td>
</tr>
<tr>
<td>Position</td>
<td>Around single corner</td>
</tr>
<tr>
<td>Original image</td>
<td>lena.jpg</td>
</tr>
<tr>
<td>Watermark</td>
<td>Watermark 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Watermarked image</td>
<td>result_lena.jpg</td>
</tr>
<tr>
<td>Metadata file</td>
<td>result_lena.csv</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature detector</td>
<td>Novel operator</td>
</tr>
<tr>
<td>Smoothing</td>
<td>LoG</td>
</tr>
</tbody>
</table>

94
| Compression parameters | Block size: 8x8  
|------------------------|------------------  
| Quality: 70%           | JSAMPROW row pointer size: 1 |

| Original image  
Dimension: 512x512  
Size: 37.9 Kb | Binary corner image  
Dimension: 512x512  
Size: 18.3 Kb | Watermarked image  
Dimension: 512x512  
Size: 46.2 Kb |
Appendix C (Evaluation)

**Evaluate of the Robustness**

<table>
<thead>
<tr>
<th>Settings &amp; Attack</th>
<th>Watermark</th>
<th>Watermarked Image</th>
<th>Watermarked Image After Attack</th>
</tr>
</thead>
</table>
| Feature detector: Harris operator  
Smoothing: Gaussian  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: single corner | ![Watermark](image1) | ![Watermarked Image](image2) | ![Watermarked Image After Attack](image3) |
| **HSV noise**  
[Holdness: 3]  
[Hue:72]  
[Saturation:146]  
[Value: 94] | ![Watermark](image1) | ![Watermarked Image](image2) | ![Watermarked Image After Attack](image3) |
| Feature detector: Novel operator  
Smoothing: LoG  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: single corner | ![Watermark](image1) | ![Watermarked Image](image2) | ![Watermarked Image After Attack](image3) |
| **HSV noise**  
[Holdness: 3]  
[Hue:72]  
[Saturation:146]  
[Value: 94] | ![Watermark](image1) | ![Watermarked Image](image2) | ![Watermarked Image After Attack](image3) |
<table>
<thead>
<tr>
<th>Feature detector: Harris operator</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing: Gaussian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sigma: 1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position: single corner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Random Pik.**
[Random seeds: 1452988117]
[Randomization: 42]
[Repeat: 5]

<table>
<thead>
<tr>
<th>Feature detector: Novel operator</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing: LoG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sigma: 1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position: single corner</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Random Pik.**
[Random seeds: 1452988117]
[Randomization: 42]
[Repeat: 5]
<table>
<thead>
<tr>
<th>Feature detector: Novel operator</th>
<th>Smoothing: LoG</th>
<th>Kernel: 5x5</th>
<th>Sigma: 1.5</th>
<th>Threshold: 125</th>
<th>Position: single corner</th>
</tr>
</thead>
</table>

**Rotation in 45 degree**

<table>
<thead>
<tr>
<th>Feature detector: Novel operator</th>
<th>Smoothing: LoG</th>
<th>Kernel: 5x5</th>
<th>Sigma: 1.5</th>
<th>Threshold: 125</th>
<th>Position: single corner</th>
</tr>
</thead>
</table>

**Scaling**
watermarked image: 512x512
scaled to: 256x256

<table>
<thead>
<tr>
<th>Feature detector: Harris operator</th>
<th>Smoothing: Gaussian</th>
<th>Kernel: 5x5</th>
<th>Sigma: 1</th>
<th>Threshold: 125</th>
<th>Position: single corner</th>
</tr>
</thead>
</table>

**HSV noise**
[Holdness: 3]
[Hue: 72]
[Saturation: 146]
[Value: 94]
<table>
<thead>
<tr>
<th>Feature detector: Novel operator</th>
<th>![Image]</th>
<th>![Image]</th>
<th>![Image]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing: LoG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position: single corner</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**HSV noise**

- Holdness: 3
- Hue: 72
- Saturation: 146
- Value: 94

<table>
<thead>
<tr>
<th>Feature detector: Harris operator</th>
<th>![Image]</th>
<th>![Image]</th>
<th>![Image]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing: Gaussian</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position: single corner</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Random Pik.**

- Random seeds: 1452988117
- Randomization: 42
- Repeat: 5
### Evaluate of the Fidelity

<table>
<thead>
<tr>
<th>Image Detail &amp; Settings</th>
<th>Evaluation Results</th>
<th>Original Image</th>
<th>Watermarked Image</th>
</tr>
</thead>
</table>
| **Image**: lena.jpg (512x512, 94.4 Kb) | **Evaluation Method 1:** MSE  
Resulting Value: 102.03  
Conclusion: Very good fidelity | | |
| **Settings**  
Feature detector: Harris operator  
Smoothing: Gaussian  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: single corner | **Evaluation Method 2:** PSNR  
Resulting Value: 39.28  
Conclusion: Good fidelity | | |
| | **Evaluation Method 3:** SSIM  
Resulting Value:  
Channel[0]: 0.69  
Channel[1]: 0.75  
Channel[2]: 0.74  
Mean: 0.72  
Conclusion: Good | | |
| **Image:** lena.jpg (512x512, 94.4 Kb) | **Settings** | **Evaluation Method 1:** MSE  
Resulting Value: 113.03  
Conclusion: Very good fidelity  
Evaluation Method 2:  
**PSNR**  
Resulting Value: 41.28  
Conclusion: Good fidelity  
Evaluation Method 3:  
**SSIM**  
Chanel[0]: 0.69  
Chanel[1]: 0.75  
Chanel[2]: 0.74  
Mean: 0.72  
Conclusion: Good Fidelity |
| **Feature detector:** Harris operator  
**Smoothing:** Gaussian  
**Kernel:** 5x5  
**Sigma:** 1  
**Threshold:** 125  
**Position:** multiple corners |

| **Image:** lena.jpg (512x512, 94.4 Kb) | **Settings** | **Evaluation Method 1:** MSE  
Resulting Value: 76.03  
Conclusion: Very good fidelity  
Evaluation Method 2:  
**PSNR**  
Resulting Value: 44.36  
Conclusion: Very good fidelity  
Evaluation Method 3:  
**SSIM**  
Chanel[0]: 0.79  
Chanel[1]: 0.85 |
| **Feature detector:** Novel operator  
**Smoothing:** LoG  
**Kernel:** 5x5  
**Sigma:** 1  
**Threshold:** 125  
**Position:** single corner |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lena.jpg</td>
<td>Resulting Value: 54.44</td>
<td>Resulting Value: 45.51</td>
<td>Resulting Value: 0.79</td>
</tr>
<tr>
<td>Settings</td>
<td>Conclusion: Very good fidelity</td>
<td>Conclusion: Very good fidelity</td>
<td>Conclusion: Very good fidelity</td>
</tr>
<tr>
<td>Feature detector: Novel operator</td>
<td></td>
<td></td>
<td>Channel[0]: 0.79</td>
</tr>
<tr>
<td>Smoothing: LoG</td>
<td></td>
<td></td>
<td>Channel[1]: 0.85</td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
<td>Channel[2]: 0.84</td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td></td>
<td>Mean: 0.82</td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
<td>Conclusion: Very good fidelity</td>
</tr>
<tr>
<td>Position: multiple corners</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>monky.png</td>
<td>Resulting Value: 202.03</td>
<td>Resulting Value: 38.80</td>
<td>Resulting Value: 0.79</td>
</tr>
<tr>
<td>Settings</td>
<td>Conclusion: Very good fidelity</td>
<td>Conclusion: Good fidelity</td>
<td>Conclusion: Very good fidelity</td>
</tr>
<tr>
<td>Feature detector: Harris operator</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smoothing: Gaussian</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Position: single corner</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mean: 0.82
Conclusion: Very good fidelity
<table>
<thead>
<tr>
<th>Image: monky.png (512x512, 626.9 Kb)</th>
<th>Settings</th>
<th>Evaluation Method 1: MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature detector: Harris operator</td>
<td></td>
<td>Resulting Value: 203.03</td>
</tr>
<tr>
<td>Smoothing: Gaussian</td>
<td></td>
<td>Conclusion: Very good</td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td>fidelity</td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td>Evaluation Method 2: PSNR</td>
</tr>
<tr>
<td>Position: multiple corners</td>
<td></td>
<td>Resulting Value: 38.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conclusion: Good fidelity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Evaluation Method 3: SSIM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resulting Value: Chanel[0]: 0.69</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chanel[1]: 0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chanel[2]: 0.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mean: 0.72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conclusion: Good Fidelity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image: monky.png (512x512, 626.9 Kb)</th>
<th>Settings</th>
<th>Evaluation Method 1: MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature detector: Novel operator</td>
<td></td>
<td>Resulting Value: 146.03</td>
</tr>
<tr>
<td>Smoothing: LoG</td>
<td></td>
<td>Conclusion: Very good</td>
</tr>
<tr>
<td>Kernel: 5x5</td>
<td></td>
<td>fidelity</td>
</tr>
<tr>
<td>Sigma: 1</td>
<td></td>
<td>Evaluation Method 2: PSNR</td>
</tr>
<tr>
<td>Threshold: 125</td>
<td></td>
<td>Resulting Value: 44.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conclusion: Very good</td>
</tr>
</tbody>
</table>
| Position: single corner | **Evaluation Method 3:** SSIM  
Resulting Value:  
Chanel[0]: 0.79  
Chanel[1]: 0.85  
Chanel[2]: 0.84  
Mean: 0.82  
Conclusion: Very good Fidelity |
| --- | --- |
| **Image:** monkey.png  
(512x512, 626.9 Kb)  
**Settings**  
Feature detector: Novel operator  
Smoothing: LoG  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: multiple corners | **Evaluation Method 1:** MSE  
Resulting Value: 54.44  
Conclusion: Very good fidelity  
**Evaluation Method 2:** PSNR  
Resulting Value: 45.01  
Conclusion: Very good fidelity  
**Evaluation Method 3:** SSIM  
Resulting Value:  
Chanel[0]: 0.80  
Chanel[1]: 0.85  
Chanel[2]: 0.85  
Mean: 0.84  
Conclusion: Very good Fidelity |

*Appendix C, Table 2: Summary of evaluation results for the fidelity*
## Evaluate of the Capacity

<table>
<thead>
<tr>
<th>Image Detail &amp; Settings</th>
<th>Evaluation Results (Statistically)</th>
<th>Evaluation Results (Experimentally)</th>
<th>Evaluation Results (Experimentally)</th>
</tr>
</thead>
</table>
| **Image:** lena.jpg (512x512, 94.4 Kb) | **Evaluation Method 1:** MSE  
Resulting Value: 103.03  
Conclusion: Very good fidelity | **Original Image** | **Watermarked Image** |
| **Watermark object:**  
Name: watermark 4.jpg  
Dimension: 48x48 px,  
Size: 22.8 Kb | **Evaluation Method 2:** PSNR  
Resulting Value: 41.28  
Conclusion: Good fidelity | | |
| **Settings**  
Feature detector: Harris operator  
Smoothing: Gaussian  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: multiple corner | **Evaluation Method 3:** SSIM  
Resulting Value:  
Chanel[0]: 0.69  
Chanel[1]: 0.75  
Chanel[2]: 0.74  
Mean: 0.72  
Conclusion: Good Fidelity | | |
| **Image:** lena.jpg (512x512, 94.4 Kb) | **Evaluation Method 1:** MSE  
Resulting Value: 183.04  
Conclusion: Fair fidelity | | |
| **Watermark object:**  
Name: watermark 4.jpg  
Dimension: 64x64 px,  
Size: 27.6 Kb | **Evaluation Method 2:** PSNR  
Resulting Value: 37.44  
Conclusion: Fair fidelity | | |
<p>| | <strong>Evaluation Method 3:</strong> SSIM | | |</p>
<table>
<thead>
<tr>
<th><strong>Settings</strong></th>
<th>Feature detector: Harris operator</th>
<th>Resulting Value: Chanel[0]: 0.69</th>
<th>Chanel[1]: 0.75</th>
<th>Chanel[2]: 0.74</th>
<th>Mean: 0.72</th>
<th>Conclusion: Good Fidelity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Smoothing: Gaussian</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kernel: 5x5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sigma: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Threshold: 125</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Position: multiple corners</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Image:</strong></td>
<td>lena.jpg</td>
<td>(512x512, 94.4 Kb)</td>
<td>Evaluation Method 1: MSE</td>
<td>Resulting Value: 344.05</td>
<td>Conclusion: Very bad fidelity</td>
<td></td>
</tr>
<tr>
<td><strong>Watermark object:</strong></td>
<td>Name: watermark 4.jpg</td>
<td>Dimension: 128x128 px, Size: 29.8 Kb</td>
<td>Evaluation Method 2: PSNR</td>
<td>Resulting Value: 23.82</td>
<td>Conclusion: Very bad fidelity</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Evaluation Method 3: SSIM</td>
<td>Resulting Value: Chanel[0]: 0.59</td>
<td>Chanel[1]: 0.64</td>
<td>Chanel[2]: 0.66</td>
<td>Mean: 0.63</td>
</tr>
</tbody>
</table>
**Image:** lena.jpg  
(512x512, 94.4 Kb)  

**Watermark object:**  
Name: watermark 4.jpg  
Dimension: 48x48 px,  
Size: 22.8 Kb  

**Settings**  
Feature detector: Novel operator  
Smoothing: LoG  
Kernel: 5x5  
Sigma: 1  
Threshold: 125  
Position: multiple corners

**Evaluation Method 1:**  
MSE  
Resulting Value: 54.44  
Conclusion: Very good fidelity

**Evaluation Method 2:**  
PSNR  
Resulting Value: 45.51  
Conclusion: Very good fidelity

**Evaluation Method 3:**  
SSIM  
Resulting Value:  
Chanel[0]: 0.79  
Chanel[1]: 0.85  
Chanel[2]: 0.84  
Mean: 0.82  
Conclusion: Very good Fidelity

---

**Image:** lena.jpg  
(512x512, 94.4 Kb)  

**Watermark object:**  
Name: watermark 4.jpg  
Dimension: 64x64 px,  
Size: 27.6 Kb  

**Settings**  
Feature detector: Novel operator  
Smoothing: LoG  
Kernel: 5x5  
Sigma: 1

**Evaluation Method 1:**  
MSE  
Resulting Value: 143.04  
Conclusion: Good fidelity

**Evaluation Method 2:**  
PSNR  
Resulting Value: 40.04  
Conclusion: Good fidelity

**Evaluation Method 3:**  
SSIM  
Resulting Value:  
Chanel[0]: 0.69  
Chanel[1]: 0.75  
Chanel[2]: 0.70  
Mean: 0.71  
Conclusion: Good Fidelity
<table>
<thead>
<tr>
<th><strong>Threshold:</strong> 125</th>
<th><strong>Position:</strong> multiple corners</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image:</strong> lena.jpg (512x512, 94.4 Kb)</td>
<td><strong>Evaluation Method 1:</strong> MSE</td>
</tr>
<tr>
<td><strong>Watermark object:</strong> Name: watermark 4.jpg Dimension: 128x128 px, Size: 27.6 Kb</td>
<td>Resulting Value: 163.04</td>
</tr>
<tr>
<td><strong>Settings</strong> Feature detector: Novel operator Smoothing: LoG Kernel: 5x5 Sigma: 1</td>
<td>Conclusion: Bad fidelity</td>
</tr>
<tr>
<td><strong>Evaluation Method 2:</strong> PSNR</td>
<td>Resulting Value: 40.04</td>
</tr>
<tr>
<td></td>
<td>Conclusion: Fair fidelity</td>
</tr>
<tr>
<td><strong>Evaluation Method 3:</strong> SSIM</td>
<td>Resulting Value:</td>
</tr>
<tr>
<td></td>
<td>Chanel[0]: 0.60</td>
</tr>
<tr>
<td></td>
<td>Chanel[1]: 0.65</td>
</tr>
<tr>
<td></td>
<td>Chanel[2]: 0.66</td>
</tr>
<tr>
<td></td>
<td>Mean: 0.64</td>
</tr>
<tr>
<td></td>
<td>Conclusion: Fair Fidelity</td>
</tr>
</tbody>
</table>

Appendix C, Table 3: Summary of evaluation results for the capacity
Appendix D (Prototype System)

Structure of Prototype Application

/-------------------------------------------------------- root
/CMake Files ------------------------------------------- build files

/GUI ------------------------------------------------- GUI of application
/GUI/images -------------------------------- application images
/GUI/source -------------------------------- source-code of GUI application
   main.cpp

/GUI/source/class --------------------- C++ class files of GUI application
   evaluate.cpp
   mainwindow.cpp
   featurewindow.cpp
   pixelwindow.cpp

/GUI/source=headers --------------------- C++ header files of GUI application
   mainwindow.h

/GUI/source/includes ------------------- custom header files
   headers.h

/GUI/gui.pro ---------------------------- configuration file

/images --------------------------------- input image
/result ------------------------------- output
/source -------------------------------- source code of watermarking application
   main.cpp
/source/compress ----------------------------- encoder algorithms
    render_jpeg.cpp

/source/embed ------------------------------- embedding algorithms
    embed_watermark_corners_harris.cpp
    embed_watermark_corners_lochandaka.cpp
    embed_watermark_corners.cpp
    embed_watermark_pixel_full_fixed_inblock_position.cpp
    embed_watermark_pixel_full_fixed_position.cpp
    embed_watermark_pixel_full_vari_inblock_position.cpp
    embed_watermark_pixel.cpp

/source/evaluate ---------------------------- evaluation methods
    mse.cpp
    psnr.cpp
    ssim.cpp

/source/extract ----------------------------- extraction algorithms
    extract_watermark_corners_harris.cpp
    extract_watermark_corners_lochandaka.cpp
    extract_watermark_corners.cpp
    extract_watermark_pixel_full_common_position.cpp
    extract_watermark_pixel_full_fixed_inblock_position.cpp
    extract_watermark_pixel_full_fixed_position.cpp
    extract_watermark_pixel_full_vari_inblock_position.cpp
    extract_watermark_pixel.cpp

/source/header --------------------------------- headers
    function_declaration.h
    includes.h
    typedef.h
/source/operators ------------------------------------------- feature detection operators
    harris_operator.cpp
    lochandaka_operator.cpp

/source/read ----------------------------------------------- read image data
    read_jpeg.cpp
    read_image.cpp
    read_jpeg.cpp
    read_pixel.cpp

/source/res ----------------------------------------------- mathematical functions
    convert_color_space.cpp
    draw_circle.cpp
    draw_image.cpp
    filters.cpp
    get_corner_points.cpp
    get_image_dimensions.cpp
    intensity_diff.cpp
    math.cpp
    pixel_intensity_boundary_adjust.cpp
    read_watermark_meta.cpp

Total Lines of code: 6212
Main Screens of Prototype Application

Fig. 1. Appendix D, Figure 1: Main window of prototype system

Appendix D, Figure 2: Watermark embed & extraction window
Source code of novel watermark operators

/* Create namespace */
using namespace cv;
using namespace std;

DERIVATIVES katussa_sobel_operator(Mat image);
Mat katussa_gauss_operator_for_lochandaka(Mat IX, int kernal, float sigma);
Mat katussa_lochandaka_response(Mat GsX, Mat GsY, Mat GsXY, float k);
Mat katussa_nonmaximal_suppression_for_lochandaka(Mat response, int threshold);

/**
 * This function implement Harris operator
 * Parameters are image, threshold
 * Return type Mat
 * */
Mat katussa_lochandaka_operator(Mat image, float k, int kernal, float sigma, int threshold)
{
    Mat res_img(image.rows, image.cols, CV_8UC1, Scalar(0));
    Mat res_img_maxsupp(image.rows, image.cols, CV_8UC1, Scalar(0));
    Mat GsX(image.rows, image cols, CV_32F);
    Mat GsY(image.rows, image cols, CV_32F);
    Mat GsXY(image.rows, image cols, CV_32F);
    Mat response(image.rows, image cols, CV_32F);
    DERIVATIVES derivatives;

    // #### (1) Compute X and Y derivatives and compute products of derivatives
derivatives = katussa_sobel_operator(image);

// ### (2) Apply Gaussian operator to Ix, Iy, Ixy
Gsx = katussa_gauss_operator_for_lochandaka(derivatives.Ix, kernal, sigma);
Gsy = katussa_gauss_operator_for_lochandaka(derivatives.Iy, kernal, sigma);
Gsxy = katussa_gauss_operator_for_lochandaka(derivatives.Ixy, kernal, sigma);

// ### (3) Create matrix for each pixel(x,y) and compute response
response = katussa_lochandaka_response(Gsx, Gsy, Gsxy, k);

// ### (4) Non Maximum suppression of response
res_img_maxsupp = katussa_nonmaximal_suppression_for_lochandaka(response, threshold);

res_img = res_img_maxsupp;
return res_img;

/**
  * This function implements sobel operator
  * Parameters is Mat image
  * Return type DERIVATIVES
  *
  * DERIVATIVES katussa_sobel_operator(Mat image)
  *
  float derivative_Sx;
  float derivative_Sy;
  float DSx_2 = 0, DSy_2 = 0, DSxy = 0;

  Mat Ix(image.rows, image.cols, CV_32F);
  //Mat Ix(image.rows, image.cols, CV_8UC1, Scalar(0));
  Mat Iy(image.rows, image.cols, CV_32F);
  Mat Ixy(image.rows, image.cols, CV_32F);
  DERIVATIVES sobel_derivatives;

  // Sobel matrix in X direction
  int Sx[3][3] = {
    { -1, 0, 1 },
    { -2, 0, 2 },
    { -1, 0, 1 }
  };

  // Sobel matrix in Y direction
  int Sy[3][3] = {
    { 1, 2, 1 },
    { 0, 0, 0 },
    { -1, -2, -1 }
  };

  for (int y=1; y<image.rows-1; y++)
  {
    for (int x=1; x<image.cols-1; x++)
    {

// Compute X derivatives
derivative_Sx = 0.0;
for (int Sx_y = 0; Sx_y <= 2; Sx_y++)
{
    for (int Sx_x = 0; Sx_x <= 2; Sx_x++)
    {
        derivative_Sx += image.at<uchar>(y + (Sx_y-1), x + (Sx_x-1)) * Sx[Sx_y][Sx_x];
    }
}
derivative_Sx = abs(derivative_Sx);
//DSx_2 = derivative_Sx*derivative_Sx;
Ix.at<uchar>(y, x) = derivative_Sx;
//printf("%d ", Ix.at<uchar>(y, x));

// Compute Y derivatives
derivative_Sy = 0.0;
for (int Sy_y = 0; Sy_y <= 2; Sy_y++)
{
    for (int Sy_x = 0; Sy_x <= 2; Sy_x++)
    {
        derivative_Sy += image.at<uchar>(y + (Sy_y-1), x + (Sy_x-1)) * Sy[Sy_y][Sy_x];
    }
}
derivative_Sy = abs(derivative_Sy);
//DSx_2 = derivative_Sy*derivative_Sy;
Iy.at<uchar>(y, x) = derivative_Sy;
//printf("%.2f ", derivative_Sy);
DSxy = derivative_Sx*derivative_Sy;
Ixy.at<float>(y, x) = DSxy;
}
sobel_derivatives.Ix = Ix;
sobel_derivatives.Iy = Iy;
sobel_derivatives.Ixy = Ixy;
return sobel_derivatives;

/**
 * This function implement gauss operator
 * Parameters is Mat sobel_derivatives
 * Return type Mat
 *
 * Mat katussa_gauss_operator_for_lochandaka(Mat IX, int kernal, float sigma)
 {
 Mat Gs_Mat(IX.rows, IX.cols, CV_32F);
 //int Gs[7][7];
 //float Gs_multiplier;
 float Gs_Ivalue;
if(kernel==5 &
  & sigma==1){
    int Gs[5][5] = {
        { 1, 4, 7, 4, 1 },
        { 4, 16, 26, 16, 4 },
        { 7, 26, 41, 26, 7 },
        { 4, 16, 26, 16, 4 },
        { 1, 4, 7, 4, 1 }
    };
    float Gs_multiplier = 1/273;
    for (int y=2; y<IX.rows-2; y++)
    {
        for (int x=2; x<IX.cols-2; x++)
        {
            Gs_Ivalue = 0.0;
            for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
            {
                for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
                {
                    Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
                //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
                //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
                }
            }
            Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;
        }
    }
}
if(kernel==5 &
  & sigma==2){
    int Gs[5][5] = {
        { 2, 7, 12, 7, 2 },
        { 7, 31, 52, 31, 7 },
        { 12, 52, 127, 52, 12 },
        { 7, 31, 52, 31, 7 },
        { 2, 7, 12, 7, 2 }
    };
    float Gs_multiplier = 1/571;
    for (int y=2; y<IX.rows-2; y++)
    {
        for (int x=2; x<IX.cols-2; x++)
        {
            Gs_Ivalue = 0.0;
            for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
            {
                for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
                {
                    Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2)) * Gs[Gs_y][Gs_x];
                //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-
2));
                //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
                }
            }
            Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;
        }
    }
}
if(kernal==5 && sigma==1.5){
    int Gs[5][5] = {
        {2, 4, 5, 4, 2 },
        {4, 9, 12, 9, 4 },
        {5, 12, 15, 12, 5 },
        {4, 9, 12, 9, 4 },
        {2, 4, 5, 4, 2 }
    };
    float Gs_multiplier = 1/159;
    for (int y=2; y<IX.rows-2; y++)
    {
        for (int x=2; x<IX.cols-2; x++)
        {
            Gs_Ivalue = 0.0;
            for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
            {
                for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
                {
                    Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-2)) * Gs[Gs_y][Gs_x];
                    //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-2));
                    //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
                }
            }
            Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;
        }
    }
}

if(kernal==7 && sigma==1){
    int Gs[7][7] = {
        {1, 4, 7, 4, 1 },
        {4, 16, 26, 16, 4 },
        {7, 26, 41, 26, 7 },
        {4, 16, 26, 16, 4 },
        {1, 4, 7, 4, 1 }
    };
    float Gs_multiplier = 1/273;
    for (int y=2; y<IX.rows-2; y++)
    {
        for (int x=2; x<IX.cols-2; x++)
        {
            Gs_Ivalue = 0.0;
            for (int Gs_y = 0; Gs_y <= 4; Gs_y++)
            {
                Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-2)) * Gs[Gs_y][Gs_x];
                //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-2));
                //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
            }
            Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;
        }
    }
}
for (int Gs_x = 0; Gs_x <= 4; Gs_x++)
{
    Gs_Ivalue += IX.at<float>(y + (Gs_y-2), x + (Gs_x-2)) * Gs[Gs_y][Gs_x];
    //Gs_Ivalue = IX.at<float>(y + (Gs_y-2), x + (Gs_x-2));
    //Gs_Ivalue2 += Gs_Ivalue * Gs[Gs_y][Gs_x];
}

Gs_Mat.at<float>(y, x) = Gs_Ivalue*Gs_multiplier;

return Gs_Mat;

/**
 * This function implement gauss operator
 * Parameters are Mat Gsx, Mat Gsy, Mat Gsxy
 * Return type Mat
 * *
 * Mat katussa_lochandaka_response(Mat Gsx, Mat Gsy, Mat Gsxy, float k)
 {
     Mat response(Gsx.rows, Gsx.cols, CV_32F);
     float a11, a12, a21, a22;
     float det;
     float trace;
     for (int y=0; y<Gsx.rows; y++)
     {
         for (int x=0; x<Gsx.cols; x++)
         {
             a11 = Gsx.at<float>(y, x) * Gsx.at<float>(y, x);
             a22 = Gsy.at<float>(y, x) * Gsy.at<float>(y, x);
             a12 = Gsxy.at<float>(y, x);
             a21 = Gsxy.at<float>(y, x);
             det = (a11*a22) - (a12*a21);
             trace = a11 + a22;
             //printf("%f ", det);
             response.at<float>(y,x) = abs( (det) - (k * (trace*trace)) ); //abs( Gsx.at<float>(y, x)*Gsy.at<float>(y, x) );
             //printf("%f ", response.at<float>(y,x));
         }
     }
     return response;
 }
Mat katussa_nonmaximal_suppression_for_lochandaka(Mat response, int threshold)
{
    Mat res_img_maxsupp(response.rows, response.cols, CV_8UC1, Scalar(0));
    int value;
    for (int y=1; y<response.rows; y++)
    {
        for (int x=1; x<response.cols; x++)
        {
            //printf("%d,%d-%.2f \# ",y, x, res_img_maxsupp.at<float>(y, x));
            //printf("%d,%d-%d \n ",y, x, res_img_maxsupp.at<uchar>( y, x));
            if(response.at<uchar>(y, x) < threshold)
                value = 0;
            else
            {
                value = 255;
            }
            res_img_maxsupp.at<uchar>(y, x) = value;
            //res_img_maxsupp.at<uchar>(y, x) = response.at<uchar>(y, x);
        }
    }
    return res_img_maxsupp;
}

Source code of watermark generator

int katussa_intensity_diff_value(int intensity_original, int intensity_watermark)
{
    int intensity_diff;
    char *binary;
    binary = (char*)malloc(32+1);

    // Check intensity is < 240.
    if(intensity_original>1 && intensity_original<248)
    {
        // Compare intensity values of original image and watermark.
        if(intensity_original>=intensity_watermark)
        {
            // LSB of difference is '0' add '1' make odd difference.
            // Difference odd means intensity of original image > intensity of watermark.
            binary = katussa_decimal_to_binary(sqrt(intensity_original-intensity_watermark));
            if(atoi(&binary[31])==0)
                intensity_diff = katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),1);
            else
                intensity_diff = katussa_binary_addition(katussa_binary_to_decimal(atoi(binary)),0);
        }
    }
}
Source code of embedder

/**
 * This function embed a watermark into original image
 * Arguments: int orgimg_width, int orgimg_height (Dimensions of original image)
 * Arguments: int wtmimg_width, int wtmimg_height (Dimensions of watermark image)
 * Arguments: IMG_MATRIX *pixeli_original_image (Pixel values of original image)
 * Arguments: IMG_MATRIX *pixeli_watermark_image (Pixel values of watermark image)
 * This function call several algorithms
 * Return type IMG_MATRIX
 *
IMG_MATRIX *katussa_embed_watermark_corners_lochandaka_single(char *watermark_meta_file_name,
IMG_MATRIX_GRAY *corner_point,
int orgimg_width, int orgimg_height,
int wtmimg_width, int wtmimg_height,
IMG_MATRIX *pixeli_original_image,
IMG_MATRIX *pixeli_watermark_image)
{
int index = orgimg_width*orgimg_height*41;
IMG_MATRIX *img_matrix = NULL;
img_matrix = (IMG_MATRIX*) malloc(index);

FILE *watermark_meta_file;
FILE *watermark_info_file;
char *watermark_info_file_name;

// Create watermark meta file
watermark_meta_file = fopen(watermark_meta_file_name, "w+");

// Declare the local variables for embed algorithm
long int total_pixel_image = orgimg_width*orgimg_height;
int total_pixel_watermark = wtmimg_width*wtmimg_height;
int x, y;
int intensity_diff_r, intensity_diff_g, intensity_diff_b;
//img watermark_embed_index = (total_pixel_image/total_pixel_watermark)-(wtmimg_width/2);

long int count = 0;
for(long int i=0; i<total_pixel_image; i++)
{
    x = i%orgimg_width;
    y = i/orgimg_width;
    if( (y>=corner_point->y && y<corner_point->y+wtmimg_width) && (x>=corner_point->x && x<corner_point->x+wtmimg_height) )
    {
        for(int j=0; j<total_pixel_watermark; j++)
        {
            if(count==j)
            {
                intensity_diff_r = katussa_intensity_diff_value(pixeli_original_image[i].intensity_r, pixeli_watermark_image[j].intensity_r);
                intensity_diff_g = katussa_intensity_diff_value(pixeli_original_image[i].intensity_g, pixeli_watermark_image[j].intensity_g);
                intensity_diff_b = katussa_intensity_diff_value(pixeli_original_image[i].intensity_b, pixeli_watermark_image[j].intensity_b);

                img_matrix[i].x = x;
                img_matrix[i].y = y;
                img_matrix[i].intensity_r = pixeli_original_image[i].intensity_r + intensity_diff_r;
                img_matrix[i].intensity_g = pixeli_original_image[i].intensity_g + intensity_diff_g;
                img_matrix[i].intensity_b = pixeli_original_image[i].intensity_b + intensity_diff_b;

                fprintf(watermark_meta_file, "%d, %d, %d, %d\n", j, intensity_diff_r, intensity_diff_g, intensity_diff_b);
            }
        }
        count++;
    }
    else
    {
        img_matrix[i].x = x;
        img_matrix[i].y = y;
    }
}
img_matrix[i].intensity_r = pixeli_original_image[i].intensity_r;
img_matrix[i].intensity_g = pixeli_original_image[i].intensity_g;
img_matrix[i].intensity_b = pixeli_original_image[i].intensity_b;

fclose(watermark_meta_file);

// Create watermark info file
watermark_info_file_name = (char*)malloc(100);
strcpy(watermark_info_file_name, watermark_meta_file_name);
strcat(watermark_info_file_name, ".info");
watermark_info_file = fopen(watermark_info_file_name, "w+");
fprintf(watermark_info_file, "%s, %d, %d, %d, %d
",
        watermark_meta_file_name, orgimg_width, orgimg_height,
wtmimg_width, wtmimg_height);
return img_matrix;
}

Source code of extractor

/**
 * This function extract a watermark from watermarked image & watermark meta file
 * Arguments: char* extract method
 * Arguments: char* watermark_meta_file_name
 * Arguments: int orgimg_width, int orgimg_height (Dimentions of original image)
 * Arguments: IMG_MATRIX *pixeli_watermarked_image (Pixel values of watermarked image)
 * Arguments: IMG_MATRIX *pixeli_watermark_meta (Pixel values of watermark meta file)
 * This function call several algorithms
 * Return type IMG_MATRIX
 */

IMG_MATRIX *katussa_extract_watermark_corners_single(IMG_MATRIX_GRAY *corner_point,

    int watermarked_width, int watermarked_height,

    int wtmimg_width, int wtmimg_height,

    IMG_MATRIX *pixeli_watermarked_image, Mat

    watermarked_image,

    char *watermark_mata_file_name)
{
    int index = wtmimg_width*wtmimg_height*41;
    IMG_MATRIX *img_matrix = NULL;
    img_matrix = (IMG_MATRIX*)malloc(index);

    long int total_pixel_image = watermarked_width*watermarked_height;

    img_matrix[i].intensity_r = pixeli_original_image[i].intensity_r;
    img_matrix[i].intensity_g = pixeli_original_image[i].intensity_g;
    img_matrix[i].intensity_b = pixeli_original_image[i].intensity_b;

    fclose(watermark_meta_file);

    // Create watermark info file
    watermark_info_file_name = (char*)malloc(100);
    strcpy(watermark_info_file_name, watermark_meta_file_name);
    strcat(watermark_info_file_name, ".info");
    watermark_info_file = fopen(watermark_info_file_name, "w+");
    fprintf(watermark_info_file, "%s, %d, %d, %d, %d
",
            watermark_meta_file_name, orgimg_width, orgimg_height,
wtmimg_width, wtmimg_height);
    return img_matrix;
}
```c
int total_pixel_watermark = wtmimg_width*wtmimg_height;
int x_wtm, y_wtm, x_wtmmed, y_wtmmed; // dimention of watermark image
int intensity_orgi_r=0, intensity_orgi_g=0, intensity_orgi_b=0;
int intensity_extractwtm_r=0, intensity_extractwtm_g=0, intensity_extractwtm_b=0;

// Declare image metrix and allocate memory
int index_wtm= total_pixel_watermark*36;
IMG_DATA *wtm_data = NULL;
wtm_data = malloc(index_wtm);

wtm_data = katussa_read_watermark_meta_data(watermark_mata_file_name);
/*for(long int i=0; i<total_pixel_watermark; i++)
{
    printf("%d,%d,%d", wtm_data[i].index, wtm_data[i].intensity_r, wtm_data[i].intensity_g, wtm_data[i].intensity_b);
}*/

long int count = 0;
for(int i=0; i<total_pixel_watermark; i++)
{
    x_wtm = i%wtmimg_width;
y_wtm = i/ wtmmimg_width;
    //printf("%d,%d", img_matrix[i].y, img_matrix[i].y);
    for(int j=corner_point->y+y_wtm; j<corner_point->y+1+y_wtm; j++)
    {
        for(int k=corner_point->x; k<corner_point->x+48; k++)
        {
            img_matrix[i].y = y_wtm;
            img_matrix[i].x = x_wtm;
            //printf("%d,%d", j, k);
            //printf("%d,%d", img_matrix[i].y, img_matrix[i].x);
            int b = watermarked_image.at<cv::Vec3b>(j, k)[0];
            int g = watermarked_image.at<cv::Vec3b>(j, k)[1];
            int r = watermarked_image.at<cv::Vec3b>(j, k)[2];
            //printf("%d,%d", j, k, b);
            //img_matrix[i].intensity_r = r;
            //img_matrix[i].intensity_g = g;
            //img_matrix[i].intensity_b = b;

            // If intensity_r odd (if LSB is 1)
            if( (wtm_data[count].intensity_r%2)==1 )
            {
                intensity_orgi_r = r - wtm_data[count].intensity_r;
                intensity_extractwtm_r = intensity_orgi_r -
                (wtm_data[count].intensity_r * wtm_data[count].intensity_r);
                img_matrix[i].intensity_r =
katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_r);
            }
            // If intensity_g odd (if LSB is 1)
            if( (wtm_data[count].intensity_g%2)==1 )
            {
                intensity_orgi_g = g - wtm_data[count].intensity_g;
                intensity_extractwtm_g = intensity_orgi_g -
                (wtm_data[count].intensity_g * wtm_data[count].intensity_g);
                img_matrix[i].intensity_g =
katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_g);
            }
            // If intensity_b odd (if LSB is 1)
        }
    }
}
```

123
if ( (wtm_data[count].intensity_b%2)==1 )
{
    intensity_ori_b = b - wtm_data[count].intensity_b;
    intensity_exctrctwtm_b = intensity_ori_b -
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
    img_matrix[j].intensity_b =
katussa_pixel_intensity_boundary_adjust(intensity_exctrctwtm_b);
}

// If intensity_r odd (if LSB is 0)
if( (wtm_data[count].intensity_r%2)==0 )
{
    intensity_ori_r = r - wtm_data[count].intensity_r;
    intensity_exctrctwtm_r = intensity_ori_r +
(wtm_data[count].intensity_r * wtm_data[count].intensity_r);
    img_matrix[i].intensity_r =
katussa_pixel_intensity_boundary_adjust(intensity_exctrctwtm_r);
}

// If intensity_g odd (if LSB is 0)
if( (wtm_data[count].intensity_g%2)==0 )
{
    intensity_ori_g = g - wtm_data[count].intensity_g;
    intensity_exctrctwtm_g = intensity_ori_g +
(wtm_data[count].intensity_g * wtm_data[count].intensity_g);
    img_matrix[i].intensity_g =
katussa_pixel_intensity_boundary_adjust(intensity_exctrctwtm_g);
}

// If intensity_b odd (if LSB is 0)
if( (wtm_data[count].intensity_b%2)==0 )
{
    intensity_ori_b = b - wtm_data[j].intensity_b;
    intensity_exctrctwtm_b = intensity_ori_b +
(wtm_data[count].intensity_b * wtm_data[count].intensity_b);
    img_matrix[i].intensity_b =
katussa_pixel_intensity_boundary_adjust(intensity_exctrctwtm_b);
}
}

//printf("%ld, ", count);
count++;

return img_matrix;

/**
 * This function extract a watermark from watermarked image & watermark meta file
 * Arguments: char* extract method
 * Arguments: char* watermark_meta_file_name
 * Arguments: int orgimg_width, int orgimg_height (Dimentions of original image)
 * Arguments: IMG_MATRIX *pixeli_watermarked_image (Pixel values of watermarked image)
 * Arguments: IMG_MATRIX *pixeli_watermark_meta (Pixel values of watermark meta file)
 * This function call several algorithms
 * Return type IMG_MATRIX
 * TODO NEED TO DEVELOPMENT FROM THE SCRCH
 */
IMG_MATRIX *katussa_extract_watermark_corners_multiple(IMG_MATRIX_GRAY *corner_point,
    int watermarked_width, int watermarked_height,
    int wtimg_width, int wtimg_height,
    IMG_MATRIX *pixeli_watermarked_image, Mat watermarked_image,
    char *watermark_mata_file_name)
{
    int index = wtimg_width*wtimg_height*41;
    IMG_MATRIX *img_matrix = NULL;
    img_matrix = (IMG_MATRIX*) malloc(index);
    long int total_pixel_image = watermarked_width*watermarked_height;
    long int total_pixel_watermark = wtimg_width*wtimg_height;
    int x_wtm, y_wtm, x_wtmed, y_wtmef; // dimension of watermark image
    int intensity_orgi_r=0, intensity_orgi_g=0, intensity_orgi_b=0;
    int intensity_extractwtm_r=0, intensity_extractwtm_g=0, intensity_extractwtm_b=0;

    // Declare image matrix and allocate memory
    int index_wtm= total_pixel_watermark*36;
    IMG_DATA *wtm_data = NULL;
    wtm_data = (IMG_DATA*) malloc(index_wtm);
    wtm_data = katussa_read_watermark_meta_data(watermark_mata_file_name);
    /*for(long int i=0; i<total_pixel_watermark; i++)
    { printf("%d(%d, %d, %d)# ", wtm_data[i].index, wtm_data[i].intensity_r,
        wtm_data[i].intensity_g, wtm_data[i].intensity_b);
    }*/
    long int count = 0;
    for(int i=0; i<total_pixel_watermark; i++)
    {
        x_wtm = i%wtimg_width;
        y_wtm = i/ wtimg_width;
        //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].x);
        for(int j=corner_point->y+y_wtm; j<corner_point->y+1+y_wtm; j++)
        {
            for(int k=corner_point->x+x_wtm; k<corner_point->x+48; k++)
            {
                img_matrix[i].y = y_wtm;
                img_matrix[i].x = x_wtm;
                //printf("%d,%d# ", j, k);
                //printf("%d,%d# ", img_matrix[i].y, img_matrix[i].x);
                int b = watermarked_image.at<cv::Vec3b>(j, k)[0];
                int g = watermarked_image.at<cv::Vec3b>(j, k)[1];
                int r = watermarked_image.at<cv::Vec3b>(j, k)[2];
                //printf("(%d,%d)-%d " , j, k, b);
                //img_matrix[i].intensity_r = r;
//img_matrix[i].intensity_g = g;
//img_matrix[i].intensity_b = b;

// If intensity_r odd (if LSB is 1)
if( (wtm_data[count].intensity_r%2)==1 )
{
    intensity_orgi_r = r - wtm_data[count].intensity_r;
    intensity_extractwtm_r = intensity_orgi_r - (wtm_data[count].intensity_r * wtm_data[count].intensity_r);
    img_matrix[i].intensity_r = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_r);
}

// If intensity_g odd (if LSB is 1)
if( (wtm_data[count].intensity_g%2)==1 )
{
    intensity_orgi_g = g - wtm_data[count].intensity_g;
    intensity_extractwtm_g = intensity_orgi_g - (wtm_data[count].intensity_g * wtm_data[count].intensity_g);
    img_matrix[j].intensity_g = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_g);
}

// If intensity_b odd (if LSB is 1)
if( (wtm_data[count].intensity_b%2)==1 )
{
    intensity_orgi_b = b - wtm_data[count].intensity_b;
    intensity_extractwtm_b = intensity_orgi_b - (wtm_data[count].intensity_b * wtm_data[count].intensity_b);
    img_matrix[j].intensity_b = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_b);
}

// If intensity_r odd (if LSB is 0)
if( (wtm_data[count].intensity_r%2)==0 )
{
    intensity_orgi_r = r - wtm_data[count].intensity_r;
    intensity_extractwtm_r = intensity_orgi_r + (wtm_data[count].intensity_r * wtm_data[count].intensity_r);
    img_matrix[i].intensity_r = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_r);
}

// If intensity_g odd (if LSB is 0)
if( (wtm_data[count].intensity_g%2)==0 )
{
    intensity_orgi_g = g - wtm_data[count].intensity_g;
    intensity_extractwtm_g = intensity_orgi_g + (wtm_data[count].intensity_g * wtm_data[count].intensity_g);
    img_matrix[i].intensity_g = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_g);
}

// If intensity_b odd (if LSB is 0)
if( (wtm_data[count].intensity_b%2)==0 )
{
    intensity_orgi_b = b - wtm_data[j].intensity_b;
    intensity_extractwtm_b = intensity_orgi_b + (wtm_data[count].intensity_b * wtm_data[count].intensity_b);
    img_matrix[i].intensity_b = katussa_pixel_intensity_boundary_adjust(intensity_extractwtm_b);
}
} } 

// printf("%ld ", count); 
count++; 
} 

return img_matrix;