COM31006 Assignment

Lucy Lau

May 2025

Introduction

This report details the implementation of image-to-image steganography as per the requirements of the COM31006 module. A video demo of the application is linked here: `https://youtu.be/MhpaLAekoRo`

1 SIFT Keypoint Detection

OpenCV's built-in SIFT functions were used to detect keypoints. OpenCV's implementation is based on the algorithm described in D. Lowe's Distinctive Image Features from Scale-Invariant Keypoints, 2004. There are four main steps.

1.1 Scale-space Extrema Detection

Scale-space filtering is a technique used in computer vision where an image is analyzed at multiple levels of detail in order to detect features with varying sizes. For SIFT, a Laplacian of Gaussian (LoG) is applied to the image with varying σ values, which detects blobs. Varying σ allows features of different sizes to be detected. However, LoG is costly, so the SIFT algorithm approximates it by using Difference of Gaussians (DoG), which is obtained by taking the difference between two Gaussian blurred images with two different σ. Next, local extrema are located for the DoGs, and these are the potential keypoints.

1.2 Keypoint Localization

Once the keypoints have been found, they are processed. The Taylor expansion is used to get a more accurate location of extrema, and rejects the keypoint if the intensity is less than a threshold value (0.03 in the paper). Edges are also removed using a 2x2 Hessian matrix to computer principal curvature, and discarding points if they are less than an edge threshold (10 in the paper).

1.3 Orientation Assignment

An orientation is assigned to each keypoint to achieve invariance to image rotation. A neighbourhood around each keypoint location in which the gradient and magnitude is calculated for that region. An orientation histogram is created, and the highest peak in the histogram and any peak above 80% are used to calculate the orientation.

1.4 Keypoint Descriptor

Each keypoint has a descriptor - a vector of size 128 describing the region around that point. Several measures are taken to achieve robustness against different image transformations such as rotation and lighting changes.

2 Watermark Embedding

2.1 Watermark Processing

First, the watermark is processed:

1. Watermark image is converted to grayscale.
2. Grayscale watermark is resized to (15, 15).
3. Using OpenCV threshold, each pixel value becomes either 0 or 1.

The threshold function for each grayscale pixel value is given below:

$$f(x) = \begin{cases} 0 & \text{if } x < 127 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

2.2 Watermark Embedding

After the watermark is processed, it is ready to be embedded into the cover image. The pseudocode is shown in Algorithm 1.

Algorithm 1 Pseudocode for watermark embedding.

```
$keypoints = SIFT(cover\_img)$
$N = 100$                                   ▷ Number of keypoints to embed the watermark
$i = 0$
$count = 0$          ▷ Keeps track of number of watermarks embedded successfully

while $i < len(keypoints)$ and $count < N$ do
    if $is\_valid(keypoints[i])$ then
        $embed\_keypoint(cover\_img, keypoints[i])$
        $count = count + 1$
        $i = i + 1$
    else
        $i = i + 1$
    end if
end while
```

First, SIFT is used to detect keypoints. The keypoints are sorted by their response to ensure the most significant keypoints are chosen to embed. A value of 100 was chosen for N to ensure enough keypoints cover the entire image to protect against cropping. The *is_valid* function checks if the keypoint is valid for embedding the watermark based on two conditions:

- If the watermark goes out of bounds of the image.
- If the watermark overlaps with another embedded watermark.

A boolean mask (not shown in the pseudocode) is used to detect overlap; after a watermark is embedded, the mask is updated with the pixel values used to embed the watermark. *embed_watermark* embeds the watermark into a 15x15 region centered on the keypoint in the image by changing the least significant bit (LSB) of only the blue channel, since it is least the least perceptible to the human eye.

3 Watermark Recovery

The pseudocode for watermark recovery is given in Algorithm 2.

Algorithm 2 Pseudocode for watermark recovery.

```
$keypoints = SIFT(img)$
$N = 100$                      ▷ Number of keypoints embedded with watermark
$i = 0$
$count = 0$                    ▷ Keeps track of number of watermarks detected
$threshold = 0.95$

while $i < len(keypoints)$ and $count < N$ do
    if $is\_valid(keypoints[i])$ then
        $result = detect\_watermark(img, keypoints[i])$
        if $result == True$ then
            $count = count + 1$
        end if
        $i = i + 1$
    else
        $i = i + 1$
    end if
end while
    return $count/N > threshold$
```

The function *detect_watermark* compares the LSB of the 15x15 region centered around that keypoint with the watermark and returns true if they match, meaning that the watermark was indeed embedded at that keypoint. The *threshold* is the minimum proportion of detected keypoints for an image to be considered authentic. For example, if there were 100 keypoints embedded into an image but only 98 keypoints were detected, then the image would still be classed as authentic. This is due to the nature of this implementation, as it was found that SIFT would occasionally detect slightly different keypoints in the watermarked image compared to the original, however it was normally only a small number of differing keypoints. If the proportion of detected keypoints is below the threshold, then that indicates that the image has been tampered with.

4 Watermark Invisibility and Robustness

Since the watermark embedding processing is performed by modifying the LSB in the blue color channel only, it is invisible to the human eye. This is because firstly the human eye is least sensitive to the blue color channel, and secondly the pixel value only change by 1, which is an insignificant change that cannot be detected by the human eye. It is also invisible to SIFT, as it will still detect the same set of keypoints on a watermarked image as the original. The watermark is also embedded at 100 keypoints so it covers the majority of the image, meaning it is robust to cropping tampering techniques. Robustness of the watermark also depends on what shape the watermark is. For example, if the watermark is a symmetric shape, it may not be robust to rotation as a rotated watermark will still look the same as the original.

5 Summary

In summary, this application can successfully embed a watermark into several keypoints (detected using SIFT) of an image by modifying the LSB of the blue channel in the image. The watermark is invisible to the human eye after embedding, but can be detected by checking the LSB of the blue channel at each keypoint.

5.1 Challenges

The first challenge encountered was how to process the watermark. Since the implementation requires modify the LSB, the watermark had to be converted into some binary representation. There are several different ways to do this, but for simplicity I chose to use binary thresholding and the watermark image I used for testing was a binary black and white image. Then there was the problem of resizing the watermark so it is small enough to embed. Again, a simple approach of using OpenCV's resize function was used to resize the watermark down to (15, 15). There were other approaches that could have been taken (see Section Limitations and Improvements), such as splitting the watermark into smaller regions instead of resizing.

Another challenge encountered was how to embed the watermark such that it is invisible to the human eye and SIFT, but could still be recovered. This was overcome by choosing to modify only the blue channel of the cover image and using LSB to make imperceptible changes. As mentioned in Section 3, SIFT was not completely robust to this change as occasionally it detected a few different keypoints in the cover image compared to the original, but this was remedied by introducing a threshold parameter in the watermark recovery function. The keypoints detected by SIFT are also sorted based on their response (strength), meaning that these keypoints are the most robust to image changes and ensures they can still be detected after embedding watermarks.

5.2 Limitations and Improvements

The application can successfully detect if an image was tampered with, however it cannot detect the specific tampering technique used. A possible solution for this would be to reconstruct the watermark from the tampered image and compare it to the original watermark. Another improvement to the application already discussed would be to change how the watermark was processed. The watermark is resized to be 15x15, and this size was chosen as it is small and has odd dimensions (so it can be embedded at the center of a keypoint). However, resizing the watermark may lose information, especially if the watermark image is large. This could be remedied by splitting the watermark into smaller sections and embedding each of those sections instead of resizing, or by keeping the aspect ratio of the watermark the same when resizing. The watermark could also be resized relative to the size of the keypoint. Also the watermark is converted to grayscale, meaning it loses color information if it is a colored image. This could be remedied by directly converting the image into a binary representation using OpenCV's encode function. However, this means that a lot more information needs to be encoded, so there is a trade-off between how much of the original watermark information to retain when processing it versus how many watermarks can be embedded or how imperceptible the watermark will be after embedding. In terms of innovation, the current application may not be very innovative, however by implementing the improvements described the application could be more innovative.