COM31006
Computer Vision Assignment Report
May 2025

Joel Foster

Demo Video Link: https://youtu.be/_Vh7p5mNGaA

The following report documents the process of creating a watermark embedding, extraction, and tampering detection tool using steganography techniques and the SIFT feature extraction process. The tool was implemented in Python using the CustomTkinter and Tkinter libraries to create an interactive GUI.

Detecting Keypoints with SIFT

The approach used in the application to embed a watermark into an image was to detect key points of the image, and embed the watermark at each of these key points. In order to extract key points of an image, the Scale-Invariant Feature Transform (SIFT) algorithm was used. The idea of SIFT is that images are transformed into local feature spaces which are invariant to transforms such as translation, rotation, scaling, etc. The steps of the algorithm at a high level are as follows:

- Scale-space extrema detection - search for features that are stable over different scales by progressively smoothing the image with a Difference of Gaussian filter, increasing the standard deviations
- Keypoint localisation - in each Difference of Gaussian image, every point is compared to its 8 neighbour pixels in the current image and the 9 neighbours each in the images above and below. If the point is a local minima or maxima for these values it is considered as a potential keypoint.
- Orientation assignment - a histogram of the directions of local gradients is computed for each selected scale, and the peak of this smoothed histogram is assigned as the orientation of the keypoint
- Keypoint description - key points need descriptors in order to be matched, so local direction is set based on orientation assignment, and a vector with 16 histograms of the 8 directions in each 16th of the Gaussian smoothed window is generated.

The implementation of the SIFT algorithm in Python is provided in the openCV computer vision library. A SIFT object can be created with *sift = cv2.SIFT.create()*, and the key points are detected using *sift.detect()*. These key points can be plotted onto the original image using the function *cv2.drawKeypoints()*.

Encoding and Embedding the Watermark

In order to encode and embed the watermark, the watermark was first processed into a grayscale image, and then thresholded with a binary threshold of 127 and a max value of 1. This converts the image into a matrix of boolean 1s and 0s for each pixel based on whether they were white or black

in the watermark image. The main carrier image is also processed by converting every pixel value to an even value so that their least significant bits are all 0 by default.

Once the images have been pre-processed and the key points have been detected, a mask matrix for each RGB colour channel is created. These masks are used to store boolean values for each pixel in the carrier image, to determine if a watermark has been written on that specific colour channel at a specific position. Each key point is iterated over, and the coordinates that the watermark would be embedded to are calculated. These coordinates are then checked against the colour channel masks to check if there is any overlap with the watermark of another nearby key point. If there is an overlap for the checked colour channel, the remaining channels are checked and added to these instead if there are no overlaps there. The implementation prioritises colour channels in the order blue, green, and then red.

Before embedding a watermark at a key point, the key point is first checked to ensure that it is not at the edge of the image so that some of the watermark would be cut off.

Embedding the watermark involves iterating over the area around the key point of the size of the watermark, and updating the least significant bit of each corresponding pixel relative to the value in the binary watermark matrix. This is done by gathering the pixel value for a selected pixel, converting this value into binary, amending the least significant bit to the new value, and then converting back to an integer and updating the value stored in the image array.

Once the watermark has been embedded in a colour channel, the respective colour mask is updated. After every keypoint has been iterated over, the program displays the resultant image and allows the user to download it.

Watermark Recovery and Tampering Detection

The process of recovering the watermarks follows a similar process to the embedding. The key points of the image are detected again using the SIFT algorithm, and the uploaded watermark is converted to grayscale and thresholded into a binary matrix. Then, for every key point detected by the SIFT algorithm, the algorithm checks each colour channel to see if the watermark exists in any one of them. For example, the blue channel is checked first, and if there exists a watermark at the location that matches the uploaded watermark, a boolean flag storing whether the image has the watermark is set to true, and the program breaks out of the key point iteration. However, if there is no watermark at the key point in the blue channel, the next channel is then checked, which would be the green channel. This is repeated if the green channel also does not contain the watermark, and if the red channel does not either, it is deemed that the watermark does not exist at the current key point.

Before recovering each watermark at a key point, the key point is again checked to ensure that it is not at the edge of the image, as no watermark would have been embedded if so.

The actual process to retrieve a watermark from a keypoint is also similar to the embedding process. A function exists which iterates over the area around the keypoint of the size given by the size of the expected watermark. For each pixel in the area, the value is retrieved and converted into binary. The least significant bit of this binary value is taken and amended to a new matrix containing a representation of the retrieved watermark. After the function is finished iterating over each pixel in the area, the watermark matrix is returned. This matrix is then compared with the original watermark and if the two are equal, in the case of the authenticity verifier, true is returned, and for tampering detection, a count of correct matches is incremented by one.

For the authenticity verifier, if at least one watermark at any keypoint matches the supplied watermark, a value of Yes is returned, else No. For the tampering detection, a count of each correct match is kept, and after all key points have been iterated over, the count of correct matches is divided by the total number of key points to determine the consistency ratio of the watermarks at key points. If this consistency is less than 80%, the tampering detection determines the image to have been tampered with, or if higher, not tampered with. This effectively tests different tampering techniques such as cropping, resizing, rotating, and removing backgrounds, due to each of these methods either completely changing the pixels in the image by interpolating, moving, or scaling the pixels which will alter the watermark, or by removing sections of the image which will remove key points of the original image, reducing the consistency score.

Invisibility and Robustness of Watermarks

The invisibility of the watermarks is nearly perfect in terms of what the human eye can perceive. The following two pictures are before and after embedding a watermark. The difference is not noticeable.

Figure A - Before watermark is embedded

(https://commons.wikimedia.org/wiki/File:Slipknot_-_The_Grey_Chapter_Tour_2016_-_D%C3%BCsseldorf_-_01830705_-_Leonhard_Kreissig_-_Canon_EOS_5D_Mark_III.jpg)

Figure B - after watermark is embedded
(https://commons.wikimedia.org/wiki/File:Slipknot_-_The_Grey_Chapter_Tour_2016_-_D%C3%BCsseldorf_-_01830705_-_Leonhard_Kreissig_-_Canon_EOS_5D_Mark_III.jpg)

The watermarks themselves are fairly robust in the sense that they are encoded into all three colour channels for different keypoints, but they are not robust to tampering, such as resizing, rotating etc. as they are either completely removed due to interpolation or rotated so that they do not align with the key points anymore. This could be improved by embedding them in the direction of each keypoint descriptor, but it would be hard to embed on a key point that has a diagonal direction.

Visibility, Recovery and Resilience Tests

Multiple images similar to the image above were used to test visibility, with edge cases such as images containing mostly blue, green or red. All these tests proved that the watermarked image was not distinguishable from the original image. Recovery tests on images were performed for images with more or less features, and images of different sizes with different sized watermarks. In most cases, without tampering, around 90% of the watermarks were successfully recovered from the key points. However, in cases where the watermark was larger and the carrier image was smaller, the recovery rate was initially a lot lower at around 40% due to issues with overlapping. The solution was altered with the channel masking method described to reduce the impact of overlapping, and a check into whether key points were at the edge of an image was performed. This managed to increase the recovery rate to about 70% for these edge case images, which is an improvement but there is still room to further improve recovery rate.

Resilience tests were performed on images by applying common image manipulation techniques such as resizing, rotating, cropping, and removing backgrounds. In the cases of resizing and rotating, none of the watermarks were recovered, meaning that the system detected the image as

being tampered with but it could also be seen as a downside as the record of the watermark is completely erased. It is likely that this is unpreventable with the chosen approach due to the way the pixels of an image are interpolated when these manipulations are applied. However, cropping and removing backgrounds still recovers some watermarks but shows a significant drop in consistency of watermarks recovered, therefore being detected to be tampered by the system.

Modifications to Specification

Watermark recovery only needs to have one valid watermark to confirm that the image contains the watermark, which is a potential deviation from the ambiguous specification which seems to insinuate that all key points need to contain the watermark to be authenticated. This change was made as it makes more sense that if an image contains any instance of the watermark then it must have initially been through the watermarking process so must be in some form authentic, without considering modifications which is left to the tampering detection.

Also, whilst not directly specified in the specification, a decision was made that watermarks used were only allowed to be square watermarks of any size up to 16x16. This is to increase the performance of the tool, preventing any watermark overlapping.

Effectiveness of Approach

Whilst the implementation effectively embedded watermarks into the carrier images and prevented overlap, this affected the recovery process, as key points that didn't have a watermark embedded due to overlap seemed to be flagged as a tampered key point when recovering watermarks, leading to a false conclusion that some of the watermarks had been tampered with.

Limitations of the approach include limitations discussed earlier such as how easily watermark recovery can be prevented by simply resizing, rotating, or performing any manipulation that would cause pixels to be interpolated or moved. Also, if the watermarked image is converted to grayscale, the watermark would be destroyed.

Innovation

I believe the most innovative element of my approach would be the decision to use all three colour channels for embedding, alternating between each based on colour masks, in order to prevent watermark overlapping. This innovation essentially triples the amount of space in the image available for watermark embedding compared to only using one colour channel.