Steganography and Image Processing Application Report

1. Introduction

This application is a menu-driven steganography and image processing tool developed in Python. It demonstrates several techniques for hiding and extracting information within images. The program consists of four main modules, each accessible via a user-friendly menu. The key functionalities include:

- Capturing a selfie from a webcam and hiding a text message in it.
- Hiding and extracting a text message in/from any image.
- Hiding one image within another.
- Superimposing two images using bit-level manipulation.

2. Main Features

Option 1: Capture Selfie and Hide/Extract Text Message

- **Capture a Selfie:**
  Uses the computer's webcam to capture and save a selfie.
- **Hide a Text Message:**
  Embeds a secret message in the selfie by modifying the last least significant bit (LSB) of each pixel, ensuring minimal visual change.
- **Extract the Message:**
  Retrieves the hidden message from the modified image.
- **Display Results:**
  Shows the original and modified images side by side using Matplotlib.

Option 2: Hide/Extract Text in Any Image

- **User Input:**
  Accepts an image file path and a text message from the user.
- **LSB Steganography:**
  Hides the text message in the image using 1 LSB (which can be changed to 2 bits if needed).
- **Result Display:**
  Displays both the original and stego image and prints the recovered hidden message.

Option 3: Hide and Extract an Image Inside Another Image

- **Image Input:**
  Prompts the user for a cover image and a secret image.

- **Bit-level Embedding:**
  Embeds the secret image into the cover image by replacing the last 2 bits of the cover image with the 2 most significant bits of the secret image.
- **Extraction:**
  Extracts and displays the hidden secret image.

Option 4: Superimpose Two Images Using 4 Bits Each

- **Image Input:**
  Accepts two images from the user.
- **Superimposition:**
  Combines the images by retaining the upper 4 bits of the first image and embedding the upper 4 bits of the second image (shifted into the lower nibble) into it.
- **Result Display:**
  Displays the two original images and the final superimposed image.

3. Explanations and Observations

Text Steganography (Options 1 & 2)

- **Process:**
  The application converts the text into its binary representation and appends a 32-bit header indicating the message length. It then replaces the LSB of each pixel in the image with these bits.
- **Observation:**
  Using only 1 bit per pixel minimizes any visible distortion in the image, maintaining high visual fidelity. When compared to library functions, our manual approach provides more insight into how LSB steganography works and allows customization (e.g., adjusting the number of bits).

Image Hiding (Option 3)

- **Process:**
  The cover image's last 2 bits are cleared, and the 2 most significant bits from the secret image are extracted and embedded into these cleared bits.
- **Observation:**
  This approach preserves the overall appearance of the cover image while embedding sufficient information from the secret image. It illustrates the principle that subtle changes in pixel values are often imperceptible to the human eye.

Superimposition (Option 4)

- **Process:**
  The technique involves combining the upper 4 bits of the first image with the upper 4 bits

of the second image (after shifting them into the lower 4 bits). This results in a composite image where the first image is dominant, but the second image is also embedded.

- **Observation:**
  This method demonstrates how bit-level manipulation can create complex visual effects. In practice, the superimposed image appears closer to the first image, with only subtle traces of the second image visible.

Comparison with Library Functions

- **Custom vs. Library:**
  While many libraries (e.g., Stegano, Steganogan) offer built-in steganography functions, this application uses custom code to provide transparency into the underlying bit-level operations.
- **Flexibility:**
  Our code allows direct manipulation of individual bits, offering the flexibility to adjust the number of bits used for hiding information. In contrast, library functions may abstract these details, making it harder to understand the process.
- **Performance:**
  For educational purposes, our implementation is sufficient and performs well on moderately sized images. Library functions might offer optimizations and additional features for handling large datasets or providing more robust error-checking.

4. Conclusion

This assignment successfully demonstrates various steganography and image processing techniques through a menu-driven application. The custom implementation provides clear insights into the workings of LSB steganography and bit-level image manipulation, offering an educational contrast to more opaque library functions. The interactive interface and visual outputs at each stage help in verifying the correctness of the operations and in understanding the trade-offs between capacity and visual quality.