**The National University of Computer and Emerging Sciences**

**Islamabad Campus**



# **ASSIGNMENT #1**

# **Digital Image Processing**

**Hamza Asad 22i-1908**

**DIP B**

# **Question #1**

**Introduction**

This report presents an analysis of two different approaches to finding neighbor pixels, in other words : “**Connected component labeling (CCL)**”.

The two methods have been applied to three following tasks :

1. Counting coins

2. Counting cars on the road

3. Number plate recognition

The report provides observations from experiments, compares my custom CCL algorithm and OpenCV’s built-in functions.

Roadmap followed in all three questions :

1. Image Acquisition and Preprocessing :
2. Capture & load image
3. Resizing and Color conversion from BGR to RGB
4. Grayscale & binarization
5. Reduce noise and proper segmentation to ensure accurate answers via Morphing operations
6. CCL algorithm :
7. Custom implementation : Used BFS to traverse through pixels and label them.
8. Used **connectedComponents() & connectedComponentsWithStats()**
9. Visualization :
10. Coloring each label
11. Bounding boxes
12. Saving each image

**Implementation**

1. **Counting coins**

Observations (CUSTOM CCL):

* Accurate when coins are well-separated
* Requires careful thresholding and selection of minimum component size to avoid noise.

Observations (BUILT-IN CCL):

* Similar results
* Easier to implement and less processing time.

|  |  |
| --- | --- |
| **Time taken (Custom CCL )** | 2.732033 seconds |
| **Time taken (Built-in CCL )** | 0.002999 seconds |

1. **Counting cars on the road**

Observations (CUSTOM CCL):

* Required careful assessment of what threshold to go for, since cars were overlapping which made it difficult to create any segmentation hence differentiating between different cars.

Observations (BUILT-IN CCL):

* Consistently accurate even with variations in the image.

|  |  |
| --- | --- |
| **Time taken (Custom CCL )** | 1.613802 seconds |
| **Time taken (Built-in CCL )** | 0.006001 seconds |

Regardless, to regard even the slightest distance between the objects, watershed algorithm was applied to ensure the program differentiates between them .

1. **Number plate Recognition**

The generated number plate image is processed to separate each character (both letters and digits). The custom algorithm then labels each connected character.

Observations (CUSTOM CCL):

* Was having a difficult time trying to single out every pixel of the letter/number.
* Had overall poor labelling and connected pixels since I had to test whether to use 4-nieghbour or 8-neighbour connectivity.
* Furthermore, it took ample time for the DFS algorithm whereas it was quick for the built-in function.
* Needed more additional filtering and testing to separate components accurately.

Observations (BUILT-IN CCL):

* Handled noise better and incredibly.

|  |  |
| --- | --- |
| **Time taken (Custom CCL )** | 2.47 seconds |
| **Time taken (Built-in CCL )** | 0.003117 seconds |

**Why go with DFS for pixel iteration?**

1. Obviously using the **for** loop wouldn’t cut it, it would be very time-consuming.
2. DFS explores all possible ways before backtracking, which ensures complete and isolated labeling of each component.
3. DFS with recursion was easier to implement, hence no additional logic required and was easier to debug for me.

# **Question #2**

**Introduction**

**Steganography** is the technique of concealing information within another medium, such as images, to maintain secrecy. The objective of this task was to implement steganography using the algorithm taught in class, with potential improvements for better efficiency and security. This report documents the approach, implementation, and results of the steganographic techniques applied in the provided code.

The implementation comprises four parts:

1. Encoding and decoding text into an image using the Least Significant Bit (LSB) method.
2. Allowing user input to encode and decode secret messages.
3. Hiding one image inside another using bitwise operations.
4. Superimposing two images while preserving visibility.

**Part 1: Text encoding and decoding in an image**

The image’s pixel values are modified by embedding the binary text into the least significant bits of each pixel.

Then for the decoding procedure, the binary values are converted back into characters. Delimiter was used to extract original msg.

**Part 2: User-Input-Based Text Encoding and Decoding**

This section extends Part 1 by allowing users to input their own image path and message for encoding. The implementation follows these steps:

* The user provides the path to an image.
* The message is input manually.
* The message is encoded into the image.
* The encoded message is extracted and displayed.

**Results:**

The implementation successfully allows users to hide and retrieve their own messages dynamically, making the system flexible and interactive.

**Part 3: Hiding an Image Within Another Image**

This section of the implementation enables concealing an image within another image using bitwise operations.

**Encoding Process:**

* The secret image is resized to match the dimensions of the cover image.
* The last two bits of each pixel in the cover image are cleared.
* The top two bits of the secret image are inserted into the last two bits of the cover image.

**Decoding Process:**

* The last two bits from the encoded image are extracted and shifted back to reconstruct the secret image.

**Results:**

The encoded image appears almost identical to the cover image, ensuring the hidden content remains unnoticeable. Upon extraction, the secret image is successfully retrieved, demonstrating the technique's feasibility.

**Part 4: Superimposing Two Images**

The final part involves merging two images using 4-bit steganography to maintain visibility while embedding information.

**Process:**

* The upper 4 bits of the main image are preserved to retain its visibility.
* The lower 4 bits of the second image are used to hide information.
* The two images are combined using bitwise OR operations.

**Results:**

The superimposed image maintains the clarity of the first image while embedding data from the second image, enabling partial visibility of both images.