K. J. Somaiya College of Engineering, Mumbai-77

| | |
|---|---|
| **Batch: C1** | **Roll No.: 16010122323** |

**Experiment No. 9**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

**Title:** Image compression by lossless technique (Run Length Coding).

**Objective:** To understand image compression by lossless technique (Run Length Coding).

**Expected Outcome of Experiment:**

| CO | Outcome |
|---|---|
| **CO5** | Design and develop applications based on 1-D and 2-D digital signals. |

**Books/ Journals/ Websites referred:**


1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html
3. www.mccormick.northwestern.edu/docs/efirst/matlab.pdf
4. A.Nagoor Kani "Digital Signal Processing", 2nd Edition, TMH Education.

**Pre Lab/ Prior Concepts:**

Variable length code can be used to remove coding redundancy. One of the way to remove the inter pixel redundancy is run length coding. When inter pixel redundancy is removed by using run length coding the mapper transforms the input data in to usually non visual format. This operation is reversible and may or may not reduce directly the amount of data required to represent the image. Run length coding is the example of a mapping that directly results in data compression in the initial stage of overall source encoding process.

**ALGORITHM:**

Compression:

- Read the binary (monochrome) Image.
- Write the runs of the pixel in a text file.
- Use the text file created in a step 2 and writes each run by using 8 bits in output file to create compressed image.
- Use the compress image.
- Expand the runs to create decompressed image.

## Implementation

```
function rle_binary_compression(input_image_path,
compressed_text_path, compressed_bin_path, decompressed_image_path)
   % RLE_BINARY_COMPRESSION - Compress and decompress a binary image
using Run-Length Encoding (RLE)

   % --- Step 1: Read and Validate Input Image ---
   img = imread(input_image_path);
   if size(img, 3) > 1
       error('Input image must be a binary (black & white) image.');
   end

   img = logical(img);  % Ensure it is binary (0s and 1s)
   [rows, cols] = size(img);
   pixels = img(:);  % Flatten image to 1D array
   expected_pixels = rows * cols;

   disp(['Step 1: Image read successfully (', num2str(rows), 'x',
num2str(cols), ').']);
   % --- Step 2: Encode using RLE ---
   runs = encode_rle(pixels);
   write_runs_to_text(runs, compressed_text_path);
   disp(['Step 2: RLE compressed data saved to ',
compressed_text_path]);
   % --- Step 3: Save RLE compressed data in binary format ---
   write_runs_to_binary(runs, compressed_bin_path);
   disp(['Step 3: RLE compressed binary data saved to ',
compressed_bin_path]);
   % --- Step 4: Read and decode the binary file ---
   decoded_pixels = read_runs_from_binary(compressed_bin_path);
   % --- Step 5: Validate Decompressed Data ---
   if length(decoded_pixels) ~= expected_pixels
       error(['Error: Decoded pixels count (',
num2str(length(decoded_pixels)), ') does not match expected (',
num2str(expected_pixels), ').']);
   end
   % --- Step 6: Reshape and Save Decompressed Image ---
   decompressed_img = reshape(decoded_pixels, rows, cols);
   imwrite(decompressed_img, decompressed_image_path);
   disp(['Step 6: Decompressed image saved as ',
decompressed_image_path]);
end
%% --- RLE Encoding ---
function runs = encode_rle(pixels)
   % Performs Run-Length Encoding on a binary pixel array.
   runs = [];
   current_pixel = pixels(1);
   run_count = 1;
   for i = 2:length(pixels)
       if pixels(i) == current_pixel
           run_count = run_count + 1;
       else
 runs = [runs; run_count, current_pixel];
           current_pixel = pixels(i);
           run_count = 1;
       end
   end
```

```
    runs = [runs; run_count, current_pixel]; % Add last run
end
```

```
%% --- Write Runs to Text File ---
function write_runs_to_text(runs, text_path)
   % Saves the RLE runs to a text file
   fileID = fopen(text_path, 'w');
   for i = 1:size(runs, 1)
       fprintf(fileID, '%d %d\n', runs(i, 1), runs(i, 2));
   end
   fclose(fileID);
end
%% --- Write Runs to Binary File ---
function write_runs_to_binary(runs, bin_path)
   % Saves RLE runs in binary format
   fileID = fopen(bin_path, 'wb');
   for i = 1:size(runs, 1)
       fwrite(fileID, runs(i, 1), 'uint16'); % Store run length as
uint16
       fwrite(fileID, runs(i, 2), 'uint8');  % Store pixel value as
uint8 (0 or 1)
   end
   fclose(fileID);
end
%% --- Read Runs from Binary File ---
function decoded_pixels = read_runs_from_binary(bin_path)
   % Reads RLE runs from a binary file and reconstructs the image
   fileID = fopen(bin_path, 'rb');
   decoded_pixels = [];
   while ~feof(fileID)
       run_count = fread(fileID, 1, 'uint16'); % Read 16-bit run count
       if isempty(run_count), break; end
       pixel_value = fread(fileID, 1, 'uint8'); % Read 8-bit pixel
value
       if isempty(pixel_value), break; end
       decoded_pixels = [decoded_pixels; repmat(logical(pixel_value),
run_count, 1)];
   end
   fclose(fileID);
end
```

## Output

Command Window

New to MATLAB? See resources for Getting Started.

```
>> rle_binary_compression('test.bmp', 'compressed_runs.txt', 'compressed_image.bin', 'decompressed_image.png');
Step 1: Image read successfully (128x128).
Step 2: RLE compressed data saved to compressed_runs.txt
Step 3: RLE compressed binary data saved to compressed_image.bin
Step 6: Decompressed image saved as decompressed_image.png
>> |
```

**Import decompressed_image.png**

Variables to be imported:

| Name | Size | Class | Value |
|---|---|---|---|
| decompressed_image | 128×128 | uint8 | 128×128 uint8 |

Preview:

**Conclusion:-**

By eliminating coding and inter pixel redundancy lossless compression is achieved. Here by applying run length coding interpixel redundancy is removed and relative data redundancy is calculated as:

$$R_D = 1 - 1/C_R$$

where $C_R$ is the compression ratio.

**Post Lab Questions**

1) Compare Lossy and lossless compression.

| Feature | Lossy Compression | Lossless Compression |
|---|---|---|
| Definition | Reduces file size by removing some data permanently | Reduces file size without losing any data |
| Data Restoration | Original data cannot be recovered | Original data can be fully restored |
| File Size | Typically results in smaller file sizes | Results in larger file sizes compared to lossy |
| Quality | May reduce quality due to data loss | Maintains original quality |
| Common Formats | JPEG, MP3, MP4 | PNG, GIF, FLAC, ZIP |
| Use Cases | Multimedia where some quality loss is acceptable (e.g., streaming, web images) | Text, software, or images where fidelity is important |
| Speed | Faster due to less data to process | Slightly slower due to need to preserve accuracy |