Digital Image Processing is the technology of applying a computer algorithms to process a digital image. The outcome of this process can be either images or a set of representative properties of the original images. Briefly explain :

a. The concept of an image.

b. The main purpose of digital image processing.

c. Application areas that use digital image processing.

a. An image is a visual representation captured or generated digitally, typically represented as a grid of pixels with color or intensity information.

b. The main purpose of digital image processing is to enhance, manipulate, analyze, and interpret images using algorithms to improve quality, extract information, and support decision-making.

c. Digital image processing finds applications in areas like medicine (diagnosis from medical images), remote sensing (mapping and monitoring), surveillance (object detection and recognition), robotics (vision systems), quality control (automated inspection), forensics (enhancement and identification), entertainment (special effects), and more.

Assuming there are two image points with coordinates (x,y) and (u,v). Calculate the distance measure between these two points using:

a. Euclidean Distance.

b. City-Block Distance.

c. Chessboard Distance.

a. Euclidean Distance:

The Euclidean distance between two points (x, y) and (u, v) is calculated using the formula: Distance = sqrt((u - x)^2 + (v - y)^2).

b. City-Block Distance:

The City-Block distance between two points (x, y) and (u, v) is calculated by summing the absolute differences of their coordinates: Distance = |u - x| + |v - y|.

c. Chessboard Distance:

The Chessboard distance between two points (x, y) and (u, v) is calculated by taking the maximum absolute difference between their coordinates: Distance = max(|u - x|, |v - y|).

Color spaces are a way to represent the color channels present in the image that gives the image that particular hue. There are several different color spaces and each has its own significance. Some of the popular color spaces are RGB (Red, Green, Blue), CMYK (Cyan, Magenta, Yellow, Black), HSV (Hue, Saturation, Value).

a. Discuss in detail about the significant of RGB and CYM/CYMK color space.

b. Explain the concept of HSV color space.

a. RGB (Red, Green, Blue) Color Space:

RGB is used for digital displays and image editing. It combines red, green, and blue to create a wide range of colors. It is significant because it closely aligns with how digital devices produce colors.

CMYK (Cyan, Magenta, Yellow, Black) Color Space:

CMYK is used for printing. It represents colors using cyan, magenta, yellow, and black inks. It is significant because it accurately represents colors for print reproduction.

b. HSV (Hue, Saturation, Value) Color Space:

HSV is a color model based on human perception. It represents colors based on hue, saturation, and value. Hue represents the color itself, saturation is the intensity or purity, and value is the brightness. It is used in computer graphics and image processing for tasks like color segmentation and correction.

A screenshot of a graph

Description automatically generated

To apply histogram equalization, we first need to calculate the cumulative distribution function (CDF) of the original histogram. Then, we map the gray levels to new values based on the CDF. Finally, we calculate the new histogram based on the transformed image.

Here is the calculation for the histogram equalization:

Step 1: Calculate the cumulative distribution function (CDF):

- Calculate the probability mass function (PMF) by dividing the frequency of each gray level by the total frequency.

- Calculate the cumulative distribution function (CDF) by summing up the PMF values.

Gray level - Frequency - PMF - CDF

0 - 123 - 0.0300 - 0.0300

1 - 78 - 0.0190 - 0.0490

2 - 281 - 0.0686 - 0.1176

3 - 417 - 0.1019 - 0.2195

4 - 639 - 0.1562 - 0.3757

5 - 1054 - 0.2576 - 0.6333

6 - 816 - 0.1997 - 0.8330

7 - 688 - 0.1680 - 1.0000

Step 2: Map gray levels to new values based on CDF:

- Multiply each CDF value by the maximum gray level (in this case, 7) and round to the nearest integer to get the new gray levels.

Gray level - New Gray Level

0 - 0

1 - 1

2 - 2

3 - 3

4 - 4

5 - 5

6 - 6

7 - 7

Step 3: Calculate the new histogram:

- Count the frequency of each new gray level in the transformed image.

Gray level - Frequency

0 - 123

1 - 78

2 - 281

3 - 417

4 - 639

5 - 1054

6 - 816

7 - 688

Here are the current and new histogram charts:

Current Histogram:

Gray Level | Frequency

---------------------

0 | 123

1 | 78

2 | 281

3 | 417

4 | 639

5 | 1054

6 | 816

7 | 688

New Histogram:

Gray Level | Frequency

---------------------

0 | 123

1 | 78

2 | 281

3 | 417

4 | 639

5 | 1054

6 | 816

7 | 688

Please note that the new histogram values remain the same as the original histogram since the gray levels were not changed during the histogram equalization process.

Spatial Filtering technique is used directly on pixels of an image. Mask is usually considered to be added in size so that it has a specific center pixel. This mask is moved on the image such that the center of the mask traverses all image pixels.

a. Write a program logic to implement spatial domain averaging filter and to observe its blurring effect on the image without using inbuilt functions

b. Write a program logic to implement spatial domain median filter to remove salt and pepper noise without using inbuilt functions

a. Program logic to implement spatial domain averaging filter (blurring effect):

1. Read the input image.

2. Define the size of the averaging filter mask (e.g., 3x3, 5x5, etc.).

3. Initialize an output image of the same size as the input image.

4. Traverse each pixel (x, y) in the input image:

- Initialize a sum variable to store the cumulative sum of pixel values within the filter mask.

- Traverse each neighbor pixel (i, j) within the filter mask:

- Check if the neighbor pixel is within the image boundaries.

- If yes, add the pixel value at (i, j) to the sum variable.

- Calculate the average value by dividing the sum by the total number of pixels in the filter mask.

- Assign the average value to the corresponding pixel (x, y) in the output image.

5. Save the output image.

6. Display or further process the blurred image as desired.

b. Program logic to implement spatial domain median filter (remove salt and pepper noise):

1. Read the input image.

2. Define the size of the median filter mask (e.g., 3x3, 5x5, etc.).

3. Initialize an output image of the same size as the input image.

4. Traverse each pixel (x, y) in the input image:

- Initialize a list to store the pixel values within the filter mask.

- Traverse each neighbor pixel (i, j) within the filter mask:

- Check if the neighbor pixel is within the image boundaries.

- If yes, add the pixel value at (i, j) to the list.

- Sort the list in ascending order.

- Calculate the median value as the middle element of the sorted list.

- Assign the median value to the corresponding pixel (x, y) in the output image.

5. Save the output image.

6. Display or further process the denoised image as desired.

There are lots of different types of texture descriptors are used to extract features of an image. Local Binary Pattern, also known as LBP, is a simple and grey-scale invariant texture descriptor measure for classification. In LBP, a binary code is generated at each pixel by thresholding it's neighbourhood pixels to either 0 or 1 based on the value of the centre pixel.

a. Describe the rule for finding LBP of an image.

b. Write a program logic to read a greyscale image and convert it into LBP image.

a. Rule for finding LBP of an image:

The Local Binary Pattern (LBP) is calculated for each pixel in the image based on its neighborhood. Here's the rule for finding the LBP of an image pixel:

1. Select a pixel in the image.

2. Define a circular neighborhood around the pixel, consisting of neighboring pixels.

3. Compare the intensity value of each neighbor with the intensity value of the center pixel.

4. If the intensity value of the neighbor is greater than or equal to the intensity value of the center pixel, assign it a value of 1.

5. If the intensity value of the neighbor is less than the intensity value of the center pixel, assign it a value of 0.

6. Combine the binary values of all neighbors in a clockwise or counterclockwise order to form a binary code.

7. Convert the binary code to decimal to obtain the LBP value for the center pixel.

8. Repeat this process for all pixels in the image.

b. Program logic to convert a grayscale image to LBP image:

1. Read the grayscale image.

2. Define the neighborhood size and radius for the LBP calculation.

3. Initialize an output image of the same size as the input image.

4. Traverse each pixel (x, y) in the input image:

- Initialize an empty binary code string.

- Traverse each neighbor pixel (i, j) within the neighborhood:

- Calculate the distance between the center pixel (x, y) and the neighbor pixel (i, j).

- If the distance is less than or equal to the specified radius:

- Compare the intensity value of the neighbor pixel with the center pixel.

- If the neighbor pixel intensity is greater than or equal to the center pixel intensity, append '1' to the binary code string.

- If the neighbor pixel intensity is less than the center pixel intensity, append '0' to the binary code string.

- Convert the binary code string to decimal to obtain the LBP value.

- Assign the LBP value to the corresponding pixel (x, y) in the output image.

5. Save the output LBP image.

6. Display or further process the LBP image as desired.

Close-up of a printed page of a computer chip

Description automatically generated

i) Assumptions for the image processing algorithm:

1) Consistent Lighting Conditions: The image processing algorithm assumes that the lighting conditions on the assembly line, where the PCBs are inspected, remain consistent

2)Proper PCB Alignment: It is assumed that the PCBs are properly aligned and fixed in position during the inspection process.

3)Known Component Types: The algorithm assumes that the types and sizes of the three components to be inspected are known and predefined.

4)Good Image Resolution: The image captured for inspection must have a sufficiently high resolution to capture the fine details of the components accurately.

ii) Image Processing Algorithm for Component Inspection:

Step 1: Image Preprocessing

- Convert the RGB image of the fully planted PCB into grayscale.

- Apply noise reduction techniques (e.g., Gaussian blur, median filter) to remove any noise or artifacts from the image.

- Perform edge detection (e.g., Canny edge detection) to highlight edges of the components, making it easier to identify them.

Step 2: Component Segmentation

- Use thresholding techniques to separate the components from the background and create binary images.

- Perform connected component analysis to detect and label individual component regions in the binary image.

Step 3: Component Size Classification

- Utilize feature extraction methods (e.g., contour analysis, blob detection) to measure the size of each segmented component region.

- Classify the components based on their sizes into the three predefined categories.

Step 4: Template Matching

- Create template images for each component size to act as references for comparison.

- Perform template matching to find instances of each component size in the binary image.

- Implement a similarity threshold to ensure accurate matches.

Step 5: Component Alignment Verification

- Calculate the centroids of the detected component regions to determine their positions relative to the PCB's reference coordinates.

- Compare the centroids with the expected positions based on design specifications.

- Identify any miss-alignments and classify them as defective components.

Step 6: Quality Assessment and Decision Making

- Keep a record of the detected component sizes, positions, and any identified defects.

- Based on the inspection results, categorize the PCB as "Pass" if all components are correctly aligned and within tolerance limits, or "Fail" if any defects are found.

Step 7: Real-time Processing and Speed Optimization

- Optimize the algorithm for efficient real-time processing, ensuring it can inspect PCBs at the required rapid rate.

- Utilize parallel processing or hardware acceleration (e.g., GPU) for faster computation.

A comparison of a few coins

Description automatically generated

<https://www.mathworks.com/help/images/ref/imfill.html>