SIGNAL PROCESSING SYSTEM DESIGN LABORATORY

Experiment 3: Multidimensional signal processing Report

Name: Piyush Dangi
Roll No: 23EE65R08
MTech 1St Year
Spec: Signal processing and Mahcine Learning

Department of Electrical Engineering
Indian Institute of Technology Kharagpur
Session : Autumn 2023

Cartesian to Polar Convesion:

- **AIM:** Functional mapping of fixed point polynomial functions using look-up tables. Dynamic range compression in imaging signals. Scan conversion of multidimensional imaging signals (Polar to Cartesian coordinate and vice versa). Space complexity estimation and code profiling for executing complexity comparison.
- **FUNCTION IMPORTED:**
- $np.mean$ is a function from the popular Python library NumPy that calculates the arithmetic mean, or average, of the elements in a given array or list.
- $Matplotlib$: from matplotlib imported pyplot.
- $time.time$:imported time library to calculate time complexity.
- $Image$: from PIL Image library is imported.
- $np.asarray$ is a NumPy function that converts input data into a NumPy array.
- **OBSERVATION:**
  - $cartesianToPolar2(Img)$:
    1. This function takes a 2D NumPy array Img as input, representing a grayscale image in Cartesian coordinates.
    2. It initializes parameters M and N as the dimensions of the input image, R as the desired number of radial lines, and Theta as the desired number of angular lines.
    3. It creates lookup tables x_lookup and y_lookup to map Cartesian coordinates to polar coordinates.
    4. Then, it iterates through each radial and angular position, performing the mapping based on trigonometric formulas and stores the values in $outImg$, which represents the polar coordinate image.
  - Main Part:
    1. Reads an image named "cart (1).jpg" and converts it into a NumPy array using np.asarray.
    2. It then processes this image by applying the cartesianToPolar2 function, transforming it into a polar coordinate representation stored in the carImgArrayOut.
    3. The polar image is then converted back to a standard grayscale image using the Image.fromarray function and saved as "polarOut.jpg."

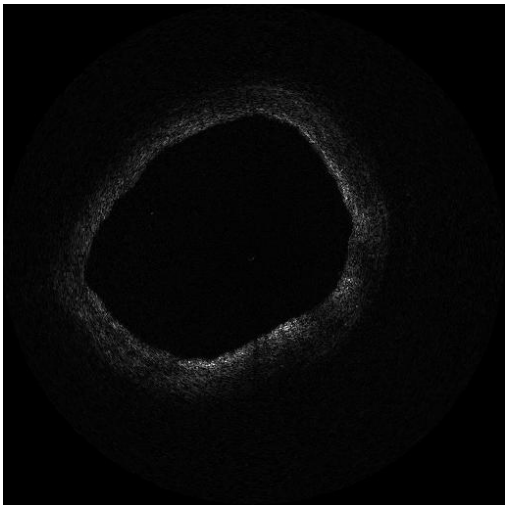

Figure 1: Input cartesian image.

- **OUTPUT:**
  MSE between cartesianToPolar output and original polar: 17.395428856382978

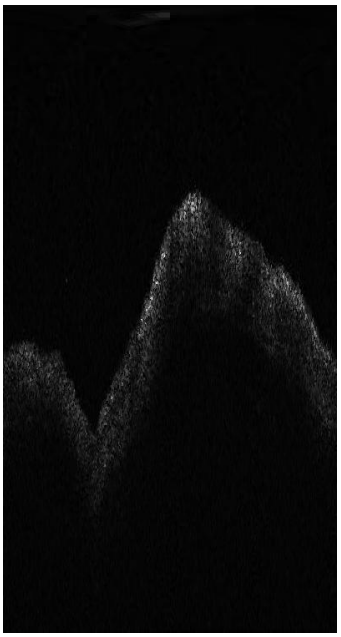

Figure 2: Out polar image.

Polar to Cartesian Convesion:

- **AIM:** Functional mapping of fixed point polynomial functions using look-up tables. Dynamic range compression in imaging signals. Scan conversion of multidimensional imaging signals (Polar to Cartesian coordinate and vice versa). Space complexity estimation and code profiling for executing complexity comparison.
- **FUNCTION IMPORTED:**
- $np.mean$ is a function from the popular Python library NumPy that calculates the arithmetic mean, or average, of the elements in a given array or list.
- $Matplotlib$: from matplotlib imported pyplot.
- $time.time$:imported time library to calculate time complexity.
- $Image$: from PIL Image library is imported.
- $np.asarray$ is a NumPy function that converts input data into a NumPy array.
- **OBSERVATION:**
  - $polarToCartesian(Img)$:
    1. This function takes a 2D NumPy array Img as input, representing a polar coordinate image.
    2. It initializes parameters R and Theta as the dimensions of the polar image and sets M and N as the dimensions of the desired output Cartesian image.
    3. It creates lookup tables m_lookup and n_lookup to map polar coordinates back to Cartesian coordinates.
    4. The function then iterates through each radial and angular position, performing the mapping using trigonometric formulas and stores the values in $outImg$, which represents the Cartesian coordinate image.
  - Main Part:
    1. Reads an image named 'polar.jpg' and converts it into a NumPy array using $np.asarray$.
    2. It processes this polar image by applying the $polarToCartesian$ function, transforming it into a Cartesian coordinate representation stored in $polarArrayOut$.

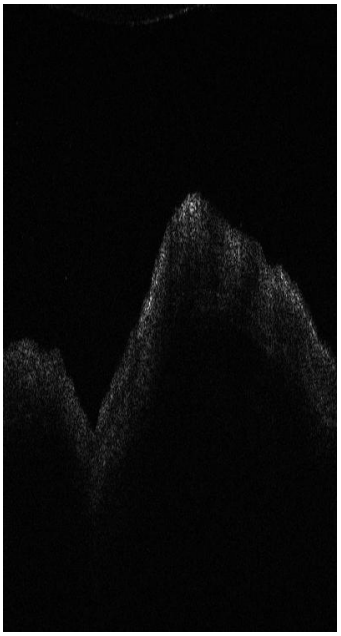

Figure 3: Input polar image.

- **OUTPUT:**
  MSE between polarToCartesian output and original cartesian: 25.91533660888672

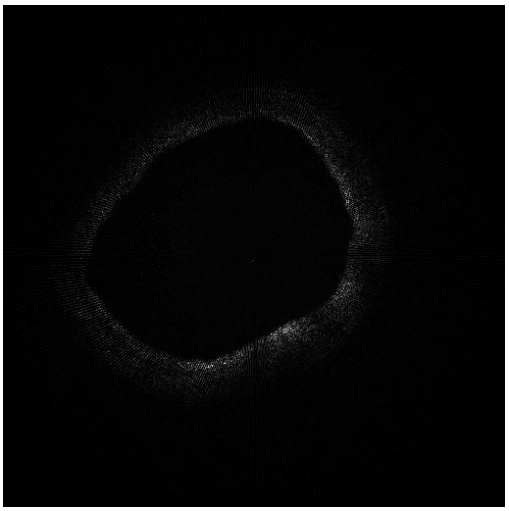

Figure 4: Out cartesian image.

Image Compression:

- **AIM:** Dynamic range compression on the gray scale image.
- **FUNCTION IMPORTED:**
- 1.from PIL import Image: This line imports the Image module from the Python Imaging Library (PIL), which is used for working with images.
- **OBSERVATION:**
  - Image Opening and Initialization:
    Begins by opening the image file ”cart (1).jpg” using the Python Imaging Library (PIL) as image.
  - Parameter Initialization:
    Three parameters are defined: a, b, and c. These parameters control the transformation and affect the contrast enhancement process. a and b are used in the logarithmic transformation, while c helps prevent taking the logarithm of zero or negative values.
  - Image Size and Arrays:
    The dimensions of the input image are determined and stored in rowImg and columnImg.
    An empty 2D NumPy array g is created to store the transformed pixel values.
  - Logarithmic Transformation:
    Loops through each pixel of the input image using nested loops.
    For each pixel, it calculates a new pixel value (temp) based on a logarithmic transformation of the original pixel’s intensity.
    The transformation formula applies the parameters a, b, and c to enhance contrast in the image.
  - Data Type Conversion:
    The transformed values in the array g are converted to unsigned 8-bit integers (uint8) to ensure they are within the valid pixel value range (0-255).
  - Image Modification:
    we use the transformed pixel values from g to update the original image, pixel by pixel, effectively enhancing the contrast.
  - Image Saving:
    The modified image is saved as ”cartComp.jpg.”

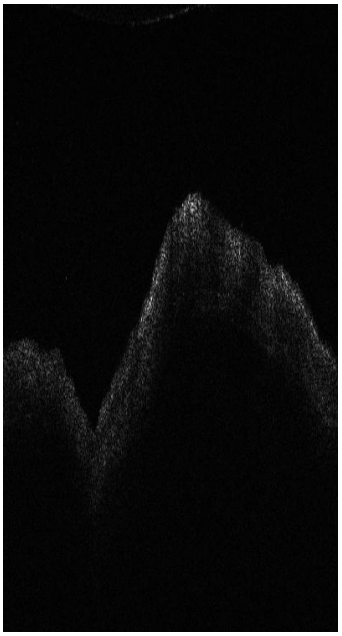

Figure 5:Input image

- **OUTPUT:**

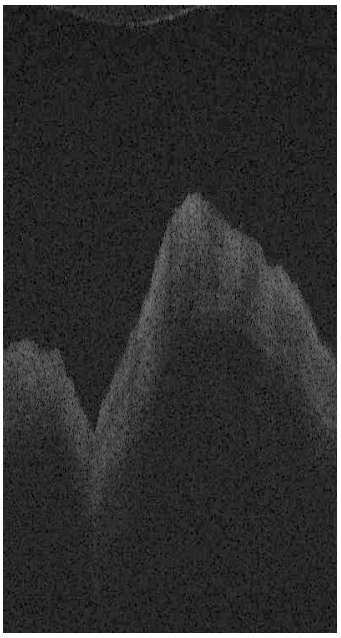

Figure 6:dynamic compression