**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Mohsin Kamal **Dated:** 26/10/2023

**Semester:** 7th **Section:** BEE 12C

**CS-477 Computer Vision**

Lab 7: Image Interpolation

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Table of Contents**

[2 Image Interpolation 3](#_Toc149229427)

[2.1 Introduction 3](#_Toc149229428)

[2.2 Objectives 3](#_Toc149229429)

[2.3 Software 3](#_Toc149229430)

[3 Lab Tasks: Image Resolution 4](#_Toc149229431)

[4 Conclusion 9](#_Toc149229432)

# Image Interpolation

## Introduction

The objective in this lab is to introduce digital images as a second useful type of signal. We will show how the A-to-D sampling and the D-to-A reconstruction processes are carried out for digital images. We will show a commonly used method of image zooming (reconstruction) that gives “poor” results.

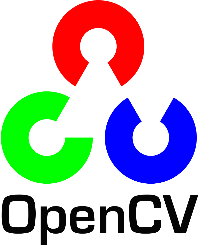
## Objectives

The objective of this lab is:

* Familiarization with digital images.
* Working with images in Python.
* Sampling of images.
* Familiarization with reconstruction of images.

## Software

OpenCV is a library that focuses on image processing and computer vision. An image is an array of colored squares called pixels. Each pixel has a certain location in the array and color values in BGR format. By referring to the array indices, the individual pixels or a range of pixels can be accessed and modified. OpenCV provides many functions for resizing, rotating, and placing objects in images. Rotation involves computing a 2-D rotation matrix which is applied for the transformation of the image.



# Lab Tasks: Image Resolution

1. Reduce the resolution of **7\_1.asc** by a factor of 4 in both horizontal and vertical dimensions (e.g., if the original image is 400 by 400, then result shall be 100 by 100) to create a decimated image using two different methods:

To read in an “.asc”, : X=np.loadtxt(‘3\_1.asc’).

1. Keep one pixel out of every 4x4 pixel area. Display the resulting image Y1.

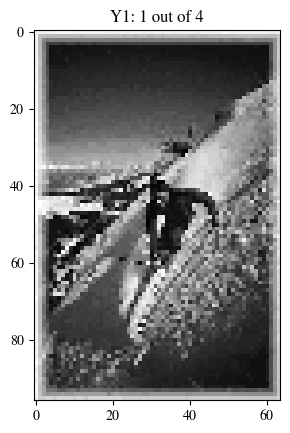
X = np.loadtxt("7\_1.asc")

Y1 = X[::4, ::4]

plt.imshow(Y1, *cmap*="gray")

plt.title("Y1: 1 out of 4")

plt.show()



1. Replace every 4x4 pixel area in **7\_1.asc** by the average value of the pixel values in that region. Display the resulting image Y2.

Y2 = np.zeros([X.shape[0] // 4, X.shape[1] // 4])

for i in range(Y2.shape[0]):

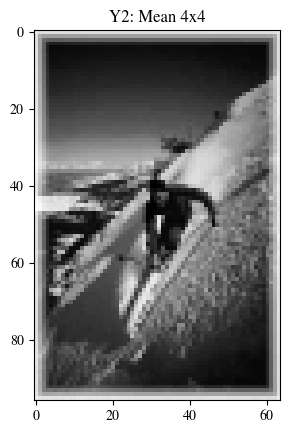
    for j in range(Y2.shape[1]):

        Y2[i, j] = np.mean(X[4 \* i : 4 \* (i + 1), 4 \* j : 4 \* (j + 1)])

plt.imshow(Y2, *cmap*="gray")

plt.title("Y2: Mean 4x4")

plt.show()



1. Enlarge Image Y1 by a factor of 4 in both horizontal and vertical dimensions (e.g., from 100 by 100 to 400 by 400) using:
2. Pixel repeating (zero order hold).

Y1\_b1 = np.zeros([X.shape[0], X.shape[1]])

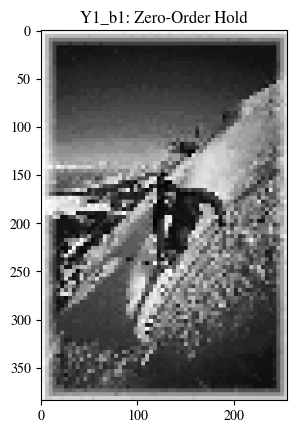
for i in range(Y1\_b1.shape[0]):

    for j in range(Y1\_b1.shape[1]):

        Y1\_b1[i, j] = Y1[i // 4, j // 4]

plt.imshow(Y1\_b1, *cmap*="gray")

plt.show()



1. Bilinear interpolation (do not use built-in interpolation function*, use your own code*).

*def* bilinear\_interpolation(*img*, *x*, *y*, *x1*, *y1*, *x2*, *y2*):

    if x1 == x2 or y1 == y2:

        return img[y1, x1]

    x = min(max(x, x1), x2)

    y = min(max(y, y1), y2)

    R1 = (x2 - x) / (x2 - x1) \* img[y1, x1] + (x - x1) / (x2 - x1) \* img[y1, x2]

    R2 = (x2 - x) / (x2 - x1) \* img[y2, x1] + (x - x1) / (x2 - x1) \* img[y2, x2]

    return (y2 - y) / (y2 - y1) \* R1 + (y - y1) / (y2 - y1) \* R2

Y1\_b2 = np.zeros([X.shape[0], X.shape[1]])

Y1\_b2[::4, ::4] = Y1

for i in range(Y1\_b2.shape[0]):

    for j in range(Y1\_b2.shape[1]):

        x, y = j, i  *# Coordinates in Y1\_b2*

        x1, y1 = max(0, x - 1), max(0, y - 1)  *# Top-left*

        x2, y2 = min(Y1\_b2.shape[1] - 1, x + 1), min(

            Y1\_b2.shape[0] - 1, y + 1

        )  *# Bottom-right*

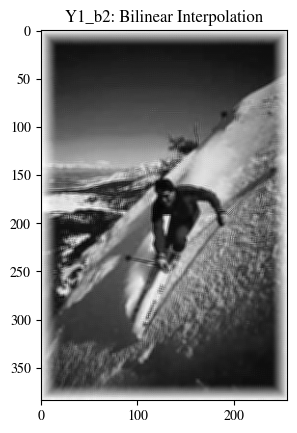
        Y1\_b2[i, j] = bilinear\_interpolation(X, x, y, x1, y1, x2, y2)

*# Display the interpolated image*

plt.imshow(Y1\_b2, *cmap*="gray")

plt.title("Y1\_b2: Bilinear Interpolation")

plt.show()



Keep the result images from (b.i) and (b.ii) the same size as **7\_1.asc,** compare the images. Compare the quality of the linear interpolation result to the zero-order hold result. Point out regions where they differ and try to justify this difference by estimating the local frequency content. In other words, look for regions of “low-frequency” content and “high-frequency” content and see how the interpolation quality is dependent on this factor. A couple of questions to think about: Are edges low frequency or high frequency features? Is the background a low frequency or high frequency feature?

**Answer:** Linear interpolation results in a much smoother image than the jagged results that zero-order hold produces. They differ highly in the regions where transitions, i.e., edges, occur. Let the following region be the basis for observing the frequency content:

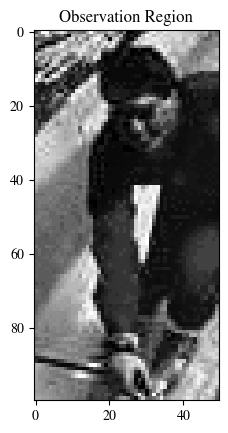
r1, r2 = 150, 250

c1, c2 = 100, 150

plt.imshow(X[r1:r2, c1:c2], *cmap*="gray")

plt.title("Observation Region")

plt.show()



fig, axs = plt.subplots(3, 1, *figsize*=(6, 12))

*# Display the original image region as a signal*

axs[0].plot(X[r1:r2, c1:c2].flatten())

axs[0].set\_title("Original Image")

*# Display the zero-order hold image region as a signal*

axs[1].plot(Y1\_b1[r1:r2, c1:c2].flatten())

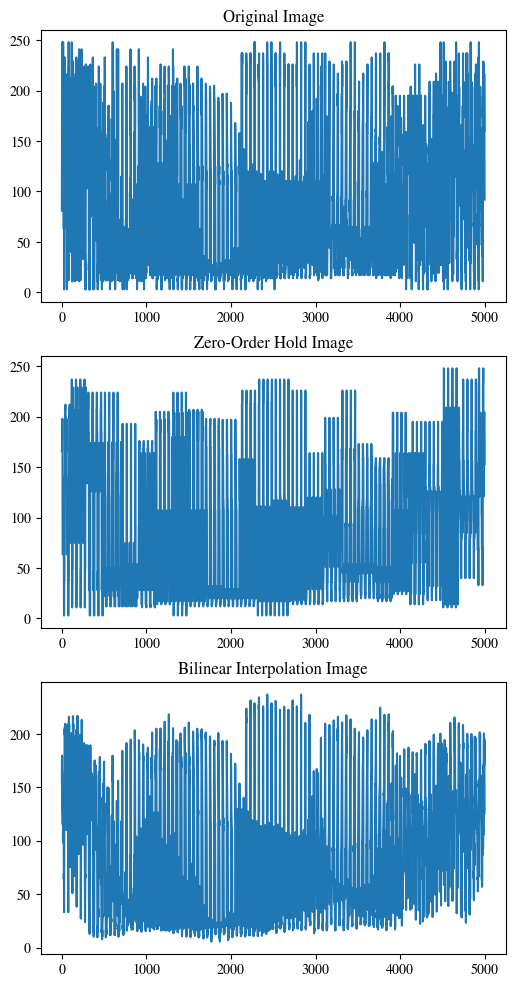
axs[1].set\_title("Zero-Order Hold Image")

*# Display the bilinear interpolation image region as a signal*

axs[2].plot(Y1\_b2[r1:r2, c1:c2].flatten())

axs[2].set\_title("Bilinear Interpolation Image")

plt.show()



Here, we can observe that bilinear interpolation preserves around the same transition profile as the original image, whereas zero-order gives rise to gaps. Moreover, edges are high frequency features as sharp transitions are observed in the perceived signal of the image. The background, on the other hand, is a low frequency feature, especially if it’s smooth. We conclude that bilinear interpolation preserves the quality of the edges much better than zero-order hold. However, it is still not perfect, as the image is smoothed, compared to higher order interpolations.

# Conclusion

In this lab, we performed several image processing tasks on the "7\_1.asc" image. The quality of the interpolation method depends on the local frequency content of the image. High-frequency features, such as edges, are better preserved with bilinear interpolation, while low-frequency features, like smooth backgrounds, may not show a significant difference between the two methods.