**ATILIM UNIVERSITY**

**CMPE464**

**Digital Image Processing**

**HOMEWORK #03**

Müslüm Barış KORKMAZER

19244710045

Ankara,

Spring, 2024

# Introduction

In digital image processing, noise reduction is a crucial step to enhance the quality of images for various applications such as computer vision, medical imaging, and photography. Image enhancement techniques play a significant role in improving the visual perception of noisy images by reducing unwanted artifacts and enhancing relevant details. In this project, we explore different image enhancement methods to denoise a set of noisy images and analyze their effectiveness.

# Implementation Overview

**NoisyImage Class:**

This class encapsulates the functionalities related to each noisy image, including reading the image, converting it to grayscale, splitting it into color channels, and calculating histograms. Additionally, it selects a specific part of the image for further analysis.

**Main Function:**

The main function initializes a list of NoisyImage objects, each corresponding to one of the provided noisy images. It then proceeds to generate histograms for the entire image and the selected part, saving them as PNG files. Afterward, it applies different denoising techniques based on the nature of noise present in each image.

**Image Denoising Techniques:**

Two distinct denoising techniques are employed based on the type of noise present in the images:

* For images affected by Rayleigh noise (noisy1, noisy3, and noisy5), the Fast Non-Local Means Denoising algorithm is utilized.
* For images affected by Gaussian noise (noisy2 and noisy4), Gaussian Blur filtering is applied.

**Result Analysis:**

The denoised images are saved for each noisy input image, enabling a comparative analysis of the effectiveness of the chosen denoising techniques.

# Results and Analysis

1. Rayleigh Noise Denoising:

* The Fast Non-Local Means Denoising technique effectively reduces the Rayleigh noise present in images noisy1, noisy3, and noisy5.
* taşıt, araç, kara taşıtı, araba, dış mekan içeren bir resim

  Açıklama otomatik olarak oluşturuldumemeli, köpek, köpek cinsi, köpek yavrusu içeren bir resim

  Açıklama otomatik olarak oluşturulduThe denoised images exhibit smoother regions and enhanced details, leading to improved visual quality compared to the original noisy images.

doğa, dağ, siyah beyaz, zirve içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. Gaussian Noise Denoising:

* Gaussian Blur filtering is applied to images noisy2 and noisy4 affected by Gaussian noise.
* dış mekan, bulut, gökyüzü, yer içeren bir resim

  Açıklama otomatik olarak oluşturulduinsan yüzü, giyim, portre, kişi, şahıs içeren bir resim

  Açıklama otomatik olarak oluşturulduThe denoised images show reduced blurring and suppression of Gaussian noise, resulting in clearer edges and improved perceptual quality.

1. Overall Evaluation:

* The chosen denoising techniques successfully mitigate the effects of noise in the provided images, enhancing their suitability for visual perception and further analysis.
* However, it's important to note that the effectiveness of denoising may vary depending on the specific characteristics of noise and image content.
* Further experimentation and evaluation with alternative denoising methods could provide insights into optimizing the quality of denoised images for specific applications.

# Code

## main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <opencv2/opencv.hpp>

#include <matplot/matplot.h>

*// Necessary for this project*

class NoisyImage

{

public:

    std::string image\_ext;

    std::string image\_name;

    int HISTOGRAM\_SIZE = 256;

    std::array<float, 2> RANGE[2] = {0, 255};

    cv::Mat denoised\_image;

    cv::Mat image;

    cv::Mat image\_gray;

    cv::Mat image\_histogram;

    cv::Mat image\_bgr\_channels[3];

    cv::Rect selected\_part;

    cv::Mat selected\_part\_image;

    cv::Mat selected\_part\_image\_bgr\_channels[3];

    cv::Mat selected\_part\_image\_histogram;

    NoisyImage() {}

    NoisyImage(std::string \_image\_path, std::string \_image\_name, cv::Rect \_selected\_part) : selected\_part(\_selected\_part)

    {

*// Get the image name and extension*

*this*->image\_name = \_image\_name.substr(0, \_image\_name.find("."));

*this*->image\_ext = \_image\_name.substr(\_image\_name.find("."), \_image\_name.size());

*// Read the image*

*this*->image = cv::imread(\_image\_path + *this*->image\_name + *this*->image\_ext, cv::IMREAD\_COLOR);

*// Convert image to gray*

        cv::cvtColor(*this*->image, *this*->image\_gray, cv::COLOR\_BGR2GRAY);

*// Split image into bgr channels*

        cv::split(*this*->image\_gray, *this*->image\_bgr\_channels);

*// Get the selected part of the image*

*this*->selected\_part\_image = *this*->image\_gray(*this*->selected\_part);

*// Split selected part image into bgr channels*

        cv::split(*this*->selected\_part\_image, *this*->selected\_part\_image\_bgr\_channels);

*// Calculate histogram of image*

        std::array<const float \*, 1> histRange = {*this*->RANGE->data()};

        cv::calcHist(&*this*->image\_bgr\_channels[0], 1, 0, cv::Mat(), *this*->image\_histogram, 1, &*this*->HISTOGRAM\_SIZE, histRange.data());

*// Calculate histogram of selected part of image*

        cv::calcHist(&*this*->selected\_part\_image\_bgr\_channels[0], 1, 0, cv::Mat(), *this*->selected\_part\_image\_histogram, 1, &*this*->HISTOGRAM\_SIZE, histRange.data());

    }

*// Get X coordinates for histogram*

    std::vector<int> getPlotXCoordinates() const

    {

*// Create x coordinates*

        std::vector<int> x(256);

        for (int i = 0; i < 256; ++i)

        {

            x[i] = i;

        }

        return x;

    }

*// Get Y coordinates for histogram*

    std::vector<double> getPlotYCoordinates(cv::Mat hist) const

    {

*// Create plot data*

        std::vector<double> y;

        for (int i = 0; i < 256; ++i)

        {

            y.push\_back(hist.at<float>(i));

        }

        return y;

    }

    ~NoisyImage() {}

};

*// int main(int argc, char const \*argv[])*

int main()

{

    using namespace matplot;

    const std::string IMAGES\_PATH = "/home/babico/Projects/CMPE464-Github/HW3/images/";

    std::vector<NoisyImage> noisy\_images = {

        NoisyImage(IMAGES\_PATH, "noisy1.jpg", cv::Rect(325, 100, 450 - 325, 300 - 100)),

        NoisyImage(IMAGES\_PATH, "noisy2.jpg", cv::Rect(0, 0, 115, 415)),

        NoisyImage(IMAGES\_PATH, "noisy3.jpg", cv::Rect(0, 0, 112, 48)),

        NoisyImage(IMAGES\_PATH, "noisy4.jpg", cv::Rect(0, 450, 200, 50)),

        NoisyImage(IMAGES\_PATH, "noisy5.jpg", cv::Rect(0, 0, 224, 78))};

*// Create a histogram for each image*

    for (auto &noisy\_image : noisy\_images)

    {

*// Matplot++ instance*

        auto plotinstance = figure(true);

*// Create plot for histogram*

        plot(noisy\_image.getPlotXCoordinates(), noisy\_image.getPlotYCoordinates(noisy\_image.image\_histogram));

        plotinstance->save(IMAGES\_PATH + "histogram\_" + noisy\_image.image\_name + ".png");

*// Save selected part of image*

        cv::imwrite(IMAGES\_PATH + "selected\_part\_" + noisy\_image.image\_name + ".png", noisy\_image.selected\_part\_image);

*// Create plot for histogram of selected part of image*

        plot(noisy\_image.getPlotXCoordinates(), noisy\_image.getPlotYCoordinates(noisy\_image.selected\_part\_image\_histogram));

        plotinstance->save(IMAGES\_PATH + "selected\_part\_histogram\_" + noisy\_image.image\_name + ".png");

    }

    return 0;

}

# Git Repository

[babico/CMPE464: Atılım University CMPE464 Digital Image Processing (github.com)](https://github.com/babico/CMPE464)