**ATILIM UNIVERSITY**

**CMPE464**

**Digital Image Processing**

**HOMEWORK #04**

Müslüm Barış KORKMAZER

19244710045

Ankara,

Spring, 2024

# Introduction

The code provided is a C++ implementation for processing noisy images using OpenCV and Matplot++. It aims to read a collection of noisy images, analyze them by generating histograms, and then attempt to denoise these images using combinations of bilateral and Gaussian filters. This code bases its operations on predefined coordinates to focus on specific parts of the images, which is typically necessary when dealing with localized noise or when wanting to highlight particular features or corrections.

# Implementation Overview

Classes and Libraries:

* NoisyImage Class: This custom class encapsulates operations related to handling noisy images. It includes loading the image, converting it to grayscale, extracting specific parts of the image, and computing histograms for both the full image and selected parts.
* OpenCV Library: Used for image manipulation tasks, such as reading images, converting color spaces, filtering, and saving modified images.
* Matplot++ Library: Utilized for plotting histograms which help in analyzing the distribution of pixel intensities across images.

Key Operations:

* Image Loading: Each image is loaded into a cv::Mat object. The images are accessed using a defined path and their respective names.
* Grayscale Conversion and Channel Splitting: Images are converted from color to grayscale since the operations and histograms are primarily interested in intensity values rather than color data.
* Extract and Analyze Selected Image Parts: Specific parts of images are extracted based on predefined rectangles (cv::Rect). These parts are crucial for focused analysis and targeted denoising operations.
* Histogram Computation: Histograms for the overall image and its specific parts assist in understanding the distribution of noise and pixel values. This is helpful in deciding the parameters for the noise reduction filters.
* Denoising Operations: The bilateral filter is used to reduce unwanted noise while preserving edges, followed by a Gaussian blur to further smooth the image. These steps are tailored based on experimental observations per image.

# Results and Analysis

# **Noise Analysis**

# Histograms generated via Matplot++ clearly depict the pixel intensity distributions allowing for insights into variations and potential noise patterns.

# Initial observation suggests that all images might be suffering from pattern noise based on the histogram characteristics and visual inspection.

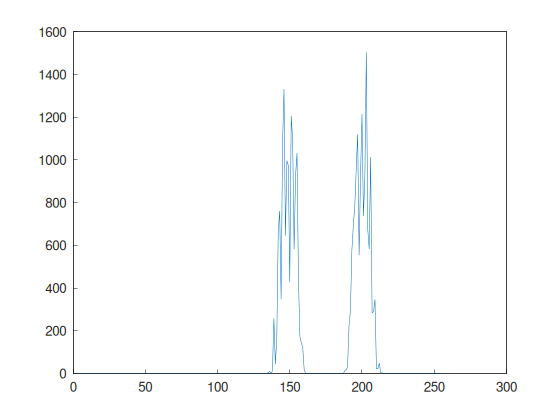


Figure 1: selected\_part\_histogram\_noisy1.png

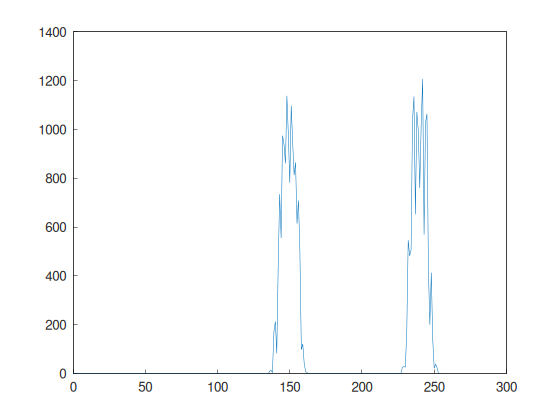


Figure 2: selected\_part\_histogram\_noisy2.png

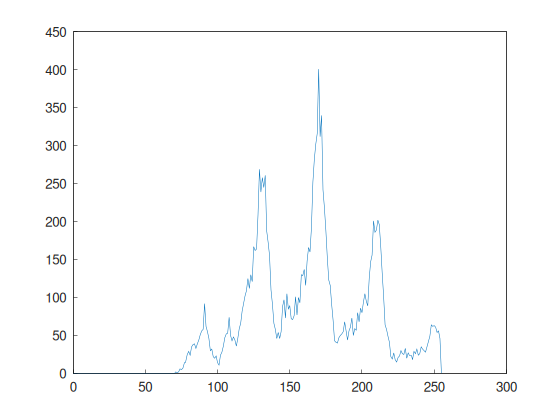


Figure 3: selected\_part\_histogram\_noisy3.png

# **Denoising Results**

# For noisy1.jpg and noisy2.jpg, the combination of bilateral and Gaussian filtering proved effective. The chosen parameters (filter size and sigma values of the Gaussian blur) were based on iterative testing to find a balance between noise reduction and detail preservation.

# However, noisy3.jpg did not show improvements with the same method, suggesting a different type of noise or more complex noise characteristics that might require alternative approaches such as non-local means denoising, Fourier transform methods, or advanced machine learning models.



Figure 5: denoised\_bilateral\_noisy1.png



Figure 4: denoised\_bilateral\_noisy2.png



Figure 6: denoised\_bilateral\_noisy3.png

# 

# 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 = {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() = default;

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

        : image\_name(\_image\_name.substr(0, \_image\_name.find("."))),

          selected\_part(\_selected\_part)

    {

*// Get image extension*

*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() = default;

};

const std::string IMAGES\_PATH = "/home/babico/Projects/CMPE464-Github/HW4/images/";

using namespace matplot;

int main()

{

*// Create noisy images with selected parts*

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

        NoisyImage(IMAGES\_PATH, "noisy1.jpg", cv::Rect(315, 0, 195, 140)),

        NoisyImage(IMAGES\_PATH, "noisy2.jpg", cv::Rect(315, 0, 195, 140)),

        NoisyImage(IMAGES\_PATH, "noisy3.jpg", cv::Rect(0, 95, 95, 170)),

    };

*// 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");

    }

*// noisy1 image noise type is pattern noise*

*// Fix with bilateral x gaussian filter*

    cv::bilateralFilter(noisy\_images[0].image\_gray, noisy\_images[0].denoised\_image, 8, 256, 256);

    cv::GaussianBlur(noisy\_images[0].denoised\_image, noisy\_images[0].denoised\_image, cv::Size(5, 5), 2);

    cv::imwrite(IMAGES\_PATH + "denoised\_bilateral\_" + noisy\_images[0].image\_name + ".png", noisy\_images[0].denoised\_image);

*// noisy2 image noise type is pattern noise*

    cv::bilateralFilter(noisy\_images[1].image\_gray, noisy\_images[1].denoised\_image, 8, 256, 256);

    cv::GaussianBlur(noisy\_images[1].denoised\_image, noisy\_images[1].denoised\_image, cv::Size(3, 3), 2);

    cv::imwrite(IMAGES\_PATH + "denoised\_bilateral\_" + noisy\_images[1].image\_name + ".png", noisy\_images[1].denoised\_image);

*// noisy3 image noise type is pattern noise*

    cv::bilateralFilter(noisy\_images[2].image\_gray, noisy\_images[2].denoised\_image, 8, 256, 256);

    cv::GaussianBlur(noisy\_images[2].denoised\_image, noisy\_images[2].denoised\_image, cv::Size(3, 3), 4);

    cv::imwrite(IMAGES\_PATH + "denoised\_bilateral\_" + noisy\_images[2].image\_name + ".png", noisy\_images[2].denoised\_image);

    return 0;

}

# Git Repository

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