**ATILIM UNIVERSITY**

**CMPE464**

**Digital Image Processing**

**HOMEWORK #1**

Müslüm Barış KORKMAZER

19244710045

Ankara,

Spring, 2024

# Introduction

In this homework assignment, we were tasked with performing various image processing tasks on a given image. The primary objectives included resizing the image, converting it to grayscale, generating binary images with different thresholds, segmenting the image into quadrants based on color channels and creating symmetric versions of the input image along both the vertical and horizontal axes.

# Implementation Overview

The following tasks were carried out using OpenCV library in C++:

1. Image Resizing: The original image was resized to 256x256 pixels.
2. Grayscale Conversion: The resized image was converted to grayscale.
3. Binary Conversion: Image was threshold using different thresholds, resulting in multiple binary images.
4. Quadrant Segmentation: The original image was divided into four quadrants where each quadrant represents a different color channel (RGB).
5. Symmetry Generation: Symmetric versions of the input image were created along both vertical and horizontal axes.

# Results and Analysis

## Original Image

sebze, biber, üretmek, mahsul, doğal gıdalar içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 1: Original Image

## Image Resizing

sebze, biber, üretmek, mahsul, doğal gıdalar içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 2: 256x256 Resized Image

## Grayscale Conversion

Converting the original image to grayscale reduces the color information to a single intensity value per pixel. Grayscale images are often used for various image processing tasks such as edge detection as they simplify the image while retaining essential features.



Picture 3: Grayscale Image

## Quadrant Segmentation

Segmenting the image into quadrants based on color channels allows us to visualize the contribution of each channel to overall image. Original image occupies the top-left quadrant, while the top-right, bottom-left and bottom-right quadrants represent the red, green and blue channels respectively. By isolating each channel we can observe the distribution and intensity of each color within the image.

iç mekan, oyuncak, tatlı biber, biber içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 4: Combined Colored Image

## Symmetry Generation

Creating symmetric versions of the input image along the vertical and horizontal axes provides insights into the bilateral symmetry of the image. By flipping the image along these axes we can observe how the features of the image mirror each other, which can be useful for tasks such as image alignment and analysis.

doğal gıdalar, üretmek, mahsul, işlenmemiş gıda, diyet yemeği içeren bir resim

Açıklama otomatik olarak oluşturuldubiber, sebze, üretmek, mahsul, dolmalık biber ve acı biber içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 5: Vertical midline symmetric Picture 6: Horizantal midline symmetric

## Binary Conversion

Thresholding the grayscale image at different levels produces binary images where pixel values are either 0 or 255 depending on whether they fall below or above the threshold. By varying the threshold, we can highlight different features of the image based on their intensity levels.

harita, metin içeren bir resim

Açıklama otomatik olarak orta güvenilirlik düzeyiyle oluşturuldu

Picture 7: Binary Threshold Value 0

hafif, çizgi film, sanat, cadılar bayramı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 8: Binary Threshold Value 155

grafik, kırmızı, ekran görüntüsü, karmin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 9: Binary Threshold Value 255

siyah, karanlık içeren bir resim

Açıklama otomatik olarak oluşturuldu

Picture 10: Binary Threshold Value 256

# Code

## main.cpp

#include <string>

#include <vector>

#include <iostream>

#include <sys/stat.h>

#include <opencv2/opencv.hpp>

const int THRESHOLD\_MAX = 1024;

const int GRAYSCALE\_MAX = 255;

const std::string IMAGES\_PATH = "../images/";

const cv::Size size256x256 = cv::Size(256, 256);

const std::string BINARY\_IMAGES\_PATH = "../images/binary/";

const cv::Mat zeroMatix = cv::Mat::zeros(size256x256, CV\_8U); *// Zero matrix*

const std::vector<cv::Rect> pieces{ *// 4 piece of an image*

    cv::Rect(0, 0, 128, 128), *// top left // (x, y, width, height)*

    cv::Rect(128, 0, 128, 128), *// top right // (x, y, width, height)*

    cv::Rect(0, 128, 128, 128), *// bottom left // (x, y, width, height)*

    cv::Rect(128, 128, 128, 128), *// bottom right // (x, y, width, height)*

};

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

{

*// Read peppers.png image file*

    cv::Mat img = cv::imread(IMAGES\_PATH + "\_peppers.png", cv::IMREAD\_COLOR);

*// Check is image loaded*

    if (img.empty())

    {

        std::cerr << "Image load failed!" << std::endl;

        return -1;

    }

*// Print success message and image size*

    std::cout << "Image loaded successfully!" << std::endl;

    std::cout << "Image size: " << img.size() << std::endl;

*// Resize image to 256x256*

    cv::resize(img, img, size256x256);

*// Save resized image to peppers\_resized.png*

    cv::imwrite(IMAGES\_PATH + "peppers\_256x256.png", img);

*// Print success message and image size*

    std::cout << "Resized image saved successfully!" << std::endl;

    std::cout << "Image size: " << img.size() << std::endl;

*// Convert image to grayscale*

    cv::Mat gray;

    cv::cvtColor(img, gray, cv::COLOR\_BGR2GRAY);

*// Save grayscale image to peppers\_grayscale.png*

    cv::imwrite(IMAGES\_PATH + "peppers\_grayscale.png", gray);

*// Print success message*

    std::cout << "Grayscale image saved successfully!" << std::endl;

*// Define matrix*

    cv::Mat binaries[THRESHOLD\_MAX];

*// Crate folder for binary convertion*

    mkdir(BINARY\_IMAGES\_PATH.c\_str(), S\_IRWXU | S\_IRWXG | S\_IROTH | S\_IXOTH);

    for (size\_t i = 0; i < THRESHOLD\_MAX; i++)

    {

*// Convert image to binary using different thresholds*

        cv::threshold(img, binaries[i], i, THRESHOLD\_MAX, cv::THRESH\_BINARY);

*// Save binary converted image*

        cv::imwrite(BINARY\_IMAGES\_PATH + "peppers\_threshold\_" + std::to\_string(i) + ".png", binaries[i]);

    }

*// Print success message*

    std::cout << "Threshold images saved successfully!" << std::endl;

*// Get color channels of image*

    cv::Mat channels[3]; *// color order Blue Green Red*

    cv::split(img, channels);

    cv::Mat results[4]; *// image order onlyBlue onlyGreen onlyRed*

    results[0] = img;

    std::vector<cv::Mat> colors[3]{

*// color order onlyBlue onlyGreen onlyRed*

        std::vector<cv::Mat>{zeroMatix, zeroMatix, channels[2]},

        std::vector<cv::Mat>{zeroMatix, channels[1], zeroMatix},

        std::vector<cv::Mat>{channels[0], zeroMatix, zeroMatix},

    };

    for (size\_t i = 0; i < 3; i++)

    {

*// Convert vectors to image*

        cv::merge(colors[i], results[i + 1]);

*// Save the result*

        cv::imwrite(IMAGES\_PATH + "peppers\_color\_filtered\_" + std::to\_string(i) + "\_colors.png", results[i + 1]);

    }

*// Print success message*

    std::cout << "Color filtered image saved successfully!" << std::endl;

*// Extract each images squares*

    cv::Mat img\_pieces[4];

    for (size\_t i = 0; i < pieces.size(); i++)

    {

        img\_pieces[i] = results[i](pieces[i]);

    }

*// Create a blank image to merge the ROIs onto*

    cv::Mat merged\_image(size256x256, CV\_8UC3);

*// Place each piece onto the appropriate position in the merged image*

    for (size\_t i = 0; i < pieces.size(); i++)

    {

        img\_pieces[i].copyTo(merged\_image(pieces[i]));

    }

*// Save merged image*

    cv::imwrite(IMAGES\_PATH + "peppers\_color\_merged.png", merged\_image);

*// Print success message*

    std::cout << "Color filtered merged image saved successfully!" << std::endl;

*// Create a new image with the same dimensions as the input image*

    cv::Mat verticalSymmetric(size256x256, img.type());

    cv::Mat horizantalSymmetric(size256x256, img.type());

*// Get left half of the image*

    cv::Mat verticalLeftHalf = img(cv::Rect(0, 0, size256x256.width / 2, size256x256.height));

*// Get upper half of the image*

    cv::Mat horizontalUpperHalf = img(cv::Rect(0, 0, size256x256.width, size256x256.height / 2));

*// Flip the left half of the image vertically directly to the output image*

    cv::flip(verticalLeftHalf, verticalSymmetric(cv::Rect(size256x256.width / 2, 0, size256x256.width / 2, size256x256.height)), 1);

*// Copy left half to output image*

    verticalLeftHalf.copyTo(verticalSymmetric(cv::Rect(0, 0, size256x256.width / 2, size256x256.height)));

*// Flip the upper half of the image horizontally to the output image*

    cv::flip(horizontalUpperHalf, horizantalSymmetric(cv::Rect(0, size256x256.height / 2, size256x256.width, size256x256.height / 2)), 0);

*// Copy upper half to output image*

    horizontalUpperHalf.copyTo(horizantalSymmetric(cv::Rect(0, 0, size256x256.width, size256x256.height / 2)));

*// Save symmetric images*

    cv::imwrite(IMAGES\_PATH + "peppers\_vertical.png", verticalSymmetric);

    cv::imwrite(IMAGES\_PATH + "peppers\_horizontal.png", horizantalSymmetric);

*// Print success message*

    std::cout << "Symmetric images saved successfully!" << std::endl;

    return 0;

}

## Git Repository

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