Comparison of Blurring/Smoothing Using Open-CV

Vishal Sharma-14BCE0298

(*Author*)

School of Computer Science and Engineering

VIT, Vellore

Abstract— Filters are the most basic operations that can be executed on images to extract information. One of its type is blurring/smoothing the image. Blurring an image is the first step to reducing the size of images without changing their appearance too much and blurring removes high-frequency components from the image, it helps in avoiding aliasing. Blurring can be thought of as a low-pass filtering operation, and is accomplished using a simple intuitive kernel matrix.

In this proposed model I plan to achieve comparison of various filter algorithms based on the output blur produced by them for the same input image. To help recognize the difference in working of each respective filter I have also proposed to add a trackbar on output of every filter in order to better understand the difference and increase interactive nature of the project.

The filters that I plan to demonstrate in my project are:-

1. Gaussian Filter
2. Homogeneous Blur
3. Median Filter
4. Cone filter
5. Box filter
6. Bilateral Filter

The trackbar will be used to take input of kernel size for Gaussian Filter, Homogeneous Blur, Median Filter, Bilateral Filter which will offer 21 different value to users. Whereas the box filter, cone filter will be implemented using a fixed matrix of size 3*3 and 5*5 respectively.

To implement blurring using above mentioned filters we use software tool OpenCv. It has a wide range of modules that helps with a lot of computer vision problems implementation. Its architecture and memory management provides you with a framework in which you can work with images and videos without worrying about allocating and de-allocating memory for your images.

Keywords—Blur, Filter, Open-CV, Gaussian Filter, Homogeneous Blur, Median Filter, Cone filter, Box filter, Bilateral Filter, Comparison

Introduction

In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen, distinctly different from the effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales. The Gaussian kernel decreases the effect of a pixel as its distance from the central pixel increases. Applying a Gaussian blur has the effect of reducing the image's high frequency components; a Gaussian blur is thus a low pass filter.

Whereas a box blur, (also known as a box linear filter) is a spatial domain linear filter in which each pixel in the resulting image has a value equal to the average value of its neighbouring pixels in the input image. It is a form of low-pass ("blurring") filter. A 3 by 3 box blur can be written as 1/9 * determinant matrix.

The median filter is a nonlinear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing. Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise

A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. This weight can be based on a Gaussian distribution. Crucially, the weights depend not only on Euclidean distance of pixels, but also on the radiometric differences (e.g., range differences, such as color intensity, depth distance, etc.). This preserves sharp edges.

A cone filter is used to average out the intensity of a pixel in such a manner that still the pixel under consideration gets the maximum weight and hence the weight changes propotional to distance from center pixel.

The Homogeneous Blur is same as box filter in its effect thereotically. By the virtue of trackbar with various sizes of kernel one can vary its degree and performance.

Literature Survey

The sources that we cited for our project are mentioned below:-

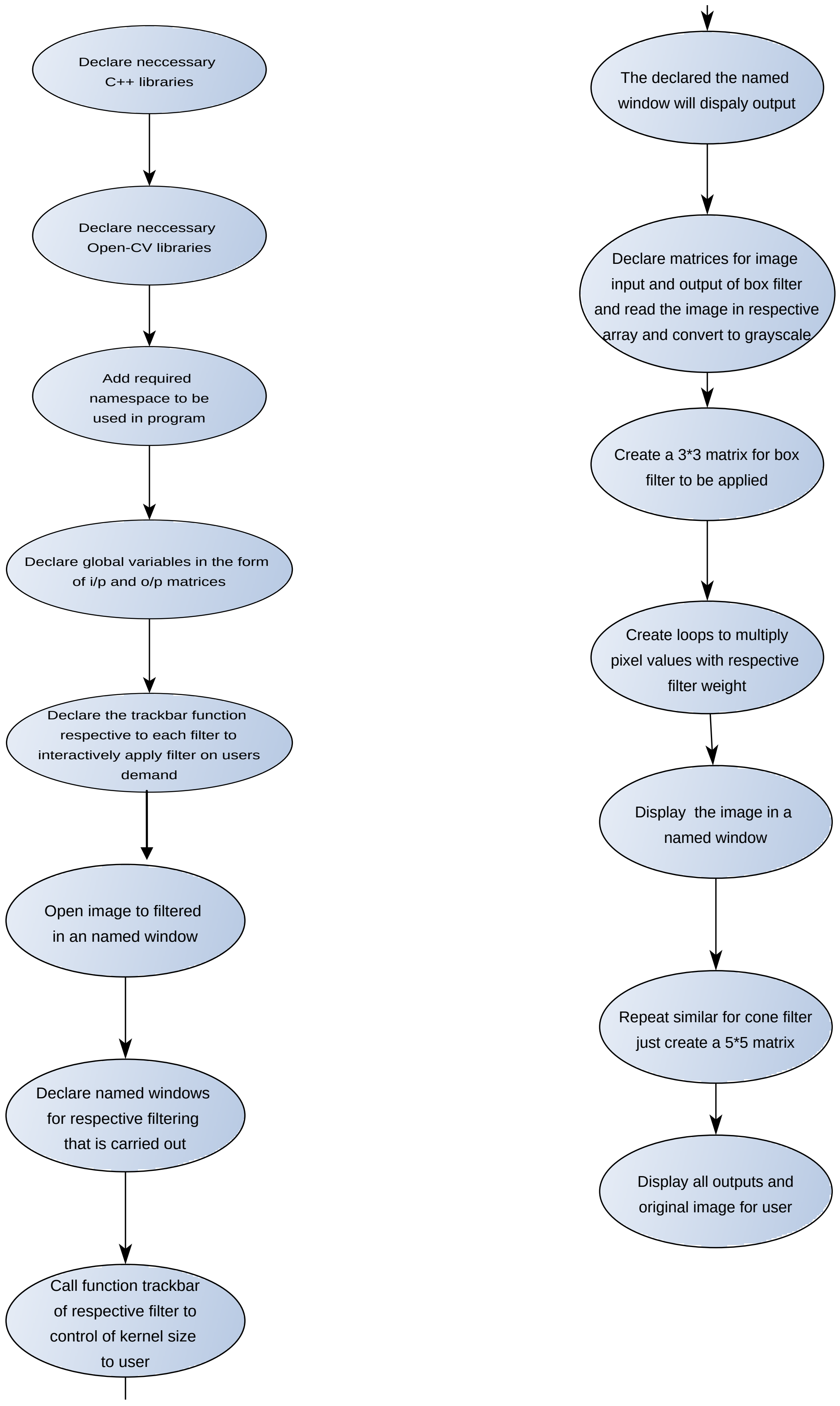
- Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing prentice hall. –
  We used the book for basic knowledge about the filters and their theoretical implementation.

- Brahmbhatt, S. (2013). *Practical OpenCV*. Apress. – We used this book for installing OpenCV software in CodeBlocks environment and understanding the functionalities of various modules present in OpenCV

- 3) Hearn, Donald, and M. Pauline Baker. "Computer Graphics—C version Prentice-Hall."

Design–

The design of the project can be understood by the following flowchart :-


Declare neccessary C++ libraries
Declare neccessary Open-CV libraries
Add required namespace to be used in program
Declare global variables in the form of i/p and o/p matrices
Declare the trackbar function respective to each filter to interactively apply filter on users demand
Open image to filtered in an named window
Declare named windows for respective filtering that is carried out
Call function trackbar of respective filter to control of kernel size to user
The declared the named window will dispaly output
Declare matrices for image input and output of box filter and read the image in respective array and convert to grayscale
Create a 3*3 matrix for box filter to be applied
Create loops to multiply pixel values with respective filter weight
Display the image in a named window
Repeat similar for cone filter just create a 5*5 matrix
Display all outputs and original image for user

Implementation-

The code written in C++ in open-cv platform is as follows along with comments for understanding:-

#include<iostream> //For enabling basic C++ functionalities

#include<opencv2/imgproc/imgproc.hpp> //contains functions for various image processes.

#include<opencv2/highgui/highgui.hpp> //this contains the functions for input and output operations.

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv; //All the OpenCV classes and functions are placed into cv namespace, to access this functionality from your code, use the cv:: specifier

Mat image, image_blurred,homo,med,bilateral; //declaring 5 matrices

int slider = 5; //initial kernel size of gaussian filter

float sigma = 0.3 * ((slider - 1) * 0.5 - 1) + 0.8; //calculating sigma for gaussian function

void on_trackbar(int, void *) { //creates trackbar for mentioned window

int k_size = max(1, slider); //selecting initial kernel size from 1 and slider input

k_size = k_size % 2 == 0 ? k_size + 1 : k_size; //checking kernel size to be odd value

setTrackbarPos("Kernel Size", "Gaussian Filter", k_size); //standard open cv function that takes trackbar name,parent image window name,initial kernel size as arguments

sigma = 0.3 * ((k_size - 1) * 0.5 - 1) + 0.8;

GaussianBlur(image, image_blurred, Size(k_size, k_size), sigma); //standard function that is used for appling gaussian blurring takes input,output image matrices,size of kernel and sigma value as arguments

imshow("Gaussian Filter", image_blurred); //shows blurred image in mentioned window

}

void on_trackbar1(int, void *) { //creates trackbar for mentioned window

int k_size = max(1, slider); //selecting initial kernel size from 1 and slider input

k_size = k_size % 2 == 0 ? k_size + 1 : k_size; //checking kernel size to be odd value

setTrackbarPos("Kernel Size", "Homogeneous blur", k_size); //standard open cv function that takes trackbar name,parent image window name,initial kernel size as arguments

blur(image, homo , Size(k_size, k_size), Point(-1,-1)); //standard function that is used for appling Homogeneous blurring

imshow("Homogeneous blur", homo); //shows blurred image in mentioned window

}

void on_trackbar2(int, void *) { //creates trackbar for mentioned window

int k_size = max(1, slider); //selecting initial kernel size from 1 and slider input

k_size = k_size % 2 == 0 ? k_size + 1 : k_size; //checking kernel size to be odd value

setTrackbarPos("Kernel Size", "Median blur", k_size); //standard open cv function that takes trackbar name,parent image window name,initial kernel size as arguments

medianBlur(image, med , slider); //standard function that is used for appling median blurring

imshow("Median blur", med); //shows blurred image in mentioned window

}

void on_trackbar3(int, void *) { //creates trackbar for mentioned window

int k_size = max(1, slider); //selecting initial kernel size from 1 and slider input

k_size = k_size % 2 == 0 ? k_size + 1 : k_size; //checking kernel size to be odd value

setTrackbarPos("Kernel Size", "Bilateral Filter", k_size); //standard open cv function that takes trackbar name,parent image window name,initial kernel size as arguments

bilateralFilter(image, bilateral , slider,slider*2,slider/2); //standard function that is used for appling bilateral blurring

imshow("Bilateral Filter", bilateral); //shows blurred image in mentioned window

}

int main() {

image = imread("lenna.png"); //takes input image in image matrix

namedWindow("Original image"); //creates original image window

```
namedWindow("Gaussian Filter");                              //creates blurred
image window

imshow("Original image", image);                          //shows input image
in original image window

sigma = 0.3 * ((slider - 1) * 0.5 - 1) + 0.8;                   //calculates sigma

GaussianBlur(image, image_blurred, Size(slider, slider), sigma);      //applies
gaussian blur

imshow("Gaussian Filter", image_blurred);                         //shows blurred
image

createTrackbar("Kernel Size", "Gaussian Filter", &slider, 21, on_trackbar);

namedWindow("Homogeneous blur");                                     //creates
blurred image window                          //shows input image in original
image window                  //calculates sigma

blur(image, homo , Size(slider, slider), Point(-1,-1));             //applies
homogeneous blur

imshow("Homogeneous blur", homo);                            //shows blurred
image

createTrackbar("Kernel Size", "Homogeneous blur", &slider, 21,
on_trackbar1);

namedWindow("Median blur");                                  //creates blurred
image window                          //shows input image in original image
window                  //calculates sigma

medianBlur(image, med , slider);      //applies median blur

imshow("Median blur", med);                          //shows blurred image

createTrackbar("Kernel Size", "Median blur", &slider, 21, on_trackbar2);

namedWindow("Bilateral Filter");                              //creates blurred
image window                          //shows input image in original image
window                  //calculates sigma

bilateralFilter(image, bilateral , slider,slider*2,slider/2);       //applies bilateral
blur

imshow("Bilateral Filter", bilateral);                       //shows blurred image

createTrackbar("Kernel Size", "Bilateral Filter", &slider, 21, on_trackbar3);



//standard function to attach trackbar named kernel size on blurred image
window taking initial kernel size and maximum kernel size

 Mat src, dst;                                         //declaring source and
destination matrix

     float sum;

     src = imread("lenna.png", CV_LOAD_IMAGE_GRAYSCALE);
//using function CV_LOAD_IMAGE_GRAYSCALE to convert image into
greyscale and loading image into source matrix.
```

```
     float Kernel[3][3] = {                              // define the kernel

                      {1/9.0, 1/9.0, 1/9.0},

                      {1/9.0, 1/9.0, 1/9.0},

                      {1/9.0, 1/9.0, 1/9.0}

                     };

       dst = src.clone();                             //copies source matrix with
a new address

       for(int y = 0; y < src.rows; y++)

          for(int x = 0; x < src.cols; x++)

             dst.at<uchar>(y,x) = 0.0;                        //.at<uchar> function
sets scalar pixel intensity at (y,x)

       for(int y = 1; y < src.rows - 1; y++){                          //convolution
operation for taking sum of average values

          for(int x = 1; x < src.cols - 1; x++){

             sum = 0.0;

             for(int k = -1; k <= 1;k++){

                for(int j = -1; j <=1; j++){

                   sum = sum + Kernel[j+1][k+1]*src.at<uchar>(y - j, x - k);
//.at<uchar> function obtains scalar pixel intensity at (y-j,x-k)

                }

             }

             dst.at<uchar>(y,x) = sum;

          }

       }



       imshow("Box Filter", dst);                                    //displaying
destination image in box Filter window

       Mat src1, dst1;                                       //declaring source
and destination matrix

     float sum1;

     src1 = imread("lenna.png", CV_LOAD_IMAGE_GRAYSCALE);
//using function CV_LOAD_IMAGE_GRAYSCALE to convert image into
greyscale and loading image into source matrix.

     float Kernel1[5][5] = {                              // define the kernel

                      {1,2,3,2,1},

                      {2,4,5,4,2},
                    {3,5,6,5,3},

                    {2,4,5,4,2},

                    {1,2,3,2,1}

                   };

        dst1 = src1.clone();                              //copies source matrix with a new address

       for(int y = 0; y < src1.rows; y++)

           for(int x = 0; x < src1.cols; x++)

               dst1.at<uchar>(y,x) = 0.0;                 //.at<uchar> function sets scalar pixel intensity at (y,x)

       for(int y = 1; y < src1.rows - 1; y++){            //convolution operation for taking sum of average values

           for(int x = 1; x < src1.cols - 1; x++){

               sum1 = 0.0;

               for(int k = -1; k <= 1;k++){

                   for(int j = -1; j <=1; j++){

                       sum1 = sum1 + Kernel[j+1][k+1]*src1.at<uchar>(y - j, x - k); //.at<uchar> function obtains scalar pixel intensity at (y-j,x-k)

                   }

               }

               dst1.at<uchar>(y,x) = sum1;

           }

       }



       imshow("Cone Filter", dst1);                       //displaying destination image in cone Filter window

     waitKey();                                           //waiting to keep result on output screen

     return 0;

}
```

Conclusion–

In conclusion we can state that all the traits specific to the filtering algorithms are verified by the project. The simultaneous presentation of all outputs obtained by applying filtering algorithms is required for comparison study. The trackbar works separately for all images making it possible for user to adjust the effect intensity by deciding interactively the kernel size of filter.

Referance-

[1.] Gonzalez, R. C., & Woods, R. E. (2002). Digital image processing prentice hall.

[2.] Brahmbhatt, S. (2013). *Practical OpenCV*. Apress

[3.] Hearn, Donald, and M. Pauline Baker. "Computer Graphics—C version Prentice-Hall."