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;
}
```