```cpp
/*****************************************************************************
*                                  EXERCISE 3                                *
*                                 ------------                               *
*                                  FILTERING                                 *
*****************************************************************************/

#include <opencv2/opencv.hpp>
#include <iostream>
#include "Ex3-CV-header.h"

using namespace cv;
using namespace std;


int main(int argc, char ** argv) {
    //double t = (double)getTickCount();                                 // TIMING

    ///_______________DON'T OVERWRITE THE ORIGINAL IMAGE!!!_____________________________
    IplImage * original, * filtered;

    /// Upload in grey scale with a resolution 640x480
    original = cvLoadImage(argv[1] , CV_LOAD_IMAGE_GRAYSCALE);

    /// To use 'lowPass'
    filtered = cvCreateImage(cvGetSize(original) , IPL_DEPTH_8U , 1);
    cvSet(filtered , 0);                                                 // Set to black




    /*************************************************************************
    *                               PARTS 1 & 2                              *
    *                              -------------                             *
    *                       Low- and High-Pass Filtering                     *
    *************************************************************************/

    ///__________________________My version (Low-Pass)___________________________________
    /// Slower but it removes much more noise in uniform areas.
    lowPass(original, filtered);

    /// Using OpenCV function (Low-Pass):
    IplImage * LP_filtered = cvCreateImage( cvGetSize(original) , IPL_DEPTH_8U , 1 );
    IplImage * HP_filtered = cvCreateImage( cvGetSize(original) , IPL_DEPTH_8U , 1 );

    ///_________________Building the Kernels for the filters______________________________

    /// LOW PASS

    float LP_mat [3][3];

    for(int i = 0 ; i < 3 ; i++)
        for(int j = 0 ; j < 3 ; j++)
            LP_mat[i][j] = 1.0 / 9;

    CvMat LP_kernel = cvMat(3 , 3 , CV_32F , LP_mat);

    //**************************************************************************

    /// HIGH PASS

    float HP_mat [3][3] = { {    0    ,  -1.0/4   ,     0  },
                            { -1.0/4 ,    2.0    ,  -1.0/4},
                            {    0    ,  -1.0/4   ,     0  }  };
    CvMat HP_kernel = cvMat(3 , 3 , CV_32F , HP_mat);

    ///________________________________FILTERING_________________________________________

    cvFilter2D(original , HP_filtered , &HP_kernel , cvPoint(-1,-1) );
    cvFilter2D(original , LP_filtered , &LP_kernel , cvPoint(-1,-1) );



    /*************************************************************************************
    *                                   PART 3                                          *
    *                                 ---------                                         *
    *                             Percentile Filter                                     *
    *************************************************************************************/

    ///____________First attempt - imaged cropped (rim rejected)_________________________
    /// Doesn't really work that well... Destroys some details and doesn't reduce
    /// much noise

    IplImage * perc_filtered = cvCreateImage(cvSize(original->width - 2, original->height - 2),
                                            IPL_DEPTH_8U, 1);

    fractileFilter(original, perc_filtered, 50);                     // Median filter: 50%



    /*************************************************************************************
    *                                   PART 4                                          *
    *                                 ---------                                         *
    *                           Laplace-Gaussian Filter                                 *
    *************************************************************************************/

    IplImage * LG_filtered = cvCreateImage(cvGetSize(original), IPL_DEPTH_8U , 1);

    /*float LG_mat [9][9] = { {0,0,1,2,2,2,1,0,0},
                              {0,1,5,10,12,10,5,1,0},
                              {1,5,15,19,16,19,15,5,1},
                              {2,10,19,-19,-64,-19,19,10,2},
                              {2,12,16,-64,-148,-64,16,12,2},
                              {2,10,19,-19,-64,-19,19,10,2},
                              {1,5,15,19,16,19,15,5,1},
                              {0,1,5,10,12,10,5,1,0},
                              {0,0,1,2,2,2,1,0,0}
                            };

    CvMat LG_kernel = cvMat(9, 9, CV_32F, LG_mat); */

    /// This gives a much better result!!!

    /*float LG_mat [3][3] = { { -1 , -1, -1},
                              { -1 ,  8, -1},
                              { -1 , -1, -1}  };*/

    CvMat LG_kernel = cvMat(3 , 3 , CV_32F , LG_mat);

    /* NOTE: Probably because of how the function cvFilter2D is implemented it's better
             to use a matrix of float and to use a constructor of CvMat with CV_32F.
             Even when the values are int, by specifying e.g. CV_16S, results are
             weird and more processing is necessary to obtain the right visual result...*/
    /// Filtering

    cvFilter2D(original , LG_filtered , &LG_kernel , cvPoint(-1,-1) );


    //*********************************************************************************

    ///_______Stream from camera in normal and filtered mode (2 separate windows)________

    doubleStream(&LG_kernel);




    /*********************************************************************************
    *                                   PARTS 5                                      *
    *                                  ---------                                     *
    *                               Triangular Filter                                *
    *********************************************************************************/

    IplImage * x_filtered = cvCreateImage(cvGetSize(original) , IPL_DEPTH_8U , 1);
    IplImage * y_filtered = cvCreateImage(cvGetSize(original) , IPL_DEPTH_8U , 1);

    /// Emphasize the horizontal direction gradients

    xGradientFilter(original, x_filtered);

    /// Emphasize the vertical direction gradients

    yGradientFilter(original, y_filtered);




    /*********************************************************************************
    *                               Display and Save                                 *
    *********************************************************************************/

    cvNamedWindow("Low-Pass Filtered Image (mine)" , WINDOW_NORMAL);
    cvNamedWindow("Original Image (grey scale)"    , WINDOW_NORMAL);
    cvNamedWindow("Laplace-Gauss Filtered Image"   , WINDOW_NORMAL);
    cvNamedWindow("Low-Pass Filtered Image"  , WINDOW_NORMAL);
    cvNamedWindow("High-Pass Filtered Image" , WINDOW_NORMAL);
    cvNamedWindow("Fractile Filtered Image"  , WINDOW_NORMAL);
    cvNamedWindow("x gradient" , WINDOW_NORMAL);
    cvNamedWindow("y gradient" , WINDOW_NORMAL);

    cvShowImage("Laplace-Gauss Filtered Image"   , LG_filtered);
    cvShowImage("Original Image (grey scale)"    , original);
    cvShowImage("Low-Pass Filtered Image (mine)" , filtered);
    cvShowImage("Low-Pass Filtered Image"  , LP_filtered);
    cvShowImage("High-Pass Filtered Image" , HP_filtered);
    cvShowImage("Fractile Filtered Image"  , perc_filtered);
    cvShowImage("x gradient" , x_filtered);
    cvShowImage("y gradient" , y_filtered);

    cvSaveImage("Original (grey scale).png" , original);
    cvSaveImage("Laplace-Gauss_Filter.png"  , LG_filtered);
    cvSaveImage("Fractile_Filter.png", perc_filtered);
    cvSaveImage("My_LP_Filtered.png" , filtered);
    cvSaveImage("LP_Filtered.png" , LP_filtered);
    cvSaveImage("HP_Filtered.png" , HP_filtered);
    cvSaveImage("x_gradient.png"  , x_filtered);
    cvSaveImage("y_gradient.png"  , y_filtered);

    cvWaitKey(0);


    /*************************************************************************************
    *                                     Clean-Up                                       *
    *************************************************************************************/

    //t = ((double)getTickCount() - t)/getTickFrequency();
    //printf("\nExecution time = %.3f s", t);                 // END TIMING

    cvReleaseImage(&original);
    cvReleaseImage(&filtered);
    cvReleaseImage(&LP_filtered);
    cvReleaseImage(&HP_filtered);
    cvReleaseImage(&perc_filtered);
    cvReleaseImage(&LG_filtered);
    cvReleaseImage(&x_filtered);
    cvReleaseImage(&y_filtered);

    cvDestroyAllWindows();

    return 0;
}
```