OpenCV Filters

Prepared by
Sergio Torres

Computer Vision
CAP 4410

2/9/20

Florida Polytechnic University

Table of Contents

List of Figures

1. Implementing Filters

1.1 Box Filter

The Box Filter had to be created using our own code and using the OpenCV function. The kernel size for both box filters starts at the kernel size that the user enters and then applies the filter. I created functions to be called where the user has to input the Image that the filter will be placed, the window that it will be shown on and the kernel size for the filter.

```
void boxFilterOrig(const Mat &Image, const char* String, int kernel_size)
{
    kernel = Mat::ones(kernel_size, kernel_size, CV_32F) / (float)(kernel_size * kernel_size);
    filter2D(Image, dst, depth, kernel, anchor, delta, BORDER_DEFAULT);

    imshow(String, dst);
}

void boxFilterOpen(const Mat &Image, const char* String, int kernel_size)
{

    boxFilter(Image, dst, depth, Size(kernel_size, kernel_size), Point(-1, 1), true, BORDER_DEFAULT);

    imshow(String, dst);

}
```

Figure 1. Box Filter using OpenCV and own code

1.2 Sobel Filter

The Sobel Filters over the X, Y, and XY axis without the use of OpenCV are included in Figures 2, 3, 4. The Sobel Filter which does use OpenCV is shown in Figure 5. The sliding window for each Sobel Filter will be put in by the user. The filters are to be called as functions with the input image, the resulting window name and the kernel size as parameters.

```
void sobFilterXOrig(const Mat& Image, const char* String, int kernel_size)
{
    newImage = Image.clone();

    int X = 0, Y = 0;
    int sobelX[3][3] =
    {
        {1, 0, -1},
        {2, 0, -2},
        {1, 0, -1}
    };

    for (int i = 1; i < Image.rows - 1; i++)
    {
        for (int j = 1; j < Image.cols - 1; j++)
        {
            for (int k = 0; k < kernel_size; k++)
            {
                for (int l = 0; l < kernel_size; l++)
                {
                    X += sobelX[k][l] * Image.at<uchar>(i + k - 1, j + l - 1);
                }
            }
            int sum = abs(X) + abs(Y);

            newImage.at<uchar>(i, j) = sum;
            X = 0;
            Y = 0;
        }
    }
    imshow(String, newImage);
}
```

Figure 2. Sobel Filter over X-Axis using own code

```
void sobFilterYOrig(const Mat &Image, const char* String, int kernel_size)
{
    newImage = Image.clone();

    int X = 0, Y = 0;

    int sobelY[3][3] =
    {
        {1, 2, 1},
        {0, 0, 0},
        {-1, -2, -1}
    };

    for (int i = 1; i < Image.rows - 1; i++)
    {
        for (int j = 1; j < Image.cols - 1; j++)
        {
            for (int k = 0; k < kernel_size; k++)
            {
                for (int l = 0; l < kernel_size; l++)
                {
                    Y += sobelY[k][l] * Image.at<uchar>(i + k - 1, j + l - 1);
                }
            }
            int sum = abs(X) + abs(Y);

            newImage.at<uchar>(i, j) = sum;
            X = 0;
            Y = 0;
        }
    }
    imshow(String, newImage);
}
```

Figure 3. Sobel Filter over Y-Axis using own code

```
void sobFilterXYOrig(const Mat &Image, const char* String, int kernel_size)
{
    newImage = Image.clone();

    int X = 0, Y = 0;
    int sobelX[3][3] =
    {
        {1, 0, -1},
        {2, 0, -2},
        {1, 0, -1}
    };

    int sobelY[3][3] =
    {
        {1, 2, 1},
        {0, 0, 0},
        {-1, -2, -1}
    };

    for (int i = 1; i < Image.rows - 1; i++)
    {
        for (int j = 1; j < Image.cols - 1; j++)
        {
            for (int k = 0; k < kernel_size; k++)
            {
                for (int l = 0; l < kernel_size; l++)
                {
                    X += sobelX[k][l] * Image.at<uchar>(i + k - 1, j + l - 1);
                    Y += sobelY[k][l] * Image.at<uchar>(i + k - 1, j + l - 1);
                }
            }
            int sum = abs(X) + abs(Y);

            newImage.at<uchar>(i, j) = sum;
            X = 0;
            Y = 0;
        }
    }
    imshow(String, newImage);
}
```

Figure 4. Sobel Filter over XY – Axis using own code

```
void sobFilterXYOpen(const Mat &Image, const char* String, int kernel_size)
{
    Mat grad;
    GaussianBlur(Image, Image, Size(kernel_size, kernel_size), 0, 0, BORDER_DEFAULT);

    Mat grad_x, grad_y;
    Mat abs_grad_x, abs_grad_y;

    Sobel(Image, grad_x, depth, 1, 0, 3, scale, delta, BORDER_DEFAULT);
    convertScaleAbs(grad_x, abs_grad_x);

    Sobel(Image, grad_y, depth, 0, 1, 3, scale, delta, BORDER_DEFAULT);
    convertScaleAbs(grad_y, abs_grad_y);

    addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);
    imshow(String, grad);

}
```

Figure 5. Sobel Filter over XY – Axis using OpenCV

1.3 Gaussian Filter

The gaussian filter was used with the Open CV function that comes with the built-in library. The code below (Figure 6) shows the Gaussian Filter being applied to the image. The window size will be set by the user and then the Gaussian Blur is applied to the image. The function for the Gaussian Blur has the input image, the resulting window name and the kernel size as parameters.

```
void gaussFilterOpen(const Mat &Image, const char* String, int kernel_size)
{

    GaussianBlur(Image, dst, Size(kernel_size, kernel_size), 0, 0);

    imshow(String, dst);
}
```

Figure 6. Gaussian Filter using OpenCV

1.4 Product Scope

In this project, the goal is to implement a Box Filters, Sobel Filters and Gaussian Filters on the images provided. We are to create our own code for the box filter apply it and compare it with the built-in OpenCV function. The Sobel Filter is to be applied over the X, Y and XY axis with our own code and then apply the Sobel Filter over the XY axis using the built-in Sobel Filter function. The Gaussian Filter is to be applied using the built-in function only. We are to apply each Filter to every image that is being provided to us and label them appropriately.

2. Results

2.1 Dog Image Results

The results for the Dog Image with every filter that was required in the project is shown below side-by-side.

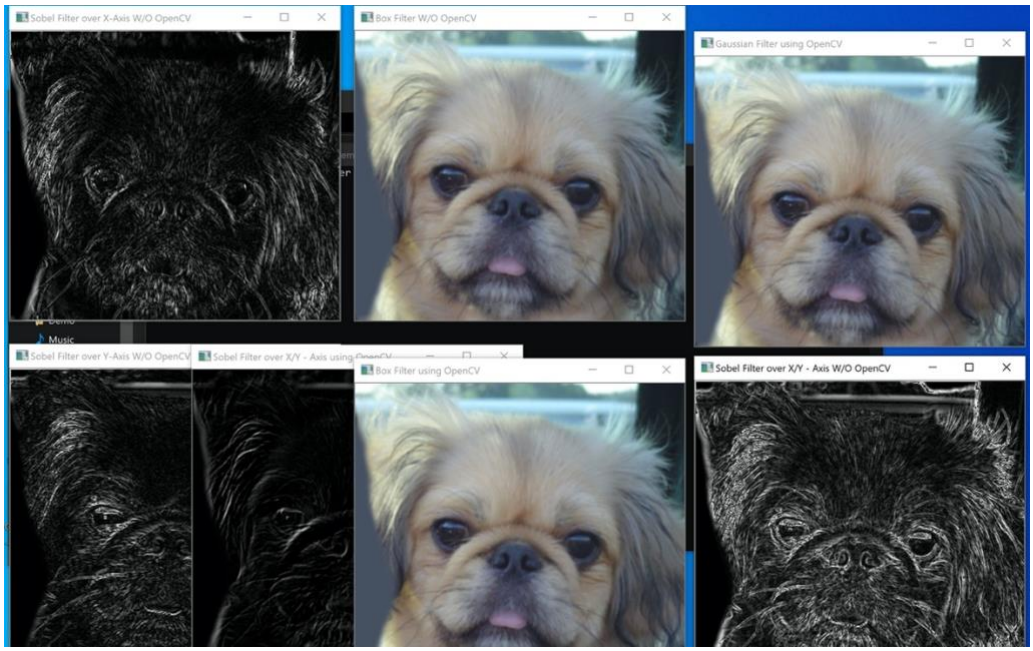

Figure 7. Dog Image with all Filters with a 3x3 sliding window

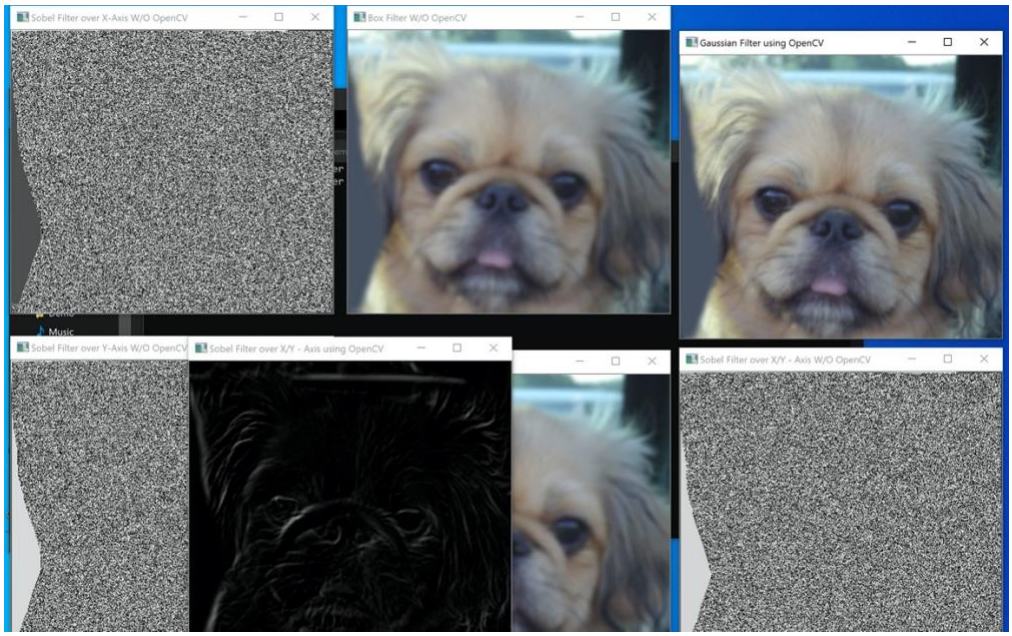

Figure 8. Dog Image with all Filters with a 7x7 sliding window

2.2 Bicycle Image Results

The results for the Bicycle Image with every filter that was required in the project is shown below side-by-side.

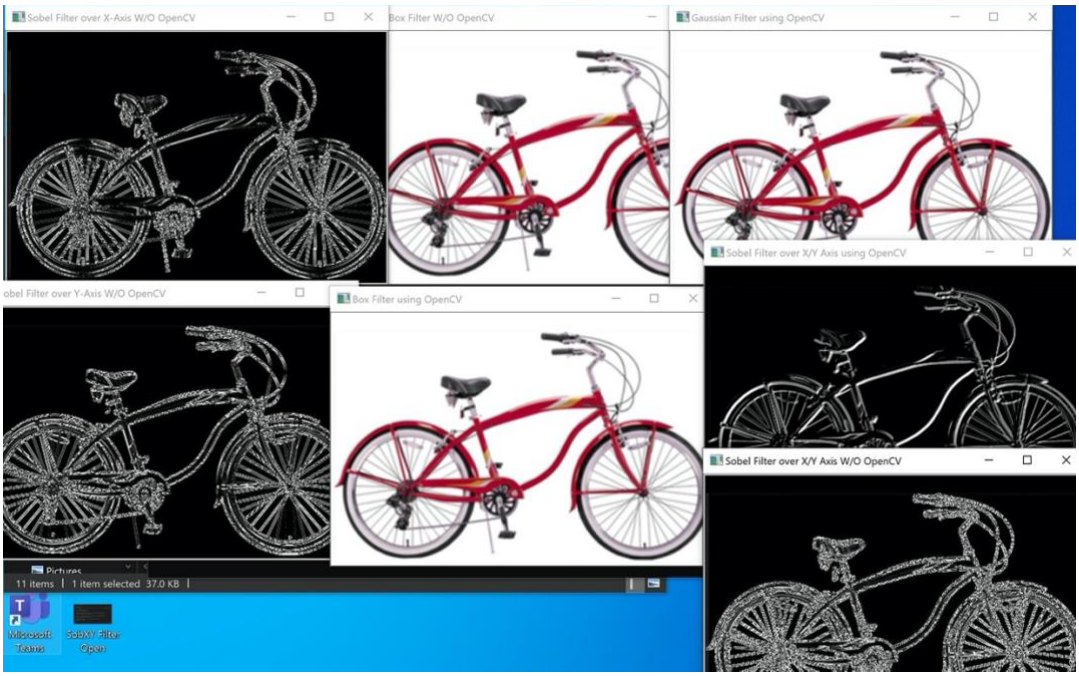

Figure 9. Bicycle Image using all Filters with a 3x3 sliding window

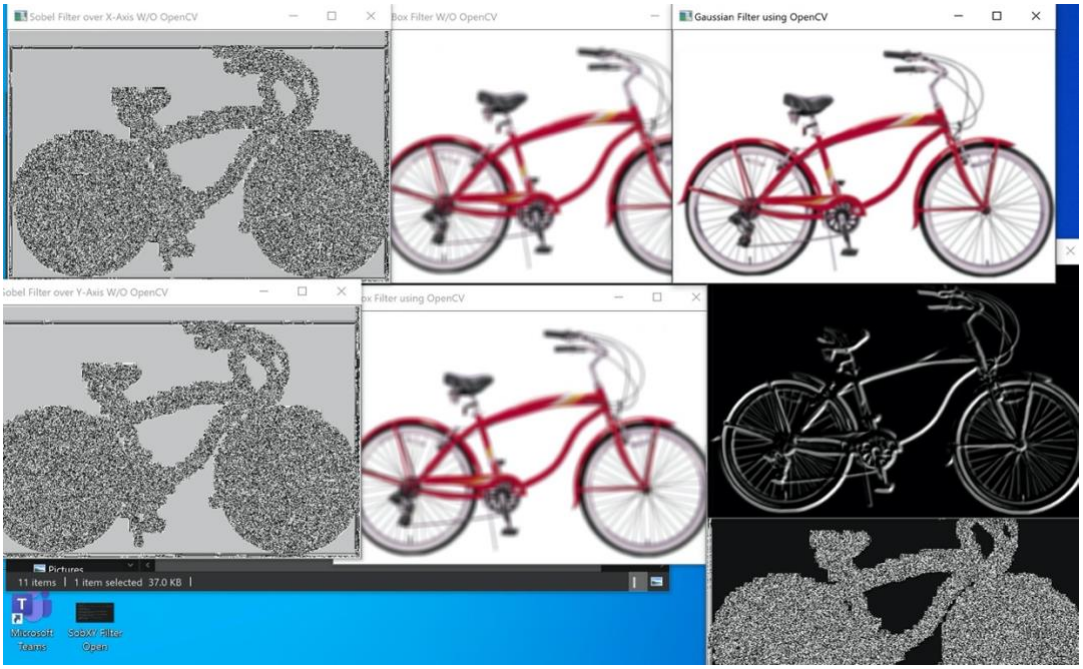

Figure 10. Bicycle Image using all Filters with a 7x7 sliding window

2.3 South_L Image Results

The results for the South_L Image with every filter that was required in the project is shown below side-by-side.

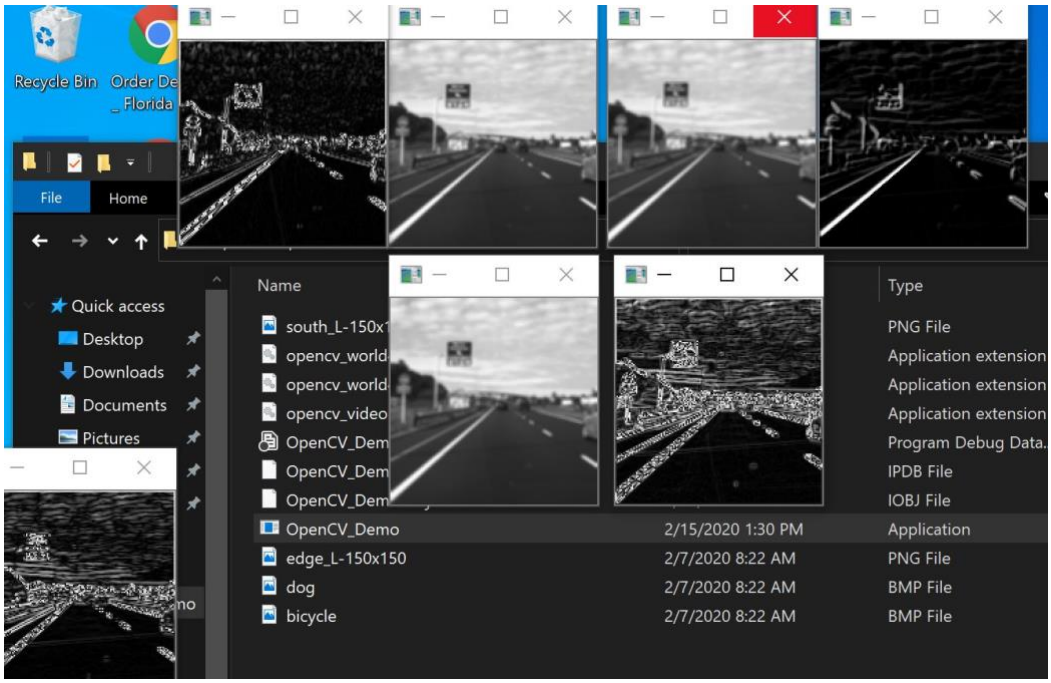

Figure 11. South_L Image using all Filters with a 3x3 sliding window

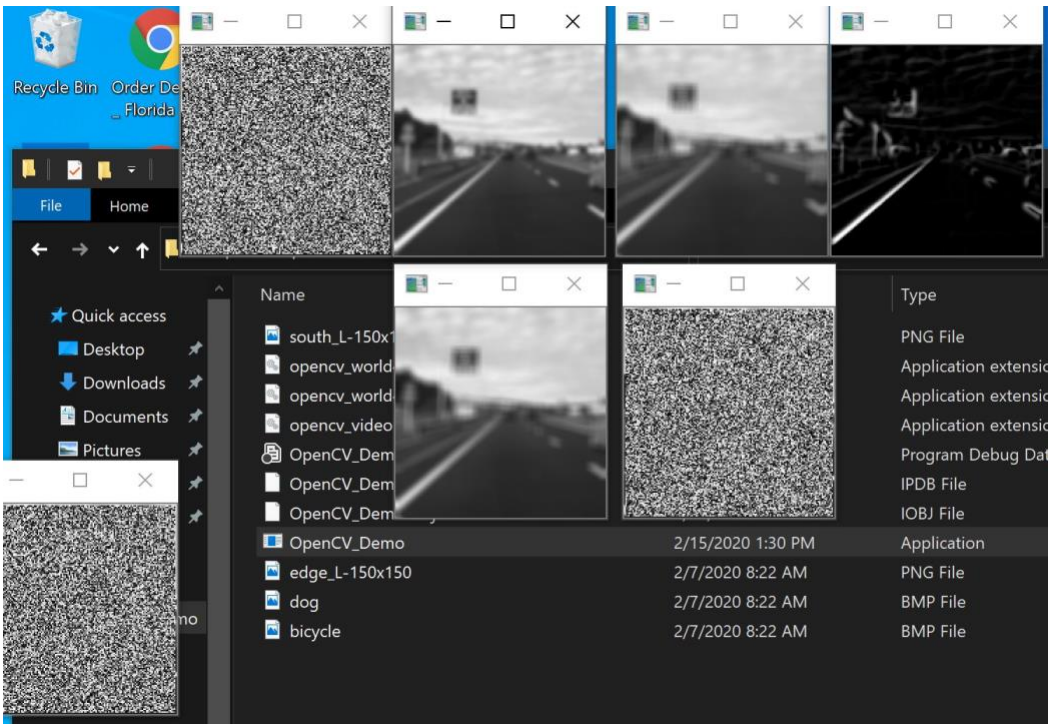

Figure 12. South_L Image using all Filters with a 7x7 sliding window

2.4 Edge Image Results

The results for the Edge Image with every filter that was required in the project is shown below side-by-side.

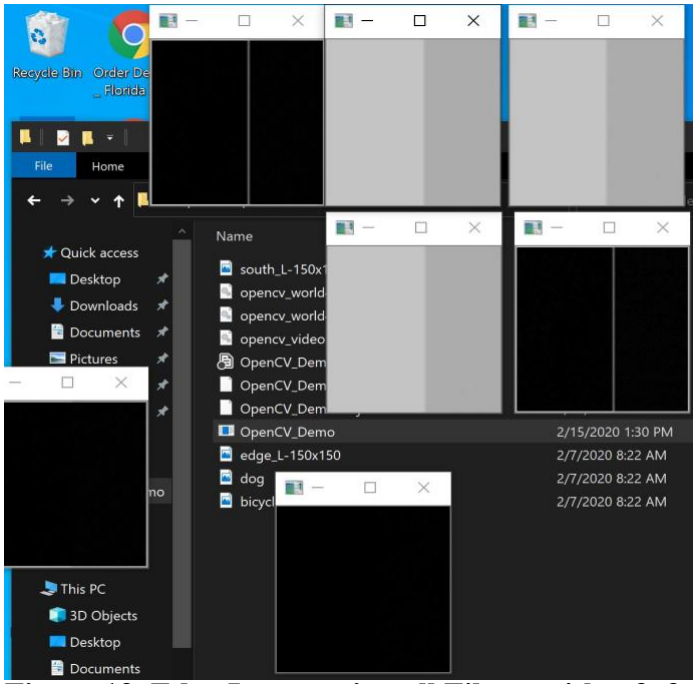

Figure 13. Edge Image using all Filters with a 3x3 sliding window

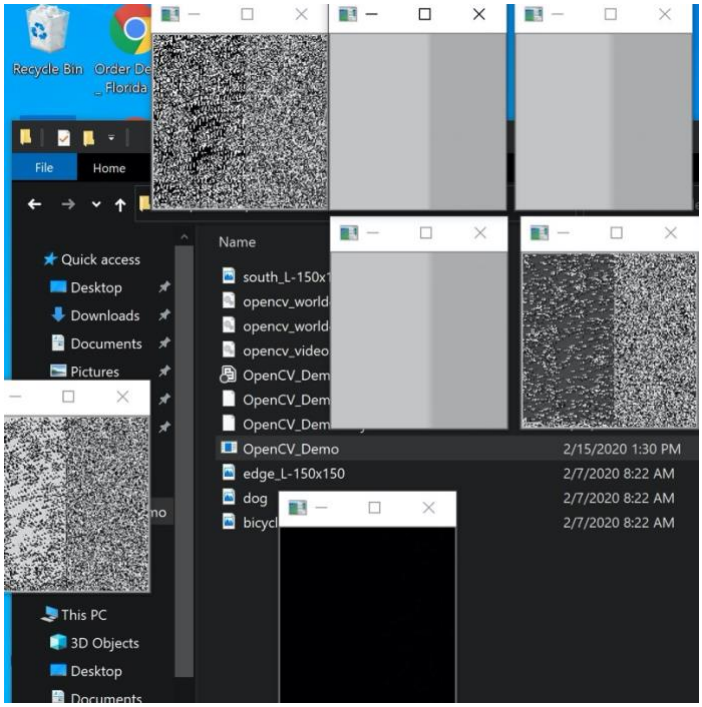

Figure 14. Edge Image using all Filters with a 7x7 sliding window

References

[1] "User Interface — Opencv 2.4.13.7 Documentation". *Docs.Opencv.Org*, 2020, https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html?highlight=namedwindow.

[2] "Sobel Edge Detection Implementation Edit." *Sobel Edge Detection Implementation - OpenCV Q&A Forum*, https://answers.opencv.org/question/110726/sobel-edge-detection-implementation/.

[3] *Edge Detection Tutorial*, www.doc.gold.ac.uk/~mas02fl/MSC101/ImageProcess/edge.html