AALBORG UNIVERSITY
STUDENT REPORT

Image processing miniproject

Horizontal and Vertical Edge Detection

Daniel Kartin

Project repository

Teacher
Tsampikos Kounalakis
tkoun@create.aau.dk
AAU CPH - MED3
Image processing

November 18, 2017

Contents

1 Algorithm

Starting out, I did some research into horizontal and vertical edge detection by going back and looking through the old slides on the subject, reading in the book, reading the opencv documentation, to understand how their version works, and doing general knowledge search on the Internet.

An edge is defined as a sudden jump in intensity in an image. Imagine the simple graph in Figure 1.1, it illustrates the hypothetical image with an intensity jump, in the algorithm this should be seen as an edge.

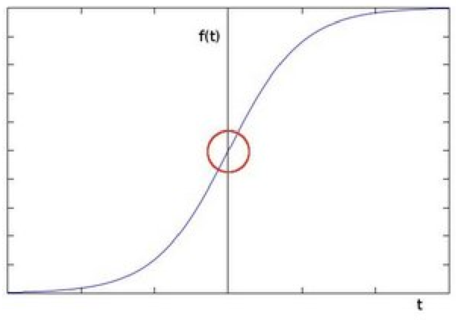

Figure 1.1: Simple representation of a hypothetical intensity jump in an image. Taken from opencv documentation for the sobel function.

Doing simple edge detection involves making two 3x3 kernels; one for horizontal and one for the vertical axis. As the assignment called for a gray scale image, the image will have to be converted to gray scale, or a gray image have to be used. The kernels that worked the best for me, looked like this:

$$xKernel = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad yKernel = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

In theory a larger kernel could be used, as long as it is an odd size. If a larger kernel is used though, the resulting edges will be thicker. These two kernels are then individually convolved with image, this calculates the approximate derivative. The resulting two images, will be called xGradient and yGradient. These images are now already showing respectively horizontal and vertical edges.

At each point of the image, we have to calculate the approximate of the gradient at that point. This is done by essentially combining the results of the two earlier convolutions. This is done by taking the following formula:

$$finalGradient = \sqrt{xGradient^2 + yGradient^2}$$

This will be the resulting final gradient, which in turn also will show both the horizontal and vertical edges in the original image.

So the resulting algorithm in steps will be:

1. Load image
2. Make kernels
3. Convolve image with x kernel
4. Convolve image with y kernel
5. Take the resulting two gradients and take the approximate gradient of them combined into one final image
6. (Optional) Have a Tuborg to celebrate

2
The code

1. An explanation of their algorithm
2. Their code
3. An explanation of their code
4. A documentation of the program (show input and output) and showing the effect of different parameters on the algorithm, if any shit

The code in it's entirety can be found in the appendix 5 and 6, or in the Project repo. This chapter will focus on the important bits, and not showcase all the code as that wouldn't be a proper use of anyone's time.

Starting out, we make the two kernels, these kernel values are based on the official numbers from the wikipedia article about edge detection, the official opencv documentation, plus trial and error.

```
//X sobel kernel
int kernelX[3][3] = {1, 0, -1, 2, 0, -2, 1, 0, -1};

//Y sobel kernel
int kernelY[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1};
```

Listing 1: Horizontal and vertical kernels

After that we take the gray scale version of the original image and clone it into all the materials that will be used later on.

```
int radius = 1;

Mat src = imgGray.clone();

//Saving the current "initial" image
Mat gradX = imgGray.clone();
Mat gradY = imgGray.clone();
Mat gradF = imgGray.clone();
```

Listing 2: Cloning the gray scale image into all the to be used materials

Now we have to convolve the image in the horizontal and vertical direction, we start off with x kernel. Convolving the image requires a double for loop, for both the x and the y positions. Starting at the very beginning of the image, it will for every pixel, add unto an integer the result from applying the kernel to the pixel and it's neighbors. The exact same thing is done for the y part of the image, except it is saved in gradY instead, and using the yKernel instead of the xKernel.

```
for (int row = radius; row < src.rows - radius; row++) {
        for (int col = radius; col < src.cols - radius; col++) {
                int scale = 0;
                for (int i = -radius; i <= radius; i++) {
                        for (int j = -radius; j <= radius; j++) {
                           scale += src.at<uchar>(row + i, col + j) * kernelX[i + radius][j + radius];
                        }
                }
                gradX.at<uchar>(row - radius, col - radius) = scale / 60;
        }
}
```

Listing 3: Looping over the the image with the x kernel to get approximate gradient on the x axis of the image.

The last part is taking the two resulting gradients from each axis, and applying the formula $gradF = \sqrt{xGrad^2 + yGrad^2}$ on every pixel from both images, essentially combining them. Then we do an extremely simple threshold to emphasize any remaining edges, but unfortunately also making the border of the image, completely white.

```
for (int row = 0; row < gradF.rows; row++) {
        for (int col = 0; col < gradF.cols; col++) {
                gradF.at<uchar>(row, col) =
                static_cast<uchar>
                (sqrt(pow(gradX.at<uchar>(row, col), 2) + pow(gradY.at<uchar>(row, col), 2)));

                if (gradF.at<uchar>(row, col) > 240) {
                        gradF.at<uchar>(row, col) = 255;
                } else {
                        gradF.at<uchar>(row, col) = 0;
                }
        }
}
```

Listing 4: Calculating an approximation of the gradient at every point, using both the x and y images

3 The program

Appendices

```cpp
#include <iostream>
#include <cmath>
#include <opencv2/opencv.hpp>

using namespace cv;

Mat img, imgGray;

void doImageProcessing() {

        //The image is really high resolution for making many windows, so I use
        //the opencv function resize, to make it more manageable
        resize(img, img, Size(640, 480));

        imshow("original image", img);

        //Converting image to grayscale
        cvtColor(img, imgGray, CV_BGR2GRAY);
        imshow("Original gray scale image", imgGray);

        //Standard Sobel kernel
        int kernelX[3][3] = {1, 0, -1, 2, 0, -2, 1, 0, -1};


        //Standard Sobel kernel
        int kernelY[3][3] = {1, 2, 1, 0, 0, 0, -1, -2, -1};


        int radius = 1;

        Mat src = imgGray.clone();

        //Saving the current "initial" image
        Mat gradX = imgGray.clone();
        Mat gradY = imgGray.clone();
        Mat gradF = imgGray.clone();

        //Looping over the the image with the x kernel
        //From this we get the gradient image
        for (int row = radius; row < src.rows - radius; row++) {
                for (int col = radius; col < src.cols - radius; col++) {
                        int scale = 0;
                        for (int i = -radius; i <= radius; i++) {
                                for (int j = -radius; j <= radius; j++) {
                                        scale +=
                                        src.at<uchar>(row + i, col + j) * kernelX[i + radius][j + radius];
                                }
                        }
                        gradX.at<uchar>(row - radius, col - radius) = scale / 60;
                }
        }
```

Listing 5: Horizontal and vertical kernels

```
imshow("X edge detection", gradX);
//Looping over the image with the y kernel
//From this we get the gradient image
for (int row = radius; row < src.rows - radius; row++) {
        for (int col = radius; col < src.cols - radius; col++) {
                int scale = 0;

                for (int i = -radius; i <= radius; i++) {
                        for (int j = -radius; j <= radius; j++) {
                                scale +=
                                src.at<uchar>(row + i, col + j)* kernelY[i + radius][j + radius];
                        }
                }
                gradY.at<uchar>(row - radius, col - radius) = scale / 60;
        }
}

imshow("Y edge detection", gradY);


//Here we calculate an approximation of the gradient at every point, using both the x and y images
for (int row = 0; row < gradF.rows; row++) {
        for (int col = 0; col < gradF.cols; col++) {

                gradF.at<uchar>(row, col) = static_cast<uchar>(sqrt(pow(gradX.at<uchar>(row, col), 2)
                 + pow(gradY.at<uchar>(row, col), 2)));
                //If the magnitude of the resulting pixel is higher than 240, max it
                //Else zero it, thus making the image binary, and kewl
                if (gradF.at<uchar>(row, col) > 1) {
                        gradF.at<uchar>(row, col) = 255;
                } else {
                        gradF.at<uchar>(row, col) = 0;
                }
        }
}

imshow("Edges", gradF);

waitKey(0);
}

int main() {

img = imread("/home/daniel/Documents/opencvFilters/stars.jpeg", CV_LOAD_IMAGE_UNCHANGED);

if (img.empty()) return -1;

doImageProcessing();
return 0;
}
```

Listing 6: Horizontal and vertical kernels