Jimmy Chion
Michael Alyono
CME211
6/7/2011

Implementation of Canny Edge Detection Algorithm

APPLICATION: This project can take a *.bmp image and will return an out *.bmp image with only the edge lines filled in. This has potential uses in various computer vision applications such as robotics. Often one wants to know where an item is located or where the edge of something is like a table. A computer can take a picture and use the canny edge detection algorithm to determine where in space the feature is located, and thus react in the proper manner

COMPUTATIONAL METHODOLOGY: This program uses the Canny Edge Detection Algorithm process (see sources). The image is read in as a BMP using the open source easyBMP C++ library and each pixel is converted into a 2-D int array that contains the brightness level of each pixel.

The first step in the process is to perform a Gaussian Filter, which reduces noise that is present in the image. The program runs through each pixel, looks at the surrounding 5x5 square, and sets the center pixel based on a weighted averaging scheme to the right. The matrix is a Gaussian distribution with sigma = 1.4.

$$\frac{1}{159} * \begin{array}{|c|c|c|c|c|}\hline 2 & 4 & 5 & 4 & 2 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 5 & 12 & 15 & 12 & 5 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 2 & 4 & 5 & 4 & 2 \\ \hline \end{array}$$

The second step is to compute the gradient magnitude and angle. This is done using a Sobel filter which looks at a 3x3 square around a pixel, applies a transformation, a computes central difference to determine the gradient. This is done in the X and Y direction which affords both magnitude and angle of the gradient. These magnitudes and angles are stored in parallel arrays.

$$\begin{array}{|c|c|c|}\hline -1 & 0 & +1 \\ \hline -2 & 0 & +2 \\ \hline -1 & 0 & +1 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

The third step is called Non-Maximum Suppression which essentially finds the local maximum of the gradient, taking into account the angle of it as well. The local maximum pixels are marked black while other pixels are left white. This brings out the basic edges of the image. The last step is to implement a hysteresis threshold where all values below the low threshold are rejected, left white, and all values above the high threshold are accepted, changed to black. Values of the middle look at neighboring pixels, and based on the overall gradient affects whether the center pixel is accepted or not.

INSTURCTIONS:

1. Navigate to src directory.
2. Type make to compile
3. Make sure your *.BMP file is located in the src folder.
4. Type ./BMP yourfilename.bmp to execute program
5. We're providing a few images that you can use: lena.bmp, sf.bmp, stanford.bmp, duke.bmp
6. The program will create a new file called output.bmp in the same directory
7. Type make clean to clean the directory

Input Parameters:

Only one argument should be added after the executable name. This argument should be the filename of the bmp image that you want to run under the algorithm. Ex: ./BMP yourfilename.bmp

We've provided a few images: lena.bmp, sf.bmp, stanford.bmp (25mb file), duke.bmp

Output:

A new file called output.bmp will be created in the src folder. This image contains the edge outline of your original bmp file.

Potential Errors:

This program has a hard time with images that have low definition or low changes in color. It has a hard time determining if there is an edge or not, however with most any object produces a recognizable output.

TESTING:

We will run several test images through the program testing different aspects of it. Determining its effectiveness is more subjective, as we are looking for how well defined and recognizable the output is in comparison to the input image.

RESULTS:

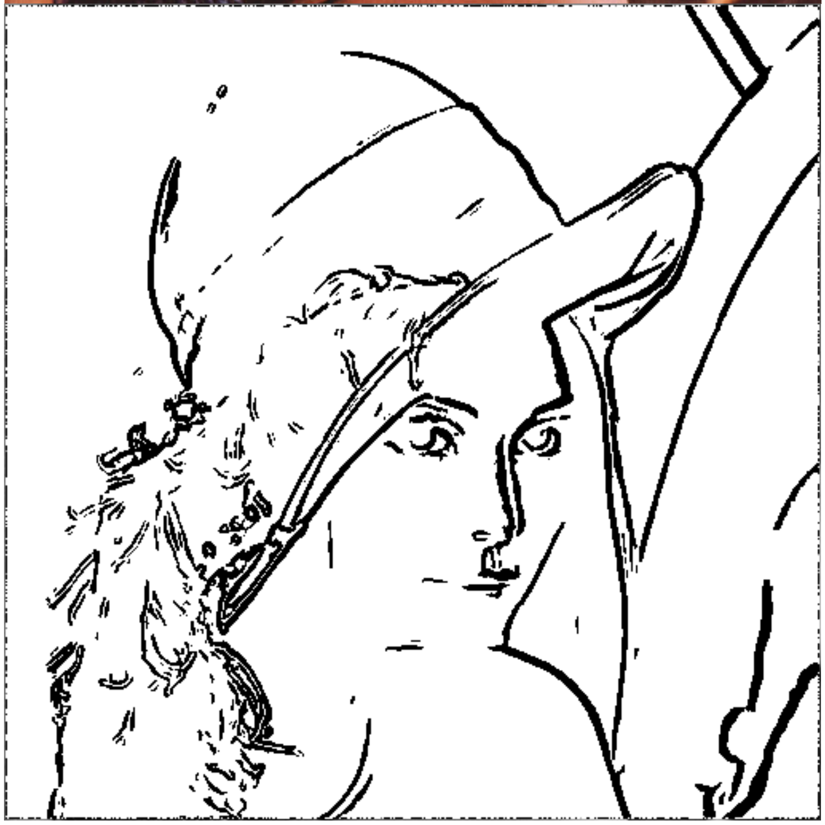
The program works very well. As you can see the San Francisco skyline is easily recognizable. The pictures of people turn out pretty well as it is recognizable that they are people, though it may be hard to identify a specific person.

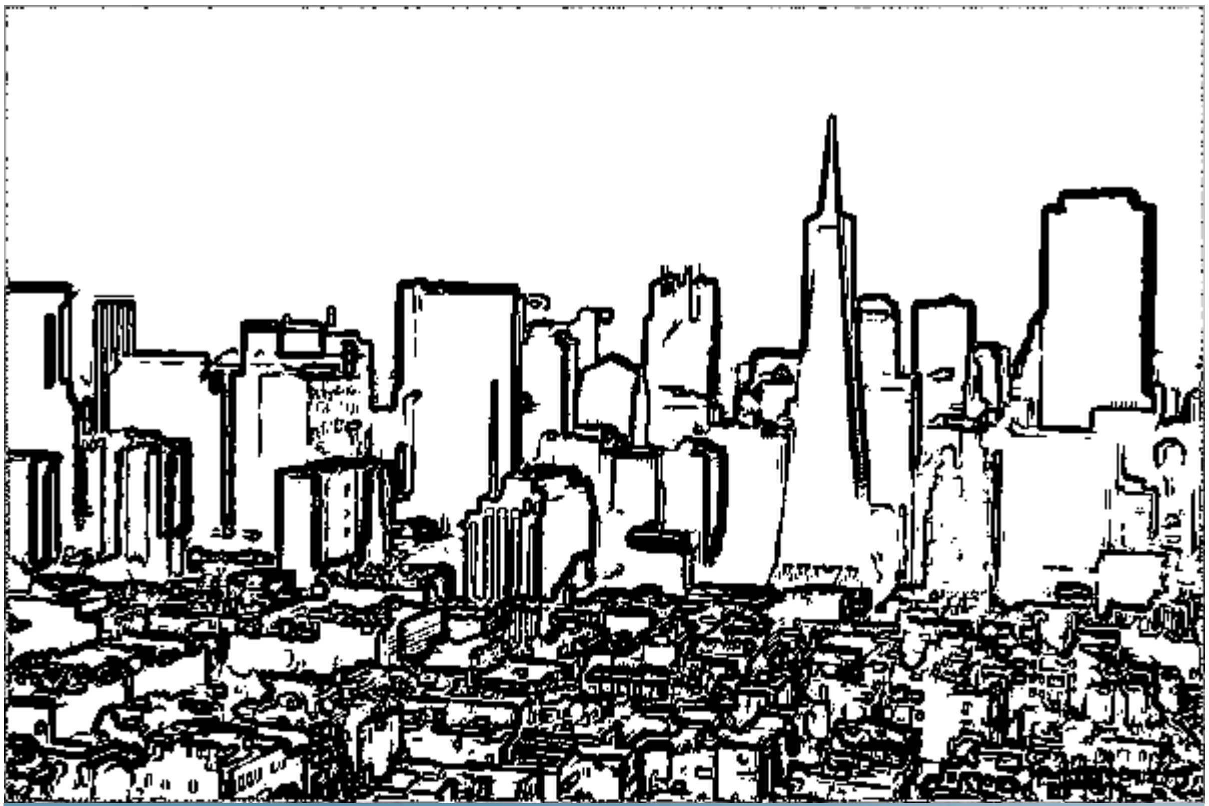
We also tested this with a 25Mb image of Stanford. And it works great and quickly, showing how robust efficient it is.

Resources used:

C++ BMP Library: http://easybmp.sourceforge.net/
Canny Edge Detection Outline: http://www.cs.uiowa.edu/~cwyman/classes/spring08-22C251/homework/canny.pdf (an assignment to a CS class)
Canny Edge Detection Info: http://en.wikipedia.org/wiki/Canny_edge_detector

Example outputs:





© Ron Niebrugge