Assignment 2 - Report

EE5178: Modern Computer Vision

Vishnu Vinod

Introduction

Edge detection is a fundamental task which is part of both computer vision and human vision. There are many existing ways of identifying edges in an image the most famous of which is the Canny Edge Detection algorithm introduced by John F. Canny in 1986. The algorithm is discussed in the next section. In this assignment we implement a Canny edge detector to carry out a simple edge detection task on the following sample image:

Figure 1: RGB Image of a clown

Canny Edge Detection

The Canny edge detection algorithm is a very useful tool in extracting visual structural information from an image. The following are the major features of the algorithm.

Original Algorithm

- Apply Gaussian blurring to smooth out the edges in the image to ensure that only the main edges in the image will be used in the detection

algorithm. Gaussian blurring involves convolution of the image with a gaussian kernel.

- Find the intensity gradients of the image along the X and the Y directions using the Sobel filters for gradient calculation or some other methods.
- Find the magnitude and the angles of the gradients at every point on the image. This can be done easily by using the intensity gradients along X and Y.
- Carry out Non-Maxima Suppression on the gradient magnitude matrix. For each pixel compare its value with that of the pixels along the direction of the gradient and suppress it by setting to zero if it is not the maximum of the three values.
- Apply double thresholding to potential edges to mark weak and strong edges separately. Double thresholding involves selecting two thresholds for weak and strong edges.
- Apply hysteresis to track the edges. Following this we suppress all the edges which are weak and not connected to any strong edges. This ensures that important weak edges are preserved.

Modified Algorithm

In this assignment we implement the Canny edge detector with the following modifications in the algorithm. These are done to make the algorithm easier to implement.

- For non-maxima suppression, the exact method for finding the pixel gradient intensities for checking if the current pixel is a maxima or not is using bilinear and bicubic interpolation. This step is avoided here and instead we use approximate the gradient direction and take the pixels closest to the direction of the gradient.
- The exact Canny algorithm uses double thresholding. However we have chosen to avoid this and go for a simple median thresholding using the median of the initial gradient magnitude matrix to threshold the non-maxima suppressed image.
- Since we no longer have weak and strong edges due to double thresholding there is no longer a need for the hysteresis step in the edge detection algorithm.

We shall now discuss the various parts of the algorithm in separate sections. The first step of reading and displaying the image can be seen in the notebook.

Converting to Grayscale

First we convert the RGB image to grayscale image. This can be done easily using the openCV function `cv2.cvtColor()`. We use the `cv2.COLOR_RGB2GRAY` argument to convert the image from the three channel RGB image to a single-channel grayscale image.

Gaussian Blurring

The input grayscale image generally contains a very large number of features with a number of fine edges. If we were to include all these edges it would generate a very messy edgemap. This happens because edge detection is easily affected by the noise in an image. In order to avoid this we can smooth the image using a gaussian kernel. The gaussian kernel is exceptionally good at removing noise from an image. We can generate a gaussian kernel of size `2k+1 x 2k+1` with a variance of σ^2 by using the following formula:

$$G_{ij} = \frac{1}{2\pi\sigma^2} \exp\Big(- \frac{(i-k)^2 + (j-k)^2}{2\sigma^2} \Big); 0 \leq i, j < 2k+1$$

In this assignemnt we use 2 different gaussian kernels with $\sigma = 1.5$ and $\sigma = 3$. Next to set the size of the second filter we proportionally increase the size from the previous kernel of size 5×5 to the nearest odd-number filter size of 11×11. The kernels are given below:

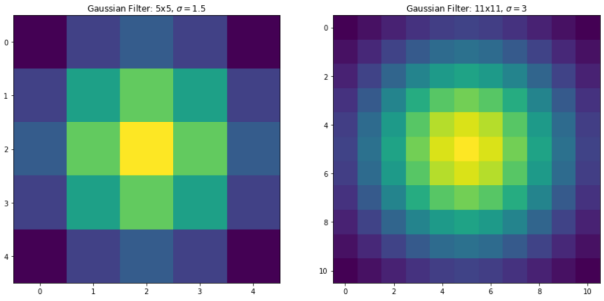


Figure 2: Gaussian Kernels

Next in order to get the smoothed image we convolve this filter with the image and get the output.

Figure 3: Gaussian Blurring using a 5×5 filter: $\sigma = 1.5$

Note that during the convolution operation we must pad the image before applying convolution so thatthe image size does not change. In order to do this we have implemented reflective padding. This is a rough implementation of `cv2.copyMakeBorder()` function and the type of padding given by `cv2.BORDER_REFLECT`.

Calculating Intensity Gradients

In order to calculate the edges we need to first calculate the gradients in the X and the Y direction for each point in the image. This can be done by convolving the image with the Sobel filters. The two filters are given as follows:

$$\text{SobelX} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \text{SobelY} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

From the $\mathbf{G_x}$ and $\mathbf{G_y}$ matrices obtained using the Sobel filters, we can now calculate the angle of the gradients as well as the magnitudes of the gradients as follows:

$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$

$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G_y}}{\mathbf{G_x}}\right)$$

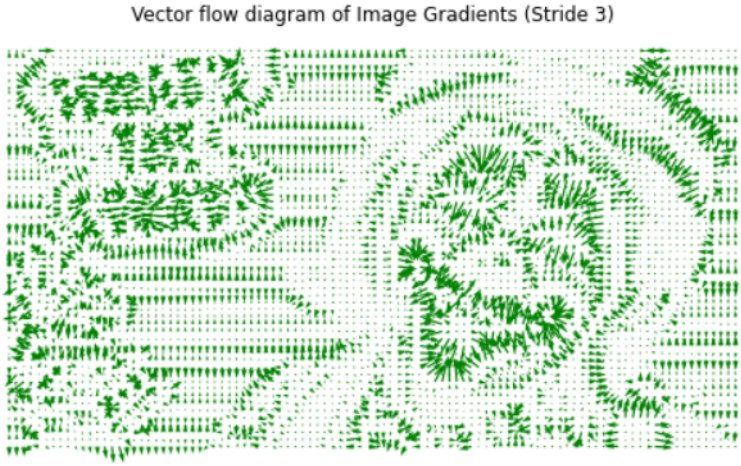
When we apply the Sobel filters onto the blurred image we get the gradients in the X and Y directions as follows:

Figure 4: Intensity Gradients using Sobel filters

Then we can print the gradient magnitude and angle plots below. Here the angle plots are such that the values of the pixels are all given in radians.

Figure 5: Gradient Magnitude and Angle Plots

The vector flow plot for the gradients is shown below:



Non-Maximum Suppression

Non-Maximum suppression is a common type of edge thinning technique that is used here as part of the Canny edge detection algorithm. Here we basically compare the value of the intensity gradient at a pixel with its values at the closest pixels along the positive and negative gradient directions. If the pixel doesn't have greater value of gradient than its neighbours it is suppressed (or set to 0).

The actual algorithm uses bilinear and bicubic interpolation to get accurate values of pixel gradient magnitudes for gradient angles along any direction. Here we instead check the gradient angles and approximate to the nearest pixels. The first step here is to convert the θ_{ij} values into degrees as well as rotate the angles by 180° for each pixel which has a negative value for the value of θ_{ij}.

The approximation for the angles is now done as given below. Overall we just adjust each θ_{ij} to align to one of the four lines:

- X axis ($\theta = 0^\circ$ or 180°) : $0^\circ \leq \theta_{ij} < 22.5^\circ$ or $157.5^\circ \leq \theta_{ij} < 180^\circ$
  We compare $\mathbf{G}_{i,j}$ with the pixels $\mathbf{G}_{i,j+1}$ and $\mathbf{G}_{i,j-1}$
- Off diagonal ($\theta = 45^\circ$) : $22.5^\circ \leq \theta_{ij} < 67.5^\circ$
  We compare $\mathbf{G}_{i,j}$ with the pixels $\mathbf{G}_{i-1,j+1}$ and $\mathbf{G}_{i+1,j-1}$.
- Y axis ($\theta = 90^\circ$) : $67.5^\circ \leq \theta_{ij} < 112.5^\circ$
  We compare $\mathbf{G}_{i,j}$ with the pixels $\mathbf{G}_{i-1,j}$ and $\mathbf{G}_{i+1,j}$.
- Main diagonal ($\theta = 135^\circ$) : $112.5^\circ \leq \theta_{ij} < 157.5^\circ$
  We compare $\mathbf{G}_{i,j}$ with the pixels $\mathbf{G}_{i-1,j-1}$ and $\mathbf{G}_{i+1,j+1}$.

If the given pixel has higher values then it is preserved otherwise it is suppressed by setting the value of the pixel to 0. An example of how NMS is applied on the gradient magnitudes is shown below.

Figure 6: Non-Maximum Suppression of Gradient Magnitudes

Median Thresholding

After carrying out NMS on the image we now carry out median thresholding with the median calculated from the gradient magnitude matrix before the NMS is applied. This is one of the most straightforward steps in the algorithm. When the pixel gradient intensity after NMS is greater than the median, we set the value at that pixel to 255 otherwise we suppress the pixel and set it to 0.

An example of the application of median thresholding to get the final edges from the Canny edge detection algorithm is shown below:

Figure 7: Median Thresholding of Gradient Magnitudes

Effect of σ on Canny Edge Detection

As discussed earlier, in this assignment we run the Canny Edge Detection for two cases where the blurring is applied using a 5×5 gaussian kernel with $\sigma = 1.5$ and using a 11×11 gaussian kernel with $\sigma = 3$. The final edges detected in both cases are shown below:

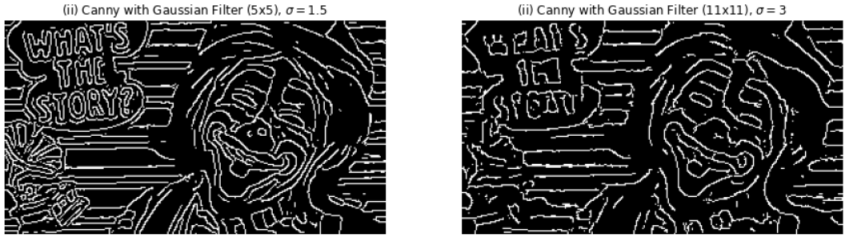


Figure 8: Canny Edge Detection

As we can clearly see the second case has fewer edges detected and the detected edges are the more prominent ones. The explanation for this is that the larger gaussian kernel over-smooths the image causing much of the edge information to be lost. We use proportionally sized gaussian kernels to compare the performance of Canny in both cases.