CannySR: Using Smart Routing to convert Canny's Binary Edge Maps to Edge Segments

Canny Edge Detector is the most widely used operator for edge detection, and OpenCV Canny (cvCanny) is the fastest known implementation of this classical algorithm. The problem with Canny is that it outputs a binary edge map, where an edge pixel (edgel) is marked (e.g., its value in the edge map is 255) and a non-edge pixel is unmarked (e.g., its value in the edge map is 0). By itself, a binary edge map is typically of low quality, consisting of gaps, notch-like structures etc. Many edge linking algorithms in the literature try to improve the quality of such binary edge maps by employing templates to fill gaps in the binary edge map and remove superfluous edgel detections.

To clean up Canny binary edge maps, and to return the map as a set of edge segments, each of which is a one-pixel wide, contiguous chain of pixels, we employ the Smart Routing (SR) algorithm from our recently proposed Edge Segment Detection Algorithm, Edge Drawing (ED), as an edge linking method for Canny's binary edge maps. The algorithm, called Canny Smart Routing (*CannySR)*, runs cvCanny to obtain a binary edge map, use the Canny edgels as anchors for SR to link the edgels and convert them to edge segments. The produced edge segments can then be used in many applications such as line, arc, circle, ellipse detection and other similar higher level object detection applications. To make Canny a parameter-free edge detector, we then implement *CannySRPF,* which first runs CanntSR with Canny's parameters at their extremes to obtain all edge segments in the image and then employs our recently proposed Edge Segment Validation Algorithm due to the Helmholtz principle to eliminate invalid edge segment detections.

1. CannySR and Experiments

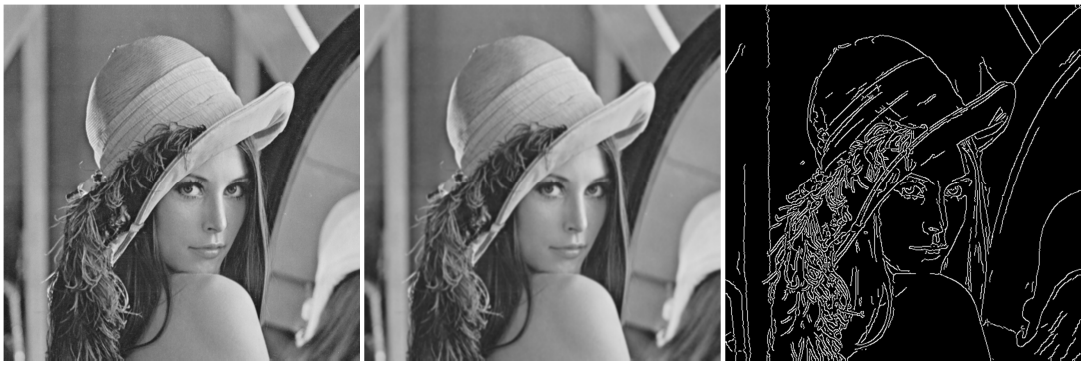
Classical Canny operator consists of the following 4 steps:

(1) Gaussian Smoothing
(2) Gradient Map and Gradient Direction Computation by the Sobel Operator
(3) Non-Maximal Suppression
(4) Double Thresholding and Hysteresis

Step (1) is implemented by OpenCV's *cvSmooth* function and the other steps are implemented by *cvCanny*, which has the following prototype:

cvCanny*(srcImg, edgeImg, lowThresh, highThresh, sobel_aperture_size);*

Here, srcImg is the input image, edgeImg is the output binary edge image, lowThresh and highThresh are the low and high canny thresholds used in step 4, and sobel_aperture_size is the size of the Sobel operator employed during gradient computation in step (2) and defaults to the classical 3x3 Sobel operator. It can also be set to 5 or 7.



Original Lena: 512x512 Lena smoothed with sigma=1.0 Binary edge map output by cvCanny

Fig.1: Steps of cvCanny for the Lena image with cannyLowThresh=40 and cannyHighThresh=80

Fig.1 shows the progression of the Lena image in the Canny edge map computation pipeline: The image is first smoothed with *cvCanny* with sigma=1.0, then cvCanny is called with low threshold=40 and high threshold=80 and the default sobel aperture size of 3. Observe that the binary edge map output by *cvCanny* is of low quality with many gaps and ragged edgels.

To convert Canny's binary edge maps to edge segments, each of which is a contiguous chain of pixels, we make use of the *SmartRouting* (SR) method from our recently proposed real-time edge segment detection algorithm, Edge Drawing (ED). The new algorithm called *CannySR* simply implements a wrapper function for cvCanny and has the following prototype:

EdgeMap ***CannySR**(unsigned char *srcImg, int width, int height, int cannyLowThresh, int cannyHighThresh, int sobelKernelApertureSize=3, double smoothingSigma=1.0);*

CannySR consists of the following steps:

(1) ***cvSmooth**(srcImg, smoothedImg, CV_GAUSSIAN, 5, 5, smoothingSigma);*
(2) ***cvCanny**(smoothedImg, binaryEdgeImg, cannyLowThresh, cannyHighThresh, sobelKernelApertureSize);*
(3) map = CreateEdgeMap(width, height);
(4) Set all edge pixels in binaryEdgeImg as anchors in EdgeMap
(5) cvSmooth(binaryEdgeMap, smoothedBinaryEdgeMap, CV_GAUSSIAN, 5, 5, 1.0);
(6) Compute the gradient magnitude and direction of all pixels that fall within smoothedBinaryEdgeMap (edge areas)
(7) Run *SmartRouting(gradientMap, directionMap, map)*
(8) Return map

The first two steps of *CannySR* is nothing but image smoothing and the computation of the binary edge map by cvCanny. *CannySR* then sets all pixels in the edge map as anchors to be used as starting points in SR. Canny's binary edge map is then smoothed with a 5x5 Gaussian kernel with sigma=1.0 so that small gaps in the binary edge map can be linked during SR. This smoothed binary edge map creates an edge areas image for which we compute the gradient map and directions. Notice that gradient computation is only performed for pixels in the edge areas. The final step is *SmartRouting* to link the edge pixels in the original Canny

binary edge map. This final step creates a set of edge segments each of which is a contiguous chain of pixels. Thus the output of *CannySR* is not a binary edge map, but a set of edge segments.

(a) Smoothed Canny Binary Edge Map (b) Black-White Edge Areas (c) CannySR output: 354 Edge Segments

Fig. 2: Steps of CannySR after cvCanny binary edge map computation: (a) Smoothed Canny binary edge map, (b) Edge areas extracted from the smoothed binary edge map, (c) 354 edge segments output by CannySR, where each color represents a different edge segment.

Fig. 2 shows the steps of *CannySR* after the computation of cvCanny's binary edge map. Fig. 2(a) shows the smoothed Canny binary edge map, Fig. 2(b) shows the edge areas, and Fig. 2(c) shows the edge segments output by *CannySR*. Notice *CannySR* output is very clean and consists only of the edge pixels in the original cvCanny output with very small gaps (1-2 pixels) also being closed during SR.

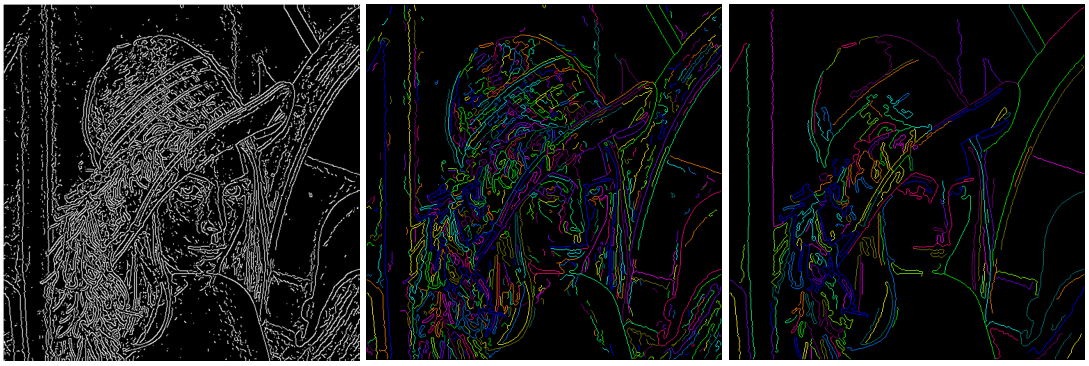
Table 1: Running times of different steps of *CannySR* for the Lena image

Step Name	Time (ms)
cvSmooth	0.92
cvCanny	4.35
SmartRouting	4.36
Total Time	9.63

Table 1 shows the dissection of the running time of *CannySR*. The running times have been obtained in a 2.2 GHz Intel Core i7-2670QM CPU.

2. CannySRPF and Experiments

Given that *CannySR* outputs a set of edge segments, each a contiguous chain of pixels, we can employ the edge segment validation algorithm due to the Helmholtz principle to eliminate invalid detections. To make *CannySR* a parameter-free edge segment detector, we simply run *CannySR* with Canny's low and high threshold values set at the extreme values, e.g., set both low and high threshold values to 20, and the validate the returned edge segments. The new algorithm is called *CannySRPF*.



(a) cvCanny edge map (b) CannySR edge segments (c) CannySRPF edge segments

Fig. 3: (a) cvCanny edge map for lowThresh=20, highThresh=20, (b) CannySR edge segments for the binary edge map, (c) validated edge segments by CannySRPF. Notice that all noise-like formations have been eliminated during edge segment validation.

Fig. 3(a) shows the edge map output by cvCanny for lowThresh and highThresh both set to 20. As you can see, the binary edge map output by cvCanny contains all details in the image but is very noisy. *CannySR* edge segments clean up some of the noise, but still contain many superfluous details. Validated edge segments by *CannySRPF,* however, contain only the major edge segments in the image cleaning up all remaining noisy edge segments.