Part 1.

1. The process of Marr-Hildreth Detector (LoG) is:

(1) Smooth the image by applying a Gaussian filter on the image, parametrized by the window size n. Since 2nd order derivatives are quite sensitive to noise.

(2) Use 2nd order derivative to detect the edges. So calculate the Laplacian of the output Gaussian from step 1.

The output would be:

(3) Since the Laplacian output (2nd order derivative) generates negative signs of the neighbors for every edge, so we need to find the zero crossing in the output from step 2 to locate every edge.

The size of the Gaussian window n is typically chosen as an odd number that is >= 6 \* σ.

The process of a Canny Edge Detector is:

(1) Smooth the image by applying a Gaussian filter, in order to reduce the impact of noise.

(2) Compute the partial derivatives as gradient, and use approximations to compute the gradient magnitude.

, edge strength

(3) Apply non-maxima suppression on the gradient magnitude to thin the edges to single edge point. The basic idea here is to explore in the direction of gradients, and preserve the largest edge strength value as edge, and suppress all other values.

(4) Detect and refine edges by conducting ***double thresholding***, steps are:

a) plot the edge values of each pixel number along gradient maxima generated from step 3.

b) set 2 thresholds: high threshold t1 and low threshold t2.

c) edge values higher than t1 are determined as edge.

d) edges connected to strong edge pixels are determined as edge.

e) all other pixels are not considered as edge.

A close up of a map

Description automatically generated

2. The polar representation of a line:

in a plane of an image, the range of θ and ρ: , , where the image is in the size of .

The process of Hough Transform to locate lines:

1) Cut and partition the image plane into accumulator cells A[ρ, θ], so that the cell at (i, j) represents the square associated with parameter values (θi, ρi).

2) Initialize all cells with value 0.

3) For each point in (xk, yk) in the image:

3.1) calculate all ρq based on ρ = xcosθ + ysinθj, where θj are all possible values partitioned in step 1.

3.2) round these ρq to its closest cell value

3.3) increment the cell value of A(q, j)

4) Now, every cell in matrix A contains the number of points in the (x, y) space that lie on the same line.

5) find the line candidates where A(i, j) is beyond a certain threshold.

3.

1) Load an image:

I = imread('ku.jpeg');

imshow(I);

A large brick building

Description automatically generated

2) Detect edges using Canny detector with thresholds of [0.2, 0.5]:

grey = rgb2gray(I);

BW = edge(grey, 'canny', [0.2 0.5]);

imshow(BW);

A close up of a logo

Description automatically generated

3) Try with different threshold combinations:

i = 0;

for a=[0.1, 0.3, 0.4, 0.6]

bw = edge(grey, 'canny', [a a+0.3]);

i = i + 1;

subplot(2,2,i),imshow(bw); title(strcat('thresholds=[', num2str(a), ', ', num2str(a+0.3), ']'));

end

A screenshot of a cell phone

Description automatically generated

With the value of thresholds going up, it tends to detect edges that are more essential, and ignoring more subtle edges.

4)