The OpenCV Canny and Hough Filters

Mr. Bryan Bagnall
SPAWAR Systems Center, Pacific
Phone: 619-553-4061
Email: bryan.bagnall@navy.mil

Mr. Sparta Cheung
SPAWAR Systems Center, Pacific
Phone: 619-553-5927
Email: sparta.cheung@navy.mil

Overview of Talk

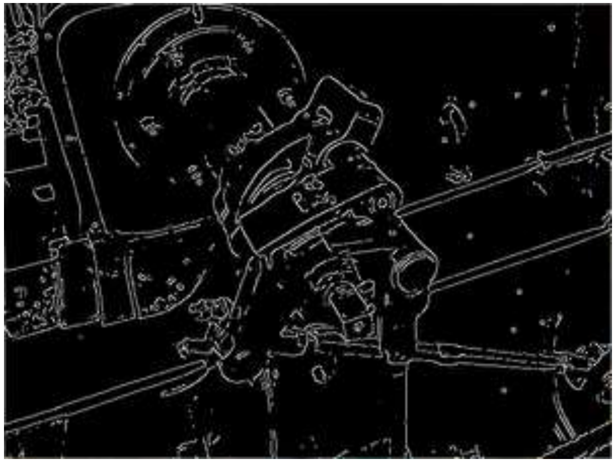
- Overview of the Canny Edge Filter
- Overview of the Hough Transform
- The OpenCV documentation
- Coding an example

The Canny Edge Filter

Multi-stage algorithm to detect a wide range of edges in images

The goals of Canny Edge Detection are:

- Mark as many real edges in the image as possible
  Edges marked should be as close as possible to the real edge
- An edge should only be marked once



The Canny Edge Filter

1st Stage: Noise reduction

In order to ensure that the image is not affected by a single “noisy” pixel to any significant degree

The filter is based on the first derivative of a Gaussian

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$where \begin{cases} x \text{ is the distance from the origin to the horizontal axis} \\ y \text{ is the distance from the origin to the vertical axis} \\ \sigma \text{ is the standard deviation of the Gaussian distribution} \end{cases}$$

The Canny Edge Filter

1st Stage: Noise reduction

Sample 5x5 Gaussian used to create the below image

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A, \ \ where\ \sigma = 1.4$$

The Canny Edge Filter

2nd Stage: Finding the intensity gradient

Edges may point in various directions → Canny uses four filters

- Horizontal (90°)
- Vertical (0°)
- Diagonals (45° and 135°)

The edge detector operator returns a value for the first derivative in the horizontal direction (G_y) and the vertical direction (G_x). The edge gradient and direction can be determined:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad \text{and} \quad \theta = arctan\left(\frac{G_y}{G_x}\right)$$

θ is then rounded to the nearest of the four angles above

The Canny Edge Filter

2nd Stage: Finding the intensity gradient

The edge detection operator we're using is the Sobel operator

This is defined mathematically as

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \qquad and \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

The Canny Edge Filter

3rd Stage: Non-maximum supression

A search is done to determine if the gradient magnitude assumes a local maximum in the gradient direction

If the rounded gradient angle is zero degrees (i.e. the edge is in the north-south direction) the point will be considered to be on the edge if its intensity is greater than the intensities in the **west and east** directions

A set of edge points is created in the form of a binary image

The Canny Edge Filter

4th Stage: Tracing edges and hysteresis thresholding

Assumption: Large intensity gradients are more likely to correspond to edges

Thresholding uses hysteresis requires two inputs

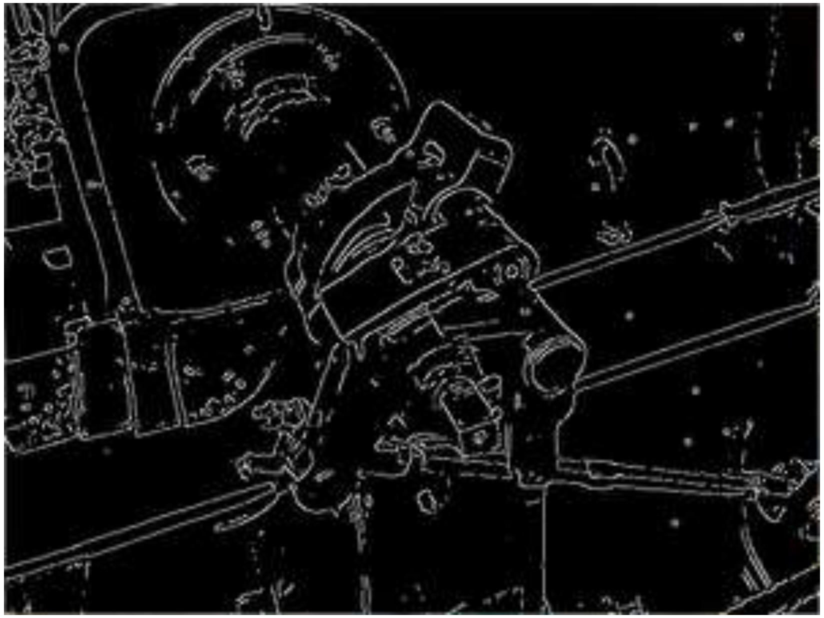
- High threshold
- Low threshold

The high threshold marks out edges that we believe to be real

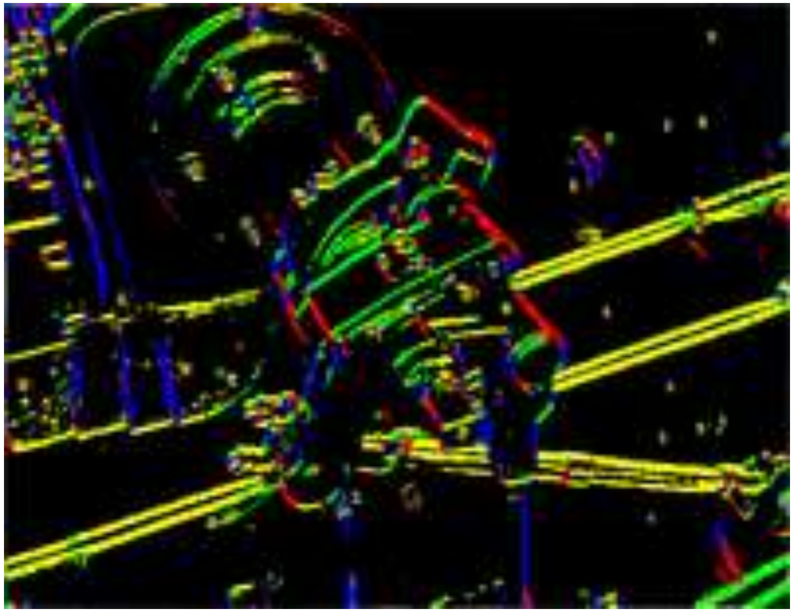
The low threshold allows us to trace lower intensity sections of that line so long as we find a starting point

Once complete we have an Edge vs. No edge binary image

The Canny Edge Filter



Edge vs. No edge Binary Image



Same image, but the four different edge directions have been colored

cv::Canny Function Documentation

Finds edges in an image using a Canny algorithm
http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-canny

void Canny(const Mat& *image*, Mat& *edges*, double *threshold1*,
double *threshold2*, int *aperatureSize=3*, bool
L2gradient=false)

The function finds edges in the input image *image* and marks them in the output map *edges* using the Canny algorithm. The smallest value between *threshold1* and *threshold2* is used for edge linking, the largest value is used to find the initial segments of strong edges

cv::Canny Function Documentation

Finds edges in an image using a Canny algorithm
http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-canny

void Canny(const Mat& *image*, Mat& *edges*, double *threshold1*,
double *threshold2*, int *aperatureSize=3*, bool
L2gradient=false)

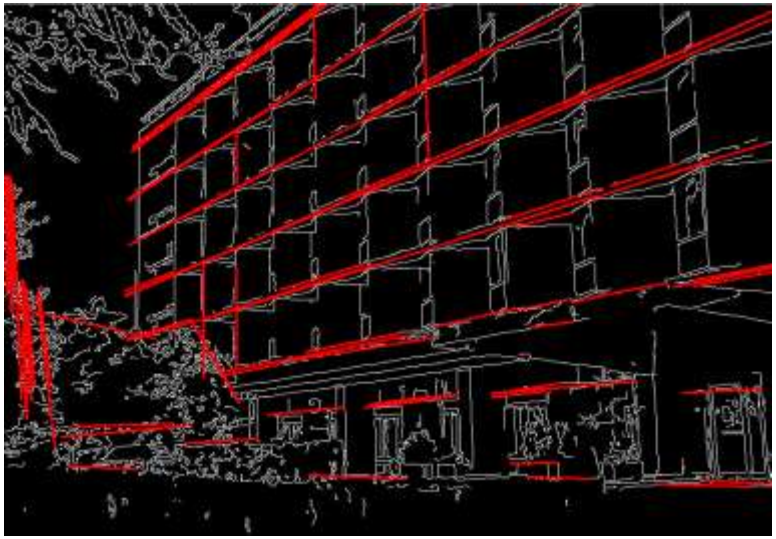
Parameters:

image – Single-channel, 8-bit input image
edges – The output edge map, it will have the same size and type as *image*
threshold1 – The low threshold for the hysteresis procedure
threshold2 – The high threshold for the hysteresis procedure
aperatureSize – Aperature size for the Sobel() operator
L2gradient – Indicates if the more accurate L2 gradient should be used (=true)

The Hough Transform

Goal is to find linear features in an image

Can also be used to detect circles in an image



The Hough Transform

In image space a straight line can be described by

$$y = mx + b$$

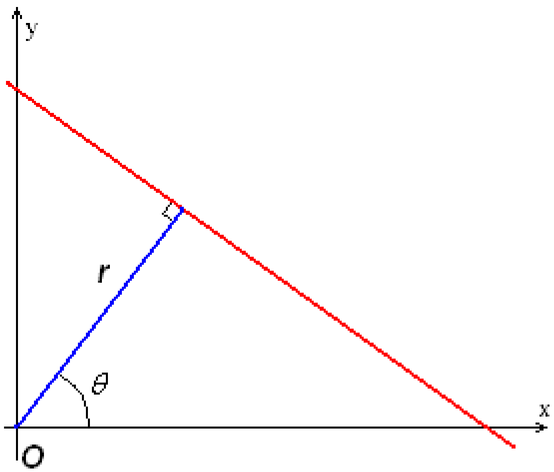
and can be graphically plotted for each pair of points (x, y)

By transforming image points in to parameter-space, considering only the parameters (m, b) a line can instead be described as a point

However, vertical lines can be unbounded in these parameters, so it is better to choose different parameters to describe our line

The Hough Transform

In Hough-space we use the parameters *r* and *θ*



Where *r* is the distance from the origin to the line

and *θ* is the angle of the vector to this closest point

Using this parameterization allows us to rewrite the line as

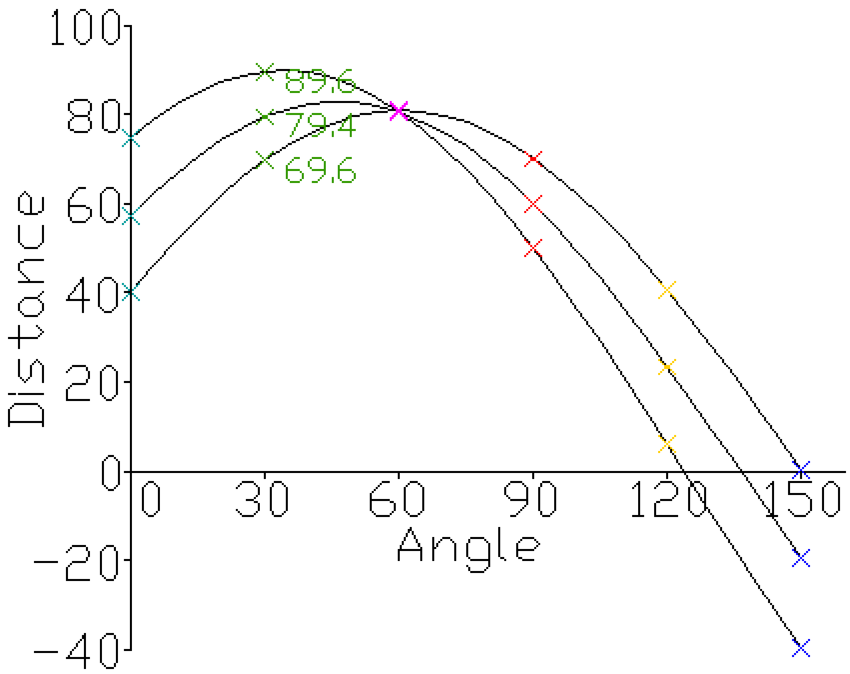
$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right), or \quad r = x\cos\theta + y\sin\theta$$

The Hough Transform

This new equation will correspond with a sinusoidal curve in the (r, θ) plane, which is unique to the point (r, θ)

If two curves (Hough-space) are superimposed, then the location where they cross corresponds to a line in the original image space

Detecting collinear points becomes a problem of detecting concurrent curves



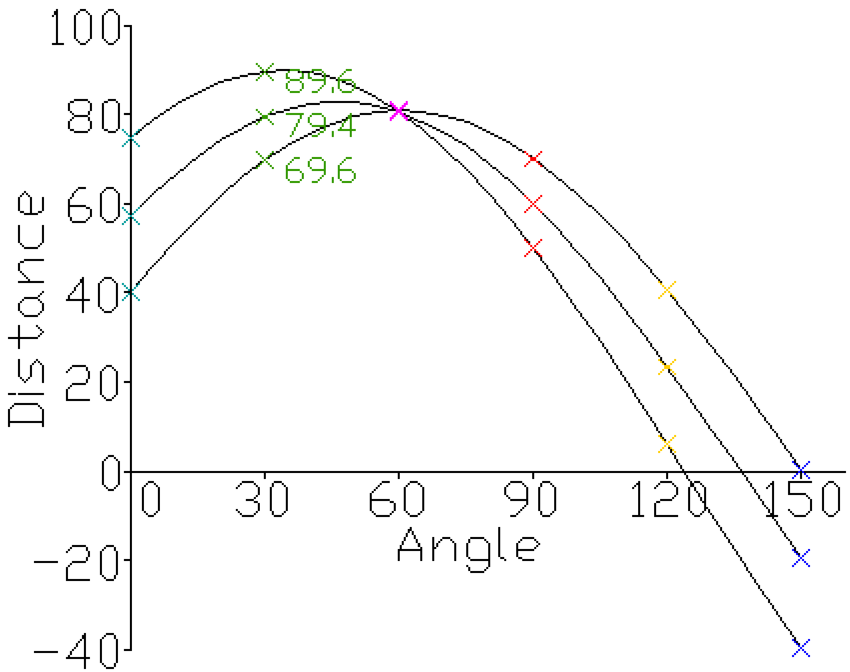
The Hough Transform

This new, unique, point (*r, θ*) can be converted back to image-space (*x, y*) and then a line is drawn on the image

The function we are going to use allows us to threshold our results

Pick the number of concurrent curves required to be a line

Some other implementations will also allow a minimum line length threshold



cv::HoughLinesP Function Documentation

Finds lines segments in a binary image using probabilistic Hough transform
http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-houghlinesp

void HoughLinesP(Mat& *image*, vector<Vec4i>& *lines*, double *rho*, double *theta*,
int *threshold*, double *minLineLength=0,* double *maxLineGap=0*)

Parameters:

image – Single-channel, 8-bit input, binary source image
lines – The output vector of lines, each line is represented by a 4-element vector (x_1, y_1, x_2, y_2) where (x_1, y_1) and (x_2, y_2) are the ending points of each line segment detected
rho – Distance resolution of the accumulator in pixels
theta – Angle resolution of the accumulator in radians

cv::HoughLinesP Function Documentation

Finds lines segments in a binary image using probabilistic Hough transform
http://opencv.willowgarage.com/documentation/cpp/imgproc_feature_detection.html#cv-houghlinesp

void HoughLinesP(Mat& *image*, vector<Vec4i>& *lines*, double *rho*, double *theta*,
int *threshold*, double *minLineLength=0,* double *maxLineGap=0*)

Parameters:

threshold – The accumulator threshold parameter, only those lines are returned that get enough votes (> threshold)
minLineLenth – The minimum line length for segments to be displayed
maxLineGap – The maximum allowed gap between points on the same line to link them together

Questions?