A Real Time Approach to Lane Marker Detection

Rahul Chandna¹, Yudhveer Singh² and Bhuvesh Batra³
www.tablearning.com
¹raulchandna@gmail.com and ²yudhveer127@gmail.com

ABSTRACT
We present a real time approach to lane marker detection. It is based on generating a filtered view of road, using selective oriented probabilistic hough transform, and dilate, erode morphological operations, which is then followed by a postprocessing step. Our algorithm can detect all lanes in still images of the street in various conditions.

KEYWORDS
Real time Lane Detection, Probabilistic Hough Transform, Edge Detector, Morphological Operators for Lane Detection, Computer vision, Lane.

1. INTRODUCTION
Car accidents kill about 50,000 people each year. Up to 90% of these accidents are caused by driver faults. Automating driving may help reduce this huge number of human fatalities. One useful technology is lane detection which has received considerable attention since the mid 1980s. Techniques used varied from using monocular to stereo vision using lowlevel morphological operations, to using probabilistic grouping and B-snakes. Lane detection is a hard problem. Challenges include: bad quality lines, shadows cast from trees, buildings and other vehicles, sharper curves, irregular/strange lane shapes, emerging and merging lanes, sun glare, writings and other markings on the road (e.g. pedestrian crosswalks), different pavement materials, and different slopes. This paper presents a simple, and effective approach to tackle most of the problems along with the OpenCV functions used in the code.

2. PROPOSED SYSTEM
This image is obtained then filtered by first converting it to gray scale, then This image is then thresholded robustly by keeping only the highest values. Edge detection is done by use of canny algorithm further optimizations is done by dilate and erode, followed probabilistic Hough transform to refine the detected straight lines and correctly detect curved lanes. Finally, a marker creation step is performed in the input image. This work provides a number of contributions. First of all, it’s robust and real time, running on a typical machine with Intel Celeron M 1.6 GHz machine. Second, it can detect any number of lane boundaries in the image not just the current lane i.e. it can detect lane boundaries of neighboring lanes as well. This is a first step towards understanding road images. Third, we present a new and fast automated vehicle control algorithm.

3. APPROACH
The first step in our system is to generate a gray scale image. When grayscale images are converted to color images, all components of the resulting image are taken to be equal; but for the reverse transformation (e.g., RGB or BGR to grayscale), the gray value is computed using the perceptually weighted formula:

\[
Y = (0.299)R + (0.587)G + (0.114)B
\]

\[
gray = cvCreateImage(cvSize(image->width, image->height),
IPL_DEPTH_8U, 1);
\]

\[
cvCvtColor(image, gray, CV_BGR2GRAY);
\]

In the case of HSV or HLS representations, hue is normally represented as a value from 0 to 360.* This can cause trouble in 8-bit representations and so, when converting to HSV, the hue is divided by 2 when the output image is an 8-bit image.

3.1. FILTERING AND THRESHOLDING
Canny used the calculus of variations—a technique which finds the function which optimizes a given functional. The optimal function in Canny’s detector is described by the sum of four exponential terms, but can be approximated by the first derivative of a Gaussian.

In order to implement the canny edge detector algorithm, a series of steps must be followed. The first step is to filter out any noise in the original image before trying to locate and detect any edges.

Since the Gaussian filter can be computed using a simple mask, it is used exclusively in the Canny algorithm. Once a suitable mask has been calculated, the Gaussian smoothing can be performed using standard convolution methods. In mathematics, a Gaussian function (named after Carl Friedrich Gauss) is a function of the form:

\[
f(x) = ae^{-(x-b)^2/2c^2}
\]

for some real constants a, b, c > 0, and e ≈ 2.718281828 (Euler’s number). Gaussian functions arise by applying the exponential function to a general quadratic function. The
Gaussian functions are thus those functions whose logarithm is a quadratic function.

The parameter \( c \) is related to the full width at half maximum (FWHM) of the peak according to

\[
\text{FWHM} = 2\sqrt{2\ln2}c = 2.35482\ldots c.
\]

Alternatively, the parameter \( c \) can be interpreted by saying that the two inflection points of the function occur at \( x = b - c \) and \( x = b + c \).

Gaussian functions are analytic, and their limit as \( x \to \infty \) is 0. Gaussian functions are among those functions that are elementary but lack elementary antiderivatives; the integral of the Gaussian function is the error function. Nonetheless their improper integrals over the whole real line can be evaluated exactly, using the Gaussian integral

\[
\int_{-\infty}^{\infty} e^{-x^2} \, dx = \sqrt{\pi}
\]

and one obtains

\[
\int_{-\infty}^{\infty} ac \frac{-(x-b)^2}{2c^2} \, dx = ac \cdot \sqrt{2\pi}.
\]

Using the following Octave code one can see the effect of changing the parameters easily

```octave
A = 1;
x0 = 0; y0 = 0;
sigma_x = 1;
sigma_y = 2;
for theta = 0:pi/100:pi
    a = cos(theta)^2/2/sigma_x^2 + sin(theta)^2/2/sigma_y^2;
    b = -sin(2*theta)/4/sigma_x^2 + sin(2*theta)/4/sigma_y^2;
    c = sin(theta)^2/2/sigma_x^2 + cos(theta)^2/2/sigma_y^2;
    [X, Y] = meshgrid(-5:.1:5, -5:.1:5);
    Z = A*exp(-a*(X-x0).^2 + 2*b*(X-x0).*(Y-y0) + c*(Y-y0).^2);
    surf(X,Y,Z); shading interp; view(-36,36); axis equal; drawnow
end
```

A convolution mask is usually much smaller than the actual image. In mathematics and, in particular, functional analysis, convolution is a mathematical operation on two functions \( f \) and \( g \), producing a third function that is typically viewed as a modified version of one of the original functions. Convolution is similar to cross-correlation. As a result, the mask is slid over the image, manipulating a square of pixels at a time.

The convolution of \( f \) and \( g \) is written \( f * g \), using an asterisk or star. Its formula is:

\[
(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) \, d\tau
\]

and

\[
\text{commutativity}
\]

While the symbol \( t \) is used above, it need not represent the time domain. But in that context, the convolution formula can be described as a weighted average of the function \( f(\tau) \) at the moment \( t \) where the weighting is given by \( g(-\tau) \) simply shifted by amount \( t \). As \( t \) changes, the weighting function emphasizes different parts of the input function.

More generally, if \( f \) and \( g \) are complex-valued functions on \( \mathbb{R}^d \), then their convolution may be defined as the integral:

\[
(f * g)(x) = \int_{\mathbb{R}^d} f(y)g(x-y) \, dy = \int_{\mathbb{R}^d} f(x-y)g(y) \, dy.
\]

The larger the width of the Gaussian mask, the lower is the detector’s sensitivity to noise. The localization error in the detected edges also increases slightly as the Gaussian width is increased.

The Gaussian mask used in my implementation is shown below.

**Figure 3** Discrete approximation to Gaussian function with \( \sigma=1.4 \).

### Step2

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. It is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation which it produces is relatively crude, in particular for high frequency variations in the image. Then, the approximate absolute gradient magnitude (edge strength) at each point can be found. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). They are shown below:
The magnitude, or EDGE STRENGTH, of the gradient is then approximated using the formula:
\[ |G| = |G_x| + |G_y| \]
Mathematically, the operator uses two 3\times3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define \( A \) as the source image, and \( G_x \) and \( G_y \) are two images which at each point contain the horizontal and vertical derivative approximations, the computations are as follows:
\[
\begin{bmatrix}
-1 & -2 & -1 \\
 0 & 0 & 0 \\
+1 & +2 & +1 
\end{bmatrix} \cdot A \quad \text{and} \quad \begin{bmatrix}
-1 & 0 & +1 \\
+2 & 0 & +2 \\
+1 & 0 & +1 
\end{bmatrix} \cdot A
\]
where * here denotes the 2-dimensional convolution operation. The \( x \)-coordinate is here defined as increasing in the "right" direction, and the \( y \)-coordinate is defined as increasing in the "down" direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:
\[ G = \sqrt{G_x^2 + G_y^2} \]
Using this information, we can also calculate the gradient's direction:
\[ \theta = \arctan \left( \frac{G_y}{G_x} \right) \]
where, for example, \( \Theta \) is 0 for a vertical edge which is darker on the left side.

Step 3
Finding the edge direction is trivial once the gradient in the \( x \) and \( y \) directions are known. However, you will generate an error whenever \( \text{sumX} \) is equal to zero. So in the code there has to be a restriction set whenever this takes place. Whenever the gradient in the \( x \) direction is equal to zero, the edge direction will equal \( 90 \) degrees or \( 0 \) degrees, depending on what the value of the gradient in the \( y \)-direction is equal to. If \( GY \) has a value of zero, the edge direction will equal \( 0 \) degrees. Otherwise the edge direction will equal \( 90 \) degrees. The formula for finding the edge direction is just:
\[ \text{theta} = \text{invatan} \left( \frac{G_y}{G_x} \right) \]

Step 4
Once the edge direction is known, the next step is to relate the edge direction to a direction that can be traced in an image. So if the pixels of a 5\times5 image are aligned as follows:
\[
\begin{array}{ccccc}
\text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\
\text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\
\text{x} & \text{x} & \text{a} & \text{x} & \text{x} \\
\text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\
\text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\
\end{array}
\]
Then, it can be seen by looking at pixel "a", there are only four possible directions when describing the surrounding pixels - 0 degrees (in the horizontal direction), 45 degrees (along the positive diagonal), 90 degrees (in the vertical direction), or 135 degrees (along the negative diagonal).
\[
cvNot( \text{gray, edge} );
\]
\[
cvCanny(\text{gray,edge,}(\text{float})\text{edge\_thresh},(\text{float})\text{edge\_thresh}*3, 3);
\]
\[
cvSobel( \text{gray, df\_dy, 1, 1, 3} );
\]
\[
\text{cvConvertScaleAbs( df\_dy , edge, 1, 0));
\]
\[
\text{cvZero( cedge );}
\]

3.2 THRESHOLD
The purpose of thresholding is to extract those pixels from some image which represent an object (either text or other line image data such as graphs, maps). Though the information is binary the pixels represent a range of intensities. Thus the objective of binarization is to mark pixels that belong to true foreground regions with a single intensity and background regions with different intensities.
\[
\text{edge} = \text{cvCreateImage(cvSize(image->width,image->height), IPL\_DEPTH\_8U, 1)};
\]
\[
\text{if(temp.val[0]>=edge\_thresh && temp.val[0]<=255 )}
\]
\[
\{ 
\text{s.val[0]=255};
\text{cvSet2D(edge,i,j,s)};
\}
\]
\[
\text{else}
\]
\[
\{ 
\text{s.val[0]=0};
\text{cvSet2D(edge,i,j,s)};
\}
\]

3.3 MORPHOLOGICAL OPERATIONS
Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image.
In grayscale morphology, images are functions mapping an Euclidean space or grid \( E \) into \( \mathbb{R} \), where \( \mathbb{R} \) is the set of reals, \( \infty \) is an element larger than any real number, and \( -\infty \) is an element smaller than any real number.
Grayscale structuring elements are also functions of the same format, called "structuring functions". Denoting an image by \( f(x) \) and the structuring function by \( b(x) \), the grayscale dilation of \( f \) by \( b \) is given by
\[
(f \ast b)(x) = \sup_{y \in E} [f(y) + b(x - y)]
\]
\[
\text{IplConvKernel* dilateKernel} = \text{cvCreateStructuringElementEx( 7, 7, 0, 0, CV\_SHAPE\_CROSS, NULL );}
\]
\[
\text{IplConvKernel* erodeKernel} = \text{cvCreateStructuringElementEx( 3, 3, 0, 0, CV\_SHAPE\_ELLIPSE, NULL );}
\]
The following figure illustrates this processing for a grayscale image. The figure shows the processing of a particular pixel in the input image. Morphological functions position the origin of the structuring element, its center element, over the pixel of interest in the input image. For pixels at the edge of an image, parts of the neighborhood defined by the structuring element can extend past the border of the image. To process border pixels, the morphological functions assign a value to these undefined pixels, as if the functions had padded the image with additional rows and columns.

### 3.4 MORPHOLOGICAL DILATION OF A GRAYSCALE IMAGE

Let \( E \) be an Euclidean space or an integer grid, and \( A \) a binary image in \( E \). The **erosion** of the binary image \( A \) by the structuring element \( B \) is defined by:

\[
A \ominus B = \{ z \in E | B_z \subseteq A \},
\]

where \( B_z \) is the translation of \( B \) by the vector \( z \), i.e.,

\[
B_z = \{ b + z | b \in B \}, \forall z \in E.
\]

When the structuring element \( B \) has a center (e.g., a disk or a square), and this center is located on the origin of \( E \), then the erosion of \( A \) by \( B \) can be understood as the locus of points reached by the center of \( B \) when \( B \) moves inside \( A \). For example, the erosion of a square of side 10, centered at the origin, by a disk of radius 2, also centered at the origin, is a square of side 6 centered at the origin. The erosion of \( A \) by \( B \) is also given by the expression:

\[
A \ominus B = \bigcap_{b \in B} A_b.
\]

In grayscale morphology, images are functions mapping an Euclidean space or grid \( E \) into \( \mathbb{R} \cup \{ \infty, -\infty \} \), where \( \mathbb{R} \) is the set of reals, \( \infty \) is an element larger than any real number, and \( -\infty \) is an element smaller than any real number.

```cpp
cvDilate(edge,edge,dilateKernel,1);
cvErode(edge,edge,erodeKernel,3);
```

Denoting an image by \( f(x) \) and the grayscale structuring element by \( b(x) \), the grayscale erosion of \( f \) by \( b \) is given by

\[
(f \ominus b)(x) = \inf_{y \in E} [f(y) - b(y - x)],
\]

### 3.5. HOUGH TRANSFORM

The Hough transform is not a fast algorithm for finding infinite lines in images of a certain size. Since additional analysis is required to detect finite lines, this is even slower. A way to speed up the Hough Transform and finding finite lines at the same time is the Progressive Probabilistic Hough Transform (PPHT). The idea of this method is to transform randomly selected pixels in the edge image into the accumulator. When a bin in the accumulator corresponding to a particular infinite line has got a certain number of votes, the edge image is searched along that line to see if one or more finite line(s) are present.

```cpp
lines=cvHoughLines2(edge,storage,
                      CV_HOUGH_PROBABILISTIC,1,CV_PI/90, edge_thresh >0 ?edge_thresh/2:1, 0, line_length );
```

Then all pixels on that line are removed from the edge image. In this way the algorithm returns finite lines. If the vote threshold is low the number of pixels to evaluate in the accumulator gets small.

### 4. EXPERIMENTAL RESULTS
5. CONCLUSION
We present a real time approach to lane marker detection. It is based on generating a filtered view of road, using selective oriented probabilistic hough transform, and dilate, erode morphological operations, which is then followed by a post-processing step. Our algorithm can detect all lanes in still images of the street in various conditions.

REFERENCES