Hough Line Transform

Prev Tutorial: Canny Edge Detector

Next Tutorial: Hough Circle Transform

Goal

In this tutorial you will learn how to:

- Use the OpenCV functions **HoughLines()** and **HoughLinesP()** to detect lines in an image.

Theory

Note
The explanation below belongs to the book **Learning OpenCV** by Bradski and Kaehler.

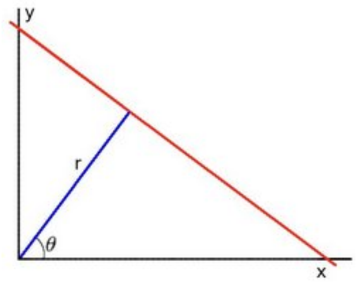
Hough Line Transform

1. The Hough Line Transform is a transform used to detect straight lines.
2. To apply the Transform, first an edge detection pre-processing is desirable.

How does it work?

1. As you know, a line in the image space can be expressed with two variables. For example:

   a. In the **Cartesian coordinate system:** Parameters: (m, b).
   b. In the **Polar coordinate system:** Parameters: (r, θ)



For Hough Transforms, we will express lines in the *Polar system*. Hence, a line equation can be written as:

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

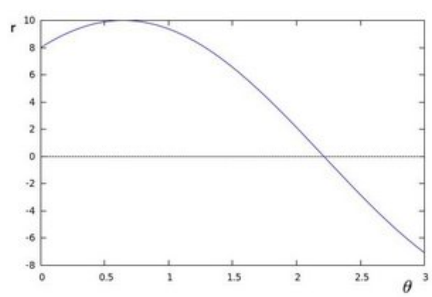
Arranging the terms: $r = x\cos\theta + y\sin\theta$

1. In general for each point (x_0, y_0), we can define the family of lines that goes through that point as:

$$r_\theta = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

   Meaning that each pair (r_θ, θ) represents each line that passes by (x_0, y_0).

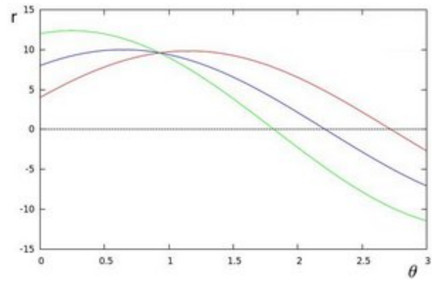
2. If for a given (x_0, y_0) we plot the family of lines that goes through it, we get a sinusoid. For instance, for $x_0 = 8$ and $y_0 = 6$ we get the following plot (in a plane θ - r):



   We consider only points such that $r > 0$ and $0 < \theta < 2\pi$.

3. We can do the same operation above for all the points in an image. If the curves of two different points intersect in the plane θ - r, that means that both

points belong to a same line. For instance, following with the example above and drawing the plot for two more points: $x_1 = 4$, $y_1 = 9$ and $x_2 = 12$, $y_2 = 3$, we get:



The three plots intersect in one single point $(0.925, 9.6)$, these coordinates are the parameters (θ, r) or the line in which (x_0, y_0), (x_1, y_1) and (x_2, y_2) lay.

4. What does all the stuff above mean? It means that in general, a line can be *detected* by finding the number of intersections between curves.The more curves intersecting means that the line represented by that intersection have more points. In general, we can define a *threshold* of the minimum number of intersections needed to *detect* a line.
5. This is what the Hough Line Transform does. It keeps track of the intersection between curves of every point in the image. If the number of intersections is above some *threshold*, then it declares it as a line with the parameters (θ, r_θ) of the intersection point.

Standard and Probabilistic Hough Line Transform

OpenCV implements two kind of Hough Line Transforms:

a. **The Standard Hough Transform**

- It consists in pretty much what we just explained in the previous section. It gives you as result a vector of couples (θ, r_θ)
- In OpenCV it is implemented with the function **HoughLines()**

b. **The Probabilistic Hough Line Transform**

- A more efficient implementation of the Hough Line Transform. It gives as output the extremes of the detected lines (x_0, y_0, x_1, y_1)
- In OpenCV it is implemented with the function **HoughLinesP()**

What does this program do?

- Loads an image
- Applies a *Standard Hough Line Transform* and a *Probabilistic Line Transform*.
- Display the original image and the detected line in three windows.

Code

C++ Java Python

The sample code that we will explain can be downloaded from here. A slightly fancier version (which shows both Hough standard and probabilistic with trackbars for changing the threshold values) can be found here.

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
    // Declare the output variables
    Mat dst, cdst, cdstP;

    const char* default_file = "sudoku.png";
    const char* filename = argc >=2 ? argv[1] : default_file;

    // Loads an image
    Mat src = imread( samples::findFile( filename ), IMREAD_GRAYSCALE );

    // Check if image is loaded fine
    if(src.empty()){
        printf(" Error opening image\n");
        printf(" Program Arguments: [image_name -- default %s] \n", default_file);
```

```
        return -1;
    }

    // Edge detection
    Canny(src, dst, 50, 200, 3);

    // Copy edges to the images that will display the results in BGR
    cvtColor(dst, cdst, COLOR_GRAY2BGR);
    cdstP = cdst.clone();

    // Standard Hough Line Transform
    vector<Vec2f> lines; // will hold the results of the detection
    HoughLines(dst, lines, 1, CV_PI/180, 150, 0, 0 ); // runs the actual detection
    // Draw the lines
    for( size_t i = 0; i < lines.size(); i++ )
    {
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
    }

    // Probabilistic Line Transform
    vector<Vec4i> linesP; // will hold the results of the detection
    HoughLinesP(dst, linesP, 1, CV_PI/180, 50, 50, 10 ); // runs the actual detection
    // Draw the lines
    for( size_t i = 0; i < linesP.size(); i++ )
    {
        Vec4i l = linesP[i];
        line( cdstP, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, LINE_AA);
    }

    // Show results
    imshow("Source", src);
    imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst);
    imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP);

    // Wait and Exit
    waitKey();
    return 0;
}
```

Explanation

C++ Java Python

Load an image:

```
    const char* default_file = "sudoku.png";
    const char* filename = argc >=2 ? argv[1] : default_file;

    // Loads an image
    Mat src = imread( samples::findFile( filename ), IMREAD_GRAYSCALE );

    // Check if image is loaded fine
    if(src.empty()){
        printf(" Error opening image\n");
        printf(" Program Arguments: [image_name -- default %s] \n", default_file);
        return -1;
    }
```

Detect the edges of the image by using a Canny detector:

```
    // Edge detection
    Canny(src, dst, 50, 200, 3);
```

Now we will apply the Hough Line Transform. We will explain how to use both OpenCV functions available for this purpose.

Standard Hough Line Transform:

First, you apply the Transform:

```
    // Standard Hough Line Transform
    vector<Vec2f> lines; // will hold the results of the detection
```

```
HoughLines(dst, lines, 1, CV_PI/180, 150, 0, 0 ); // runs the actual detection
```

- with the following arguments:
  - *dst*: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
  - *lines*: A vector that will store the parameters (r, θ) of the detected lines
  - *rho* : The resolution of the parameter r in pixels. We use **1** pixel.
  - *theta*: The resolution of the parameter θ in radians. We use **1 degree** (CV_PI/180)
  - *threshold*: The minimum number of intersections to "*detect*" a line
  - *srn* and *stn*: Default parameters to zero. Check OpenCV reference for more info.

And then you display the result by drawing the lines.

```
// Draw the lines
for( size_t i = 0; i < lines.size(); i++ )
{
    float rho = lines[i][0], theta = lines[i][1];
    Point pt1, pt2;
    double a = cos(theta), b = sin(theta);
    double x0 = a*rho, y0 = b*rho;
    pt1.x = cvRound(x0 + 1000*(-b));
    pt1.y = cvRound(y0 + 1000*(a));
    pt2.x = cvRound(x0 - 1000*(-b));
    pt2.y = cvRound(y0 - 1000*(a));
    line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
}
```

Probabilistic Hough Line Transform

First you apply the transform:

```
// Probabilistic Line Transform
vector<Vec4i> linesP; // will hold the results of the detection
HoughLinesP(dst, linesP, 1, CV_PI/180, 50, 50, 10 ); // runs the actual detection
```

- with the arguments:
  - *dst*: Output of the edge detector. It should be a grayscale image (although in fact it is a binary one)
  - *lines*: A vector that will store the parameters $(x_{start}, y_{start}, x_{end}, y_{end})$ of the detected lines
  - *rho* : The resolution of the parameter r in pixels. We use **1** pixel.
  - *theta*: The resolution of the parameter θ in radians. We use **1 degree** (CV_PI/180)
  - *threshold*: The minimum number of intersections to "*detect*" a line
  - *minLineLength*: The minimum number of points that can form a line. Lines with less than this number of points are disregarded.
  - *maxLineGap*: The maximum gap between two points to be considered in the same line.

And then you display the result by drawing the lines.

```
// Draw the lines
for( size_t i = 0; i < linesP.size(); i++ )
{
    Vec4i l = linesP[i];
    line( cdstP, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, LINE_AA);
}
```

Display the original image and the detected lines:

```
// Show results
imshow("Source", src);
imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst);
imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP);
```

Wait until the user exits the program

```
// Wait and Exit
waitKey();
return 0;
```

Result

Note

The results below are obtained using the slightly fancier version we mentioned in the *Code* section. It still implements the same stuff as above, only adding the Trackbar for the Threshold.

Using an input image such as a sudoku image. We get the following result by using the Standard Hough Line Transform: