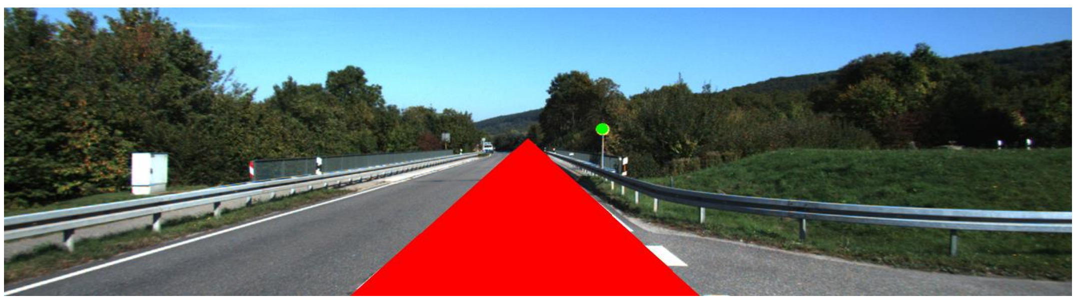
LAB 3: Road and Signs Detection with the Hough Transform

Computer Vision 2022

P. Zanuttigh

Find Lines and Circles in an Image



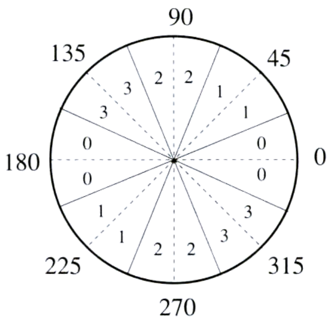
Detect the street lines and the circular road signs:

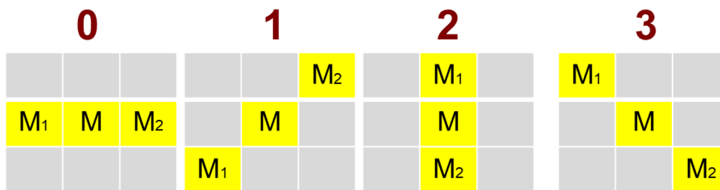
1. Find the edges (e.g., with Canny)
2. Find the street lines with the Hough Transform
3. Find the circles corresponding to road signs
   - Using the Hough Transform for circles

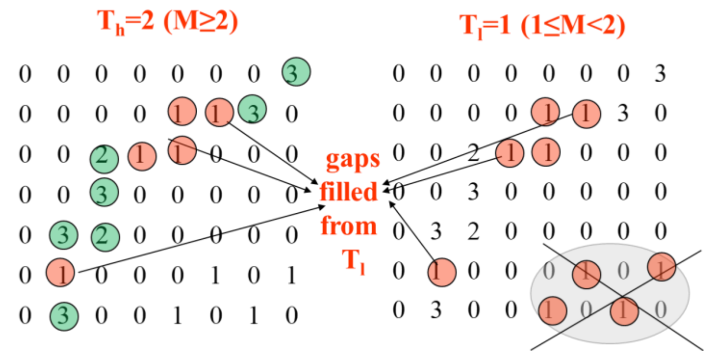
Recall: Canny Edge Detector

1. Smoothing with a Gaussian filter
2. Compute gradient (module and direction)
3. Quantize the gradient angles
4. Non-maxima suppression
5. Thresholding with double threshold

Canny: Key Advancements







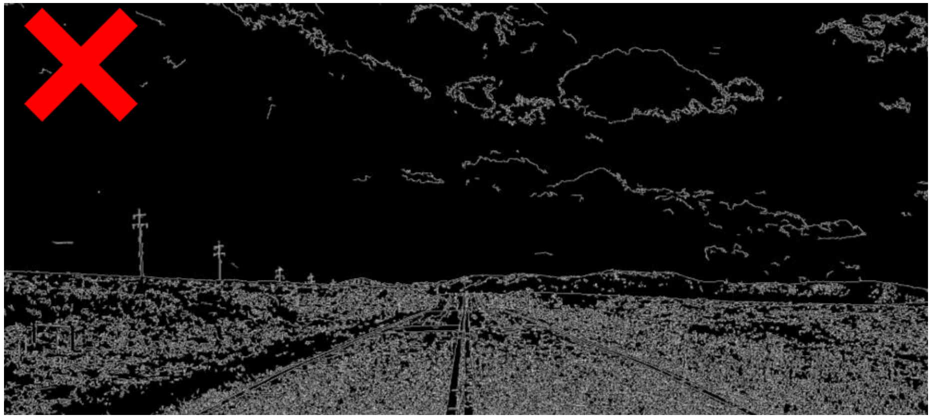
Non-maxima Suppression

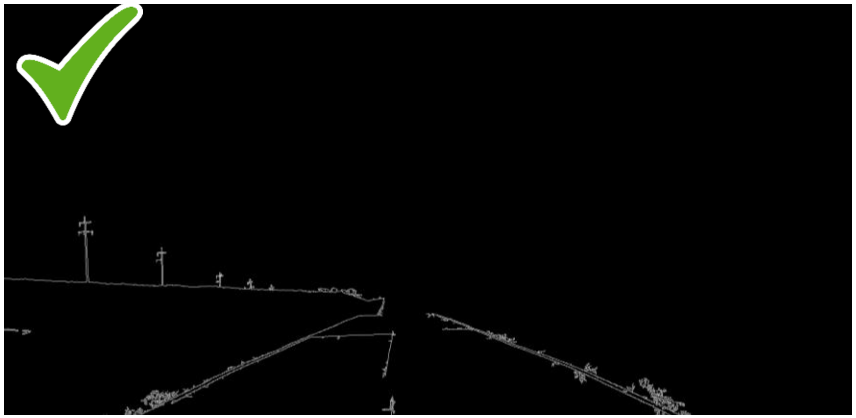
- Assign the gradient orientation α(x,y) to one of the 4 sectors
- Check the 3x3 region of each M(x,y), if the value at the center is not greater than the 2 values along the gradient, then M(x,y) is set to 0

Hysteresis Thresholding

- Gradient bigger than T_h: edge points
- Gradient between T_l and T_h : marked as edges only if connected to edge points
  - A small T_l with a larger T_h allows to find well connected edges with a few false detections due to noise

Canny: Parameters





The Canny algorithm has 3 parameters:

1. σ : Gaussian smoothing, find only large structures or small details, controls robustness to noise
2. T_l: Low threshold, keep low for edge linking
3. T_h : High threshold, sets the number of edges found (increase to avoid false detections)

Before searching lines with HT verify the edge detector output !

Canny in OpenCV

```
void cv::Canny  (  InputArray      image,
                   OutputArray     edges,
                   double          threshold1,
                   double          threshold2,
                   int             apertureSize = 3,
                   bool            L2gradient = false
                )
```

The Canny algorithm is implemented in OpenCV

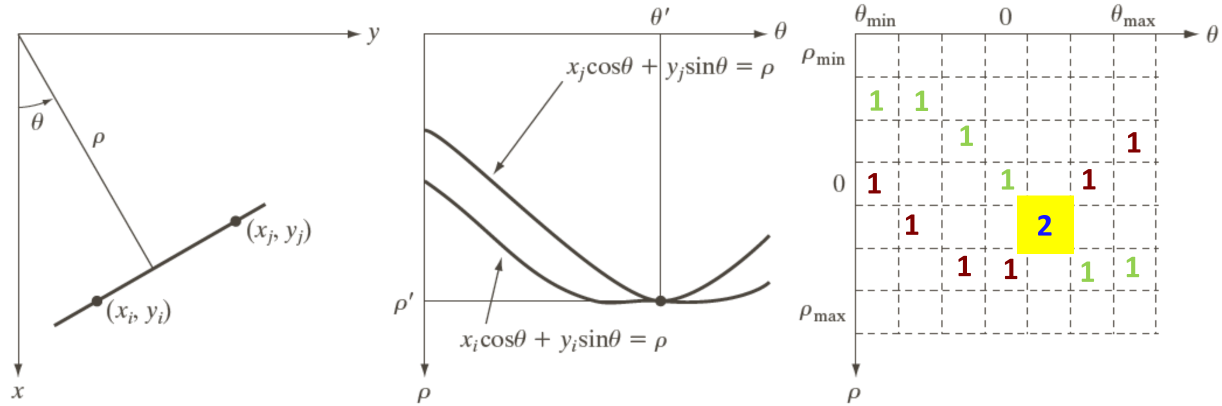
- *image: input image, typically grayscale*
- *edges: output binary image with the edge map*
- *threshold1, threshold2* $\leftrightarrow T_l, T_h$
- *apertureSize* $\leftrightarrow$ σ (smoothing implemented inside the gradient extraction)
- L2gradient : use $\sqrt{x^2+y^2}$ or approximate with $|x|+|y|$

Hough Transform

$$y_i = ax_i + b$$

$$\Downarrow$$

$$b = -x_i a + y_i$$

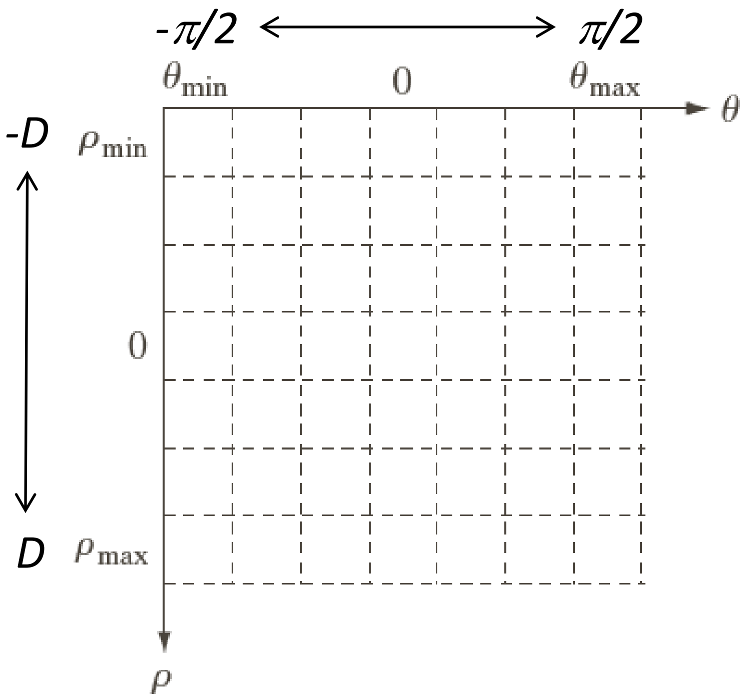


a b c

FIGURE 10.32 (a) (ρ, θ) parameterization of line in the xy-plane. (b) Sinusoidal curves in the $\rho\theta$-plane; the point of intersection (ρ', θ') corresponds to the line passing through points (x_i, y_i) and (x_j, y_j) in the xy-plane. (c) Division of the $\rho\theta$-plane into accumulator cells.

1. Compute edge detection and get edge points
2. The parameter space is quantized in cells, there is a counter for each cell
3. For each edge pixel:
   A. Let θ vary on the quantized interval $(-\frac{\pi}{2}, \frac{\pi}{2})$ and compute the corresponding ρ values
   B. For each crossed cell increment by 1 the counter
4. The counter for each cell contains the number of pixels collinear on that line
   - Use a threshold on the counter values to get the lines

Accumulation Cells



Few Large Cells	Many Small Cells
Accepts pixels not perfectly aligned	Requires precise alignment
Stable w.r.t. noise	Sensitive to noise
Poor lines localization	Accurate lines localization
Fast	Slow

- The parameter space is quantized along ρ and θ
  - ρ and θ: distance from the origin and orientation
- The cell subdivision allows to handle points not perfectly aligned
- *Cell size*: quantization of the line localization params

Example

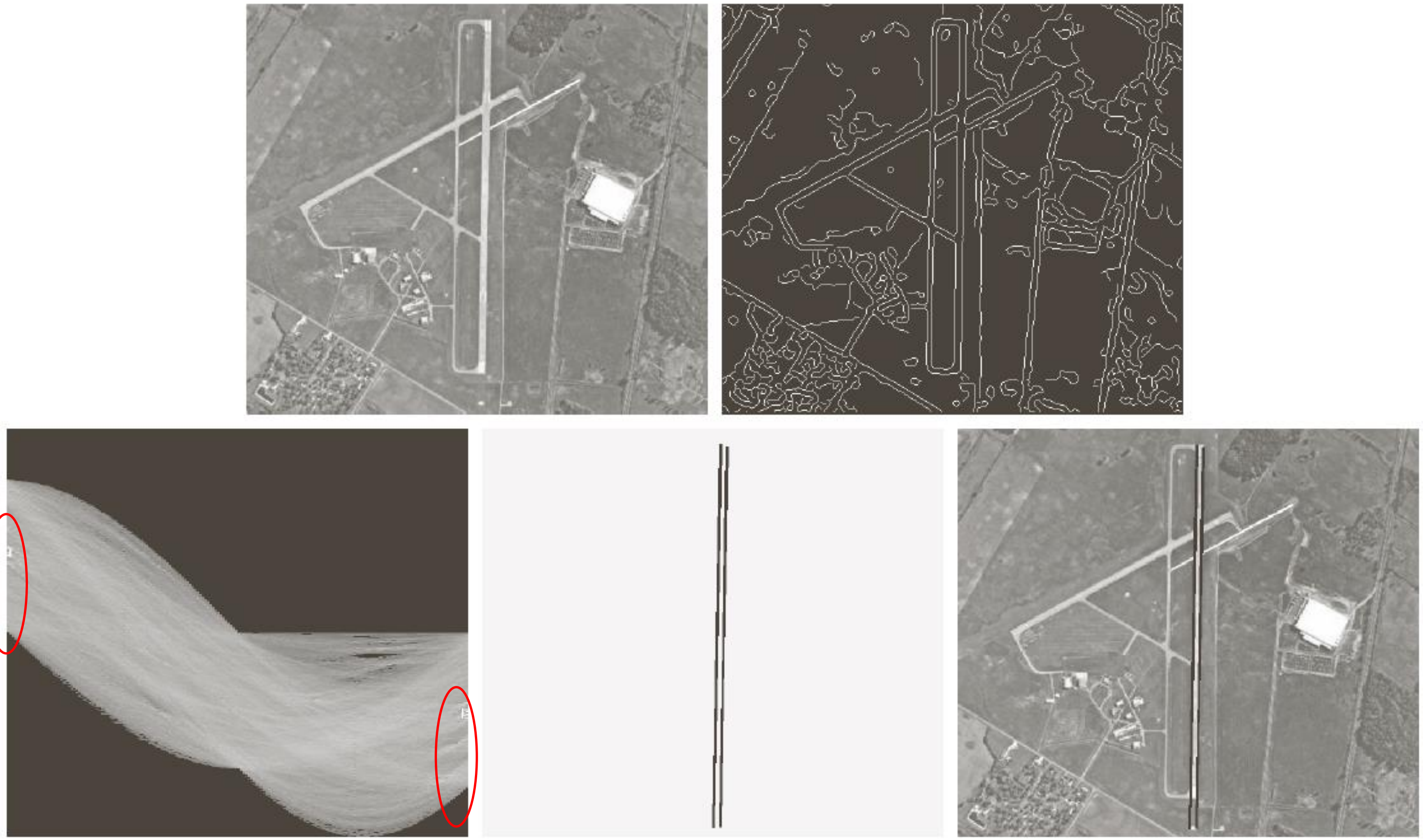
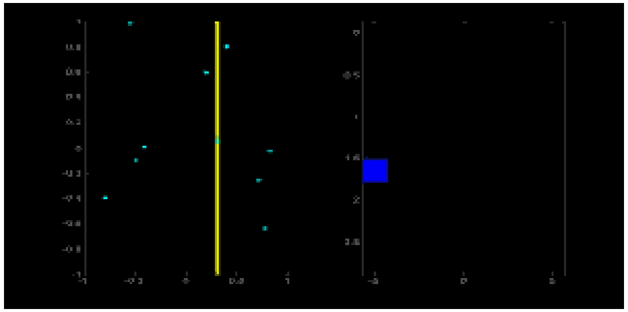
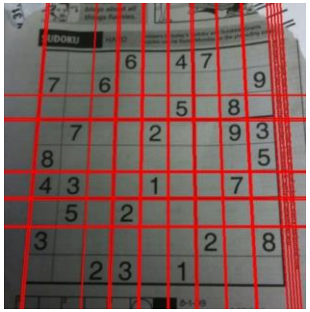


FIGURE 10.34 (a) A 502 × 564 aerial image of an airport. (b) Edge image obtained using Canny's algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes). (e) Lines superimposed on the original image.

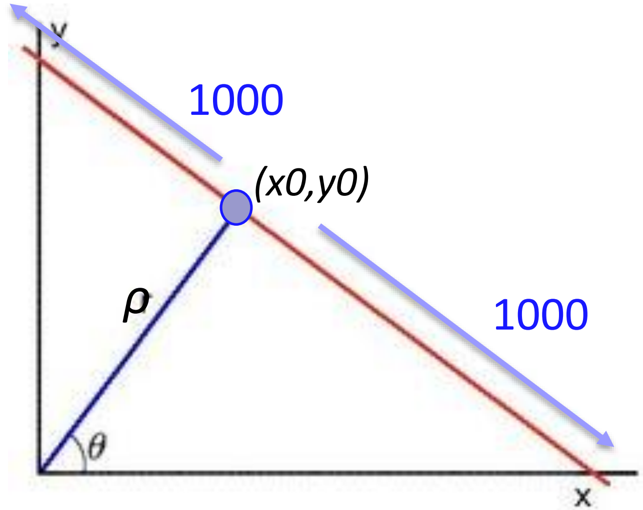
OpenCV: HoughLines





void cv::HoughLines	(	InputArray	image,	Output of edge detector
		OutputArray	lines,	Array of 2-elements vectors (ρ,θ)
		double	rho,	Cell size (ρ-dim [pixels])
		double	theta,	Cell size (θ-dim [rad])
		int	threshold,	Min # of points for line
		double	*srn = 0,*	*For multi-resolution*
		double	*stn = 0,*	*For multi-resolution*
		double	min_theta = 0,	Min angle [rad],0=vertical
		double	max_theta = CV_PI	Max angle [rad]
	)			

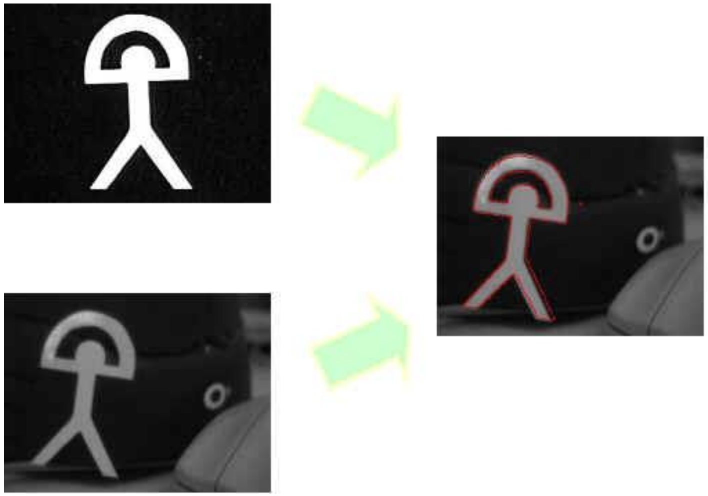
Get Line Positions and Draw Lines




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

cv::line plots a line over the image

Hough Transform for Circles



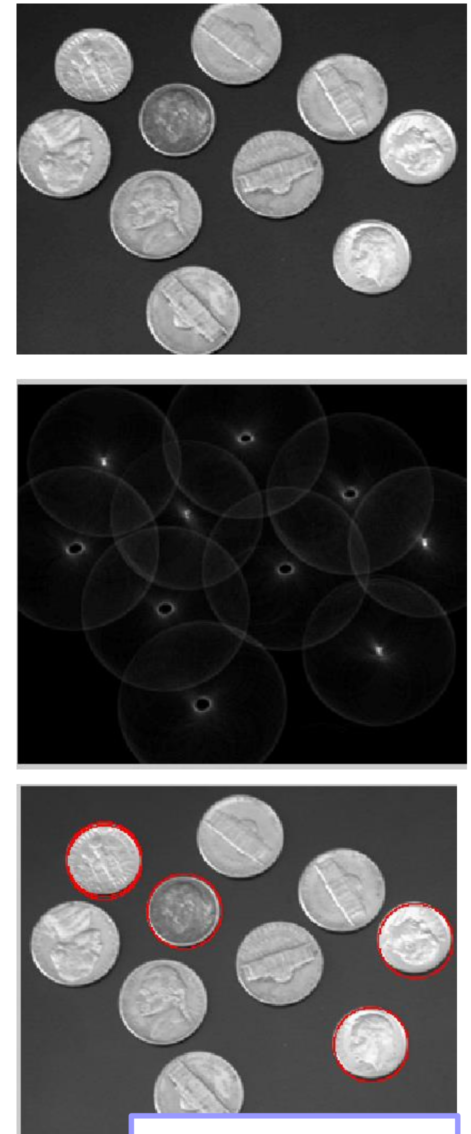
$g(\mathbf{v},\mathbf{c})=0$ $\longrightarrow$ $(x-c_1)^2+(y-c_2)^2=c_3^2$

variables

parameters

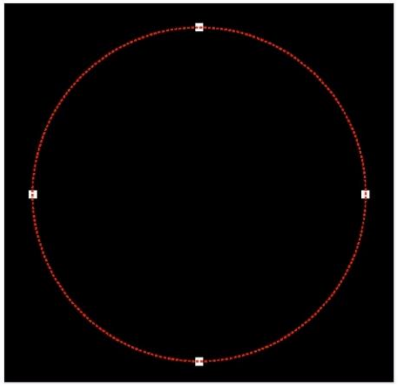
Center: (c_1, c_2)
Radius: c_3

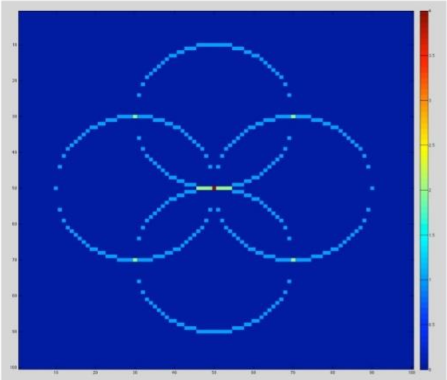
Dimensionality of vector **c** (parameter space) is 3



cv::HoughCircles

cv::HoughCircles





cv::circle plots a circle over the image

void cv::HoughCircles	(	InputArray	image,	Grayscale input image (*edge detection performed inside the function*)
		OutputArray	circles,	List of found circles: each element is a 3-element floating-point vector (x, y, radius)
		int	method,	HOUGH_GRADIENT (only available option)
		double	dp,	Image resolution / accumulator resolution (ratio)
		double	minDist,	Minimum distance between two circles
		double	param1 = 100,	*Th* of Canny (*Tl = Th / 2*)
		double	param2 = 100,	Threshold for accumulator count to detect a circle
		int	minRadius = 0,	Minimum radius
		int	maxRadius = 0	Maximum radius
	)			