Lecture #06: Edge Detection

Winston Wang, Antonio Tan-Torres, Hesam Hamledari
Department of Computer Science
Stanford University
Stanford, CA 94305
{wwang13, tantonio, hesamh}@cs.stanford.edu

1   Introduction

This lecture covers edge detection, Hough transformations, and RANSAC

2   Edge Detection

2.1   Motivation for Edge Detection

Edge detection is extremely relevant for mammalian eyes. Certain neurons within the brain are adept at recognizing straight lines. The information from these neurons is put together in the brain for recognition of objects. In fact, edges are so useful for recognition in humans, line drawings are almost as recognizable as the original image.
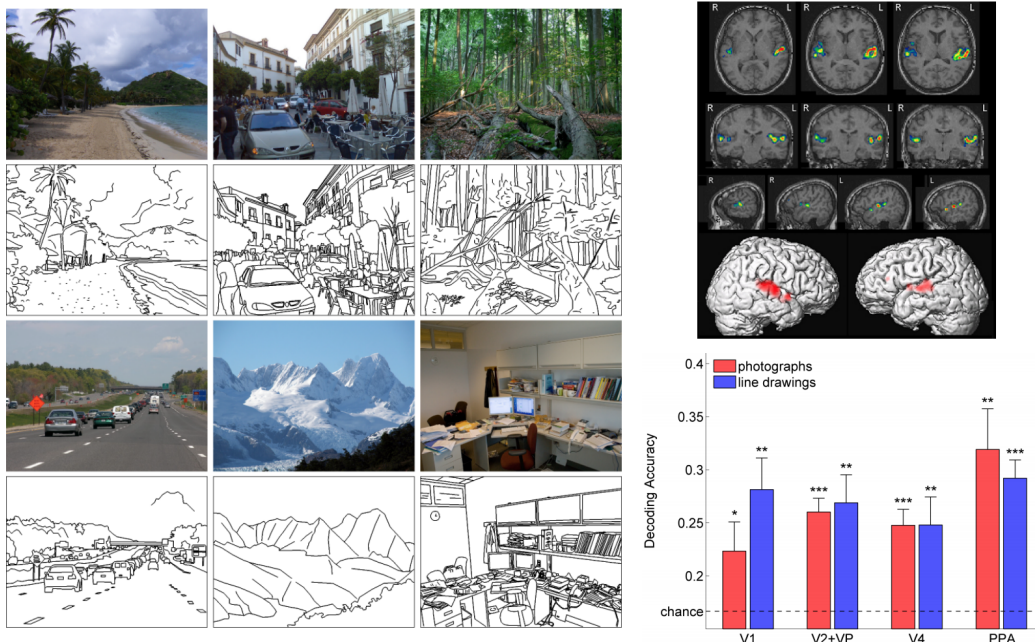

Fig. 1. Certain areas of the brain react to edges; the line drawings are as recognizable as the original image; image source: [4]

We would like to be able to extract information, recognize objects, and recover the geometry and viewpoint of an image

2.2 Edge Basics

There are four possible sources of edges in an image:

- Surface normal discontinuity
- Depth discontinuity
- Surface color discontinuity
- Illumination discontinuity

Edges occur in images when the magnitude of the gradient is high

2.3 Finding the Gradient

In order to find the gradient, we must first find the derivatives in both the x and y directions

2.3.1 Discrete Derivatives

$$\frac{dy}{dx} = \lim_{\delta x \to 0} \frac{f(x) - f(x - \delta x)}{\delta x} = f'(x)$$
$$\frac{dy}{dx} = \frac{f(x) - f(x - 1)}{1} = f'(x)$$
$$\frac{dy}{dx} = f(x) - f(x - 1) = f'(x)$$

It is also possible to take the derivative three different ways

- Backward: $f'(x) = f(x) - f(x - 1)$
- Forward: $f'(x) = f(x) - f(x + 1)$
- Central: $f'(x) = f(x + 1) - f(x - 1)$

Each of these can also be represented as a filter (convoluting the filter with the image gives the derivative)

- Backward: $f'(x) = f(x) - f(x - 1) \to [0, 1, -1]$
- Forward: $f'(x) = f(x) - f(x + 1) \to [-1, 1, 0]$
- Central: $f'(x) = f(x + 1) - f(x - 1) \to [1, 0, -1]$

The gradient (∇f) can be calculated as follows:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$
$$= \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

We can also calculate the magnitude and the angle of the gradient:

$$|\nabla f(x, y)| = \sqrt{f_x^2 + f_y^2}$$
$$\theta = \tan^{-1}(f_y / f_x)$$

2.4 Reducing noise

Noise will interfere with the gradient, making it impossible to find edges using the simple method, even though the edges are still detectable to the eye. The solution is to first smooth the image.
Let f be the image and g be the smoothing kernel. Thus, in order to find the smoothed gradient, we must calculate (1D example):

$$\frac{d}{dx}(f * g)$$

By the derivative theorem of convolution:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

This simplification saves us one operation. Smoothing removes noise but blurs edges. Smoothing with different kernel sizes can detect edges at different scales

2.5 Sobel Noise Detector

This algorithm utilizes 2 3×3 kernels which, once convolved with the image, approximate the x and y derivatives of the original image.

$$\mathbf{G}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad\qquad \mathbf{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

These matrices represent the result of smoothing and differentiation

$$\mathbf{G}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

The Sobel Filter has many problems, including poor localization. The Sobel Filter also favors horizontal and vertical edges over oblique edges

2.6 Canny Edge Detector

The Canny Edge Detector has five algorithmic steps:

- Suppress noise
- Compute gradient magnitude and direction
- Apply non-maximum suppression
- Hysteresis thresholding
- Connectivity analysis to detect edges

2.6.1 Suppress noise

We can both suppress noise and compute the derivatives in the x and y directions using a method similar to the Sobel filter.

2.6.2 Compute gradient magnitude and direction

From above,

$$|\nabla f(x, y)| = \sqrt{f_x^2 + f_y^2}$$
$$\theta = \tan^{-1}(f_y/f_x)$$

2.6.3 Apply non-maximum suppression

The purpose of this portion of the algorithm is to make sure the edges are specific. Thus, we assume that the edge occurs when the gradient reaches a maximum. We suppress any pixels that have a non-maximum gradient.
Basically, if the pixel is not the largest of the three pixels in the direction and opposite the direction of its gradient, it is set to 0. Furthermore, all gradients are rounded to the nearest $45°$

2.6.4 Hysteresis thresholding

All remaining pixels are subjected to hysteresis thresholding. This part uses two values, for the high and low thresholds. Every pixel with a value above the high threshold is marked as a strong edge. Every pixel below the low threshold is set to 0. Every pixel between the two thresholds is marked as a weak edge

2.6.5 Connectivity analysis to detect edges

The final step is connecting the edges. All strong edge pixels are edges. For weak edge pixels, only the weak edge pixels that are linked to strong edge pixels are edges. The part uses BFS or DFS to find all the edges.

3 Hough Transforms

3.1 Intro to Hough Transform

Hough Transform is a way to detect particular structures in images, namely lines. However, Hough transform can be used to detect any structure whose paramteric equation is known. It gives a robust detector under noise and partial occlusion.

3.2 Goal of Hough Transform for detecting lines

Hough transform can be used to detect lines in images. To do this, we want to locate sets of pixels that make up straight lines in the image. This works to detect lines in an image after an edge detector is applied to get the pixels of just the edges (and thus we find which sets of those pixels make up straight lines).

3.3 Detecting lines using Hough Transform in a,b space

Say we have a x_i, y_i. There are infinite lines that could pass through this point. We can define a line that passes through this pixel x_i, y_i as

$$y_i = a * x_i + b$$

. Using this, we can transform each pixel into a, b space by re-writing this equation as:

$$b = -a * x_i + y_i$$

This equation represents a line in a, b space, and each a, b point on the line represents a possible line passing through our point x_i, y_i.

Thus, for each pixel x_i, y_i in our set of edge pixels, we transform it into a, b space to get a line.
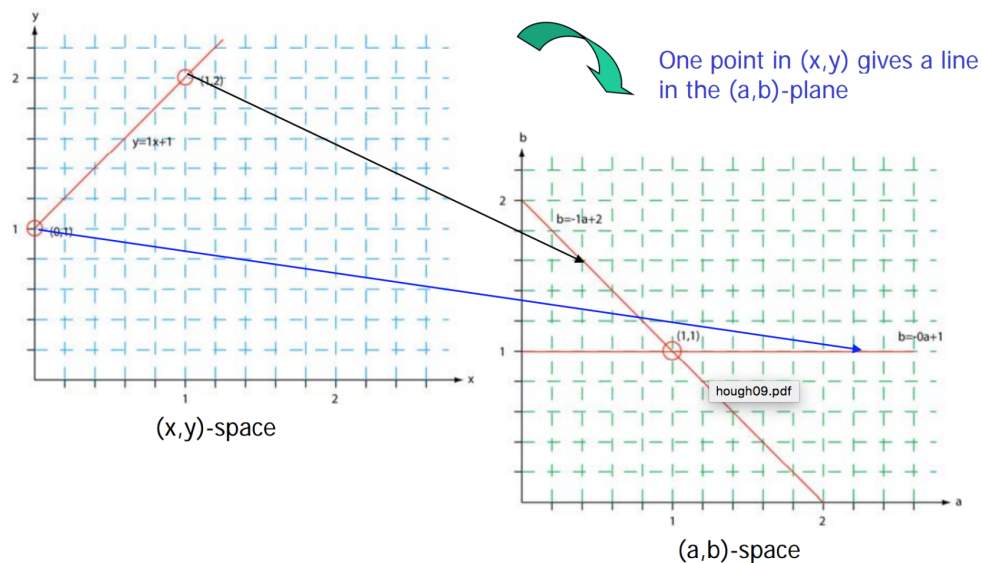

Fig. 2. The transformation from the original space to the Hough space; source: lecture slides

The intersection of lines in a, b space represent the a, b values that compromise a line $y_i = a * x_i + b$ passing through those points. Example: Say we have two points $x_1, y_1 = (1, 1)$, and $x_2, y_2 = (2, 3)$.

We transform these points into a, b space with the lines $b = -a * 1 + 1$ and $b = -a * 2 + 3$. Solving for the intersection of these two lines gives us $a = 2$ and $b = -1$. This intersection point in (a, b) space gives us the values for the line that goes through both points in x, y space.
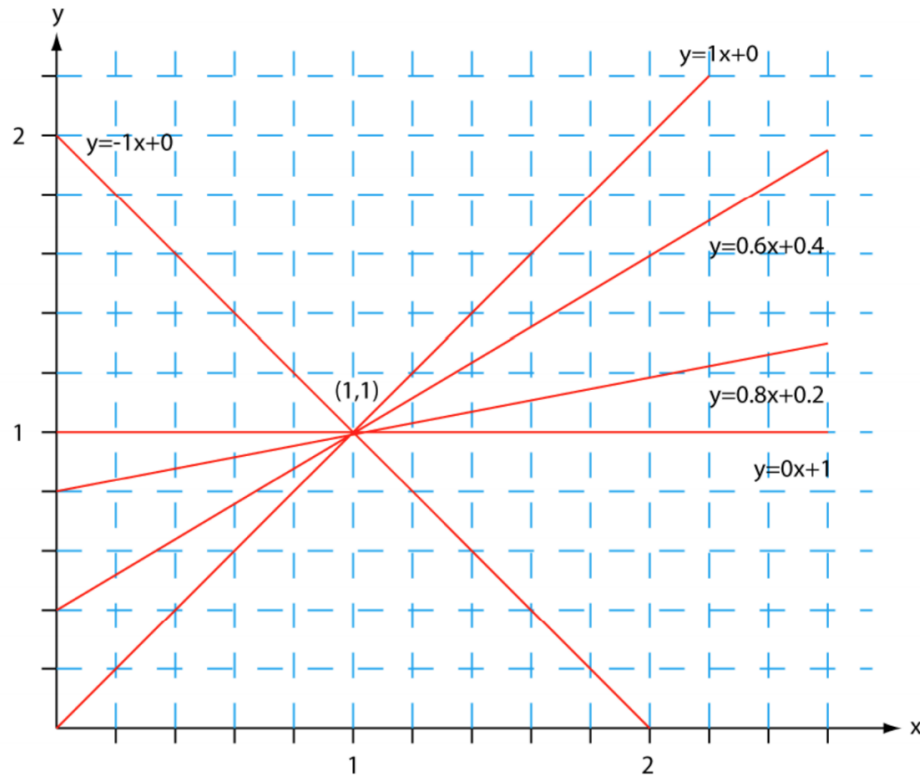

Fig. 3. The lines passing through a point in the original space; source: lecture slides

3.4 Accumulator Cells

In order to get the "best" lines, we quantize the a, b space into cells. For each line in our a, b space, we add a "vote" or a count to each cell that it passes through. We do this for each line, so at the end, the cells with the most "votes" have the most intersections and therefore should correspond to real lines in our image.

The algorithm for Hough transform in a, b space is as follows:

5

Algorithm for Hough transform

- Quantize the parameter space (a b) by dividing it into cells
- This quantized space is often referred to as the accumulator cells.
- Count the number of times a line intersects a given cell.
  - For each pair of points (x_1, y_1) and (x_2, y_2) detected as an edge, find the intersection (a',b') in (a, b) space.
  - Increase the value of a cell in the range $[[a_{min}, a_{max}],[b_{min}, b_{max}]]$ that (a', b') belongs to.
  - Cells receiving more than a certain number of counts (also called 'votes') are assumed to correspond to lines in (x,y) space.

Fig. 4. The Hough transform algorithm; source: lecture slides

3.5 Hough transform in ρ, θ space

A problem with using a, b space to represent lines is that they are limited and cannot represent vertical lines. To solve this, we use polar coordinates to represent lines. For a pixel x_i, y_i, we transform it using the equation:

$$x * \cos\theta + y * \sin\theta = \rho$$

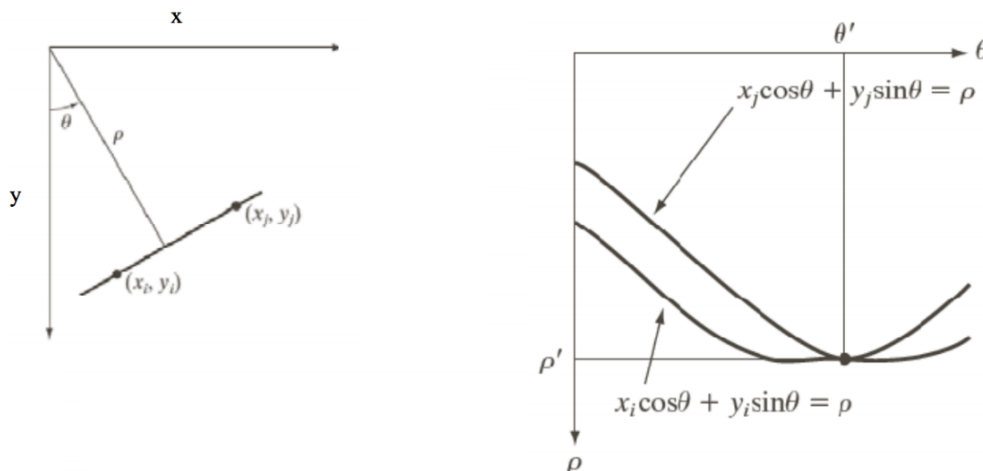In ρ, θ space, points are not represented as lines but instead as sine wave-like functions.

Fig. 5. The Hough transform in ρ, θ space; source: lecture slides

The intersection of these functions in ρ, θ space still correspond to the ρ, θ that comprise a line passing through those points.

So, for each pixel x_i, y_i we transform it into a function in ρ, θ space. We apply the same accumulator cell algorithm to count the most intersections of functions.

In this case, we quantize our ρ, θ space into cells, and add a "vote" to each cell that our function passes through. The cells with the most votes are the most likely real lines in our image.

3.6 Concluding Remarks

Advantages of the Hough Transform is that it is conceptually simple (just transforming and finding intersection in Hough space). It is also fairly easy to implement, and it can handle missing and occluded data well. Another advantage is that it can find other structures other than lines, as long as the structure has a parametric equation.

Some disadvantages include that it gets more computationally complex the more parameters you have. It can also only look for one kind of structure (so not lines and circles together). The length and the position of a line segment can also not be detected by this. It can be fooled by "apparent" lines, and co-linear line segments cannot be separated.

4 RANSAC

With the increase in model complexity (i.e., the number of parameters), the Hough transform loses its effectiveness; this section elaborates on the design of the RAndom Sample Consensus (RANSAC) technique [1] which provides a computationally efficient means of fitting models in images. We begin with an introduction of the RANSAC's basic idea and then introduce its algorithm.

4.1 Introduction to RANSAC Basics

The RANSAC algorithm is used for estimating the parameters of models in images (i.e., model fitting). The basic idea behind RANSAC is to solve the fitting problem many times using randomly selected minimal subsets of the data and choosing the best performing fit. To achieve this, RANSAC tries to iteratively identify the data points that correspond to model we are trying to fit.

Fig. 6a illustrates an example where a linear model (i.e., 2 parameters) is to be fitted to the data; while the majority of data points fit a linear model, the two points in the top right corner can significantly affect the accuracy of overall fit (if they are included in the fit). The RANSAC algorithm aims to address this challenge by identifying the "inliers" and "outliers" in the data.

RANSAC randomly selects samples of the data, with the assumption that if enough samples are chosen, there will be a low probability that at all samples provides a bad fit.
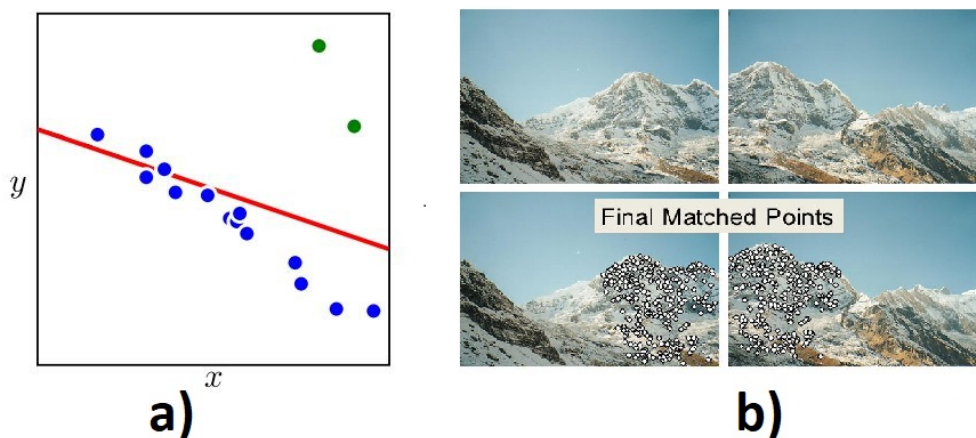

Fig. 6. a) the outliers are detected by RANSAC to improve parameter estimation; b) RANSAC application for image stitching; sources([3]and Derek Hoiem)

4.2 Applications

The RANSAC algorithm can be used to estimate parameters of different models; this is proven beneficial in image stitching (Fig. 6b), outlier detection, lane detection (linear model estimation), and stereo camera calculations.

4.3 The Algorithm

The RANSAC algorithm iteratively samples nominal subsets of the original data (e.g., 2 points for line estimation); the model is fitted to each sample, and the number of "inliers" corresponding to this fit is calculated; this includes the data points that are close to the fitted model. The points closer than a threshold (e.g., 2 standard deviation, or a pre-determined number of pixels) are considered "inliers". The fitted model is considered good if a big fraction of the data is considered as "inliers" for that fit. In the case of a good fit, the model is re-fitted using all the inliers, and the outliers are discarded. This process is repeated, and model estimates with a big enough fraction of inliers (e.g., bigger than a pre-specified threshold) are compared to choose the best-performing fit. Fig. 7 illustrates this process for a linear model and its three samples. The third sample (Fig. 6c) provides the best fit as it includes the most number of inliers.
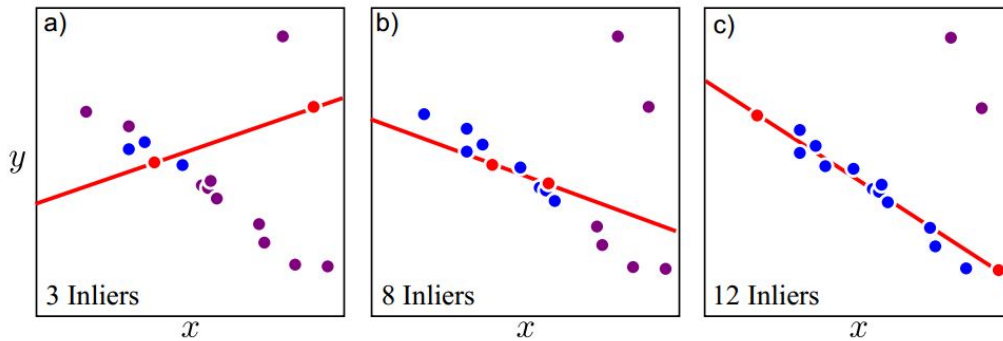

Fig. 7. The demonstration of the RANSAC algorithm for a linear model estimation and three random samples; source([3])

```
Determine n, t, d and k as above
Until there is a good fit or k iterations have occurred
    draw a sample of n points from the data
    uniformly and at random

    fit to that set of n points

    for each data point outside the sample

      test the distance from the point to the line
      against t; if the distance from the point to the line
      is less than t, the point is close

    end
    if there are d or more points close to the line
    then there is a good fit.  Refit the line using all
    these points, and terminate
end
```

Fig. 8.
The pseudocode for RANSAC; source([2])

The RANSAC is detailed in Fig. 8. The major steps included in the RANSAC loop:

1. Randomly select a seed group from data.
2. Perform parameter estimation using the selected seed group.
3. Identify the inliers (points close to the estimated model).
4. (If there exists a sufficiently large number of inliers,)re-estimate the model using all inliers.
5. repeat steps 1-4 and finally keep the estimate with most inliers and best fit.

4.4 How Many Samples are Needed?

The RANSAC is a non-deterministic model fitting approach; this means that the number of samples need to be large enough to provide a high confidence estimate of parameters. The number of required samples depends on 1) the number of fitted parameters and 2) the amount of noise. Fig. 9 lists the minimum number of samples needed based on p=0.99 and variations of sample size (i.e., the number of parameters) and the fraction of outliers (i.e., noise). More samples are needed for estimating bigger models and noisier data. A large enough number of samples (k) need to be chosen such that a low probability of only seeing bad samples (P_f) is guaranteed:

$$P_f = (1 - W^n)^k = 1 - p$$

where W and n are respectively the fraction of inliers and the number of points needed for model fitting. The minimum number of samples:

$$k = \frac{log(1 - p)}{log(1 - W^n)}$$

RANSAC: Computed k (p=0.99)

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Fig. 9. The number of samples for different choices of noise population and model size; source: David Lowe)

4.5 Advantages, Limitations, and Considerations

The advantages of the RANSAC lie in its simple implementation and wide application range in the model fitting domain. Other advantages include its computational efficiency; the sampling approach provides a better alternative to solving the problem for all possible combinations of features.

In some cases, it'd be more efficient to use the Hough transform instead of the RANSAC:

1. 1) The number of parameters are small; for example, linear model estimation (2 parameters) can be achieved efficiently using Hough transform, while image stitching requires a more computationally frugal approach such as RANSAC.

2. 2) If the noise population is high; as we saw earlier, increase in noise requires a more extensive sampling approach (higher number of samples), increasing computation cost. Increased noise reduces the chances of correct parameter estimation and the accuracy of inlier classification.

The poor performance in highly noisy data is a primary limitation of the RANSAC; this is espcially crucial as real-world problems have high rate of outliers.

References

[1] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[2] David Forsyth and Jean Ponce. *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.

[3] Simon JD Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[4] Dirk B. Walther, Barry Chai, Eamon Caddigan, Diane M. Beck, and Li Fei-Fei. Simple line drawings suffice for functional mri decoding of natural scene categories. *Proceedings of the National Academy of Sciences*, 108(23):9661—-9666, 2011.