Detecting Lines by Hough Transform

Lab Exercise 4 for Beeldverwerken

Informatics Institute
University of Amsterdam
The Netherlands

May 1, 2018

Abstract

The aim of this laboratory exercise is to experiment with edge detection and implement the Hough transform for finding lines in images. This will enable automation of some user-interactive steps in the earlier assignments.

The core Sections 2 and 3 of this assignment were gratefully taken from Laboratory 6 exercise of Computer Vision CITS4240, from the School of Computer Science at The University of Western Australia.

1 Straight Edge Detection

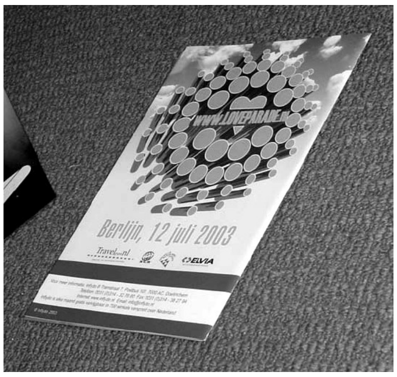
In assignment 1 we tried to reproject a rectangular object that had been imaged at an angle (Figure 1a), and therefore had become a general quadrilateral. You implemented the core as a determination of a projective transformation in the homogeneous coordinate framework. But you still had to denote by hand where the corner points were that formed the input of the algorithm.

In another part of that assignment, you produced a pseudo-3D image, continuing a calibration cube with a checkerboard into (pseudo)-space (Figure 1b). Again, the crucial points were indicated by hand (or given in a file).

In assignment 2, we learned about the Canny edge detector. But if you can find edges in an image, you would expect that this would enable you to automate the interfaces of the previous assignments. We are going to do that by detecting significant straight edges in the images, using the Hough transform. This will detect straight stretches of edge points.

2 Finding Lines using the Hough Transform

The Hough transform can be used to find sets of straight lines of edge pixels in an edge image. **(15 points)** Write a function, `hough` that meets the specification in Listing 1.


(a)

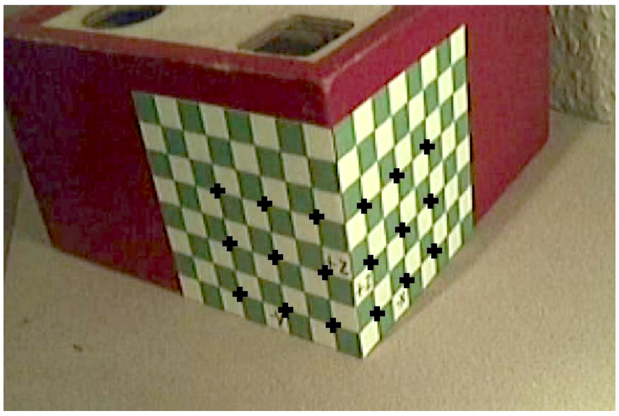

(b)

Figure 1: Images from the first assignment: (a) a straightening task; (b) a pseudo-3D task.

Before you start heed this warning! In all that follows beware of getting confused between x and y, and row and column number.

- An x value corresponds to a *column* number
- A y value corresponds to a *row* number

If you find that your code does not work read this warning again! The steps you should follow:

⋆ Perform edge detection. You can either use your own Canny edge detector or MATLAB's `edge` function (see the help page for `edge`), with the `'canny'` option, to generate an edge image from the image you want to process. (You may need to experiment with the hysteresis threshold values to get a reasonable result, but you do not need to get only the points on the edges you are interested in - the Hough transform method is going to do the grouping).

⋆ Use the `zeros` function to construct an empty accumulator array of size `nrho * ntheta`.

⋆ Now for each non-zero point (x_i, y_i) [1] in the edge image, we can ask what lines pass through that point. All such lines will satisfy an equation of the form

$$x_i \sin(\theta) - y_i \cos(\theta) = \rho \tag{1}$$

where θ is the orientation of the line and ρ is its distance from the origin. The Hough transform divides the (ρ, θ)-parameter space into so-called *accumulator cells*, whose values range over the expected values of the orientation

$$0 \leq \theta < \pi, \qquad -D \leq \rho < D,$$

where D is the distance across the diagonal of the image).

[1]Hint: use `[y,x] = find()` in Matlab

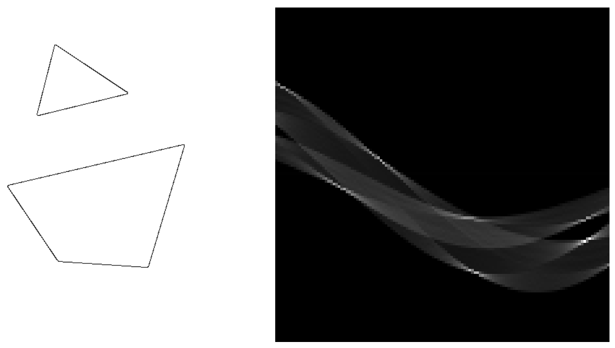


Figure 2: A typical Hough transform.

Theory Questions

2.1. **(3 points)** Study Section 8.3 of the Lecture Notes to understand the principle behind this equation. You will note that the parametrization there (in 8.3.1, item 3) is slightly different than the one here. Still, they describe the same problem - what is the correspondence? (Hint: is it merely Matlab coordinates, or is there a geometrical difference?)

2.2. **(3 points)** Why can ρ be negative, and how is this related to the range of θ? To get a better feeling for what is going on, you can play with the Hough transform using an applet like `http://www.rob.cs.tu-bs.de/content/04-teaching/06-interactive/HNF.html`.

⋆ Then for each edge point (x_i, y_i) in the image we let the parameter θ equal each of the allowed discretized values of θ, and solve for the corresponding ρ using (1).

The resulting ρs are then rounded off to the nearest allowed discretised value in the ρ axis. This will give us a locus of points lying approximately on a sinusoidal curve, for each edge point (x_i, y_i).

The accumulator array should have all its entries corresponding to the discretised ρ and θ values that make up this sinusoidal curve incremented by 1.

At the end of this procedure, a value of $M(i, j)$ in this accumulator array corresponds to M points in the xy plane that lie on the line

$$x \sin(\theta_i) - y \cos(\theta_i) = \rho_j$$

The dominant straight lines in the image then correspond to the local maxima in the accumulator array.

⋆ **(4 points)** Perform the hough transform on some images, including the simple shapes.

Your Hough Transform should look something like Figure 2. The fiddly bit of this code is getting the conversion between ρ and θ and the corresponding row and column number in the accumulator array correct. There is some code in Listing 1 to save you some grief!

Listing 1: Hough Function

```
function h = hough(im, Thresh, nrho, ntheta)
% HOUGH
%
% Function takes a grey scale image, constructs an edge map by applying
% the Canny detector, and then constructs a Hough transform for finding
% lines in the image.
%
% Usage:  h = hough(im, Thresh, nrho, ntheta)
%
% arguments:
%           im     - The grey scale image to be transformed
%           Thresh - A 2 -vector giving the upper and lower
%                    hysteresis threshold values for edge()
%           nrho   - Number of quantised levels of rho to use
%           ntheta - Number of quantised levels of theta to use
%
% returns;
%           h      - The Hough transform
% ...
rhomax = sqrt(rows^2 + cols^2);  % The maximum possible value of rho.
drho =  2*rhomax/(nrho-1);       % The increment in rho between successive
                                 % entries in the accumulator matrix.
                                 % Remember we go between +-rhomax.

dtheta = pi/ntheta;              % The increment in theta between entries.
thetas = [0:dtheta:(pi-dtheta)]; % Array of theta values across the
                                 % accumulator matrix.
% ...
% for each x and y of nonzero edge values:
  % for each theta in thetas:
    % rho = evaluate (1)
    % To convert a value of rho or theta
    % to its appropriate index in the array use:
    rhoindex = round(rho/drho + nrho/2);
    thetaindex = round(theta/dtheta + 1);
```

3 Finding the Lines as Local Maxima

Having calculated the Hough Transform we now want to find what the dominant lines are and overlay them in the image.
(12 points) Write a function with the specification in Listing 2. What this code has to do is find

Listing 2: Houghlines Function

```
function  houghlines(im, h, thresh)
% HOUGHLINES
%
% Function takes an image and its Hough transform, finds the
% significant lines and draws them over the image
%
% Usage:  houghlines(im, h, thresh)
%
% arguments:
%         im     - The original image
%         h      - Its Hough Transform
%         thresh - The threshold level to use in the Hough Transform
%                  to decide whether an edge is significant

...

for n = 1:nregions
   mask = bwl == n;       % Form a mask for each region.
   region = mask .* h;    % Point-wise multiply mask by Hough Transform
                          % to give you an image with just one region of
                          % the Hough Transform.
   ...
end
```

the local peaks within the Hough transform.

- ⋆ Recalculate `rhomax` from the size of the image and, from the number of rows and columns in `h` recalculate `drho` and `dtheta`
- ⋆ Threshold the Hough Transform at your specified threshold value. This should give isolated regions that contain within them a peak indicating a maximal 'vote' for a line.
- ⋆ Use `bwlabel` to form labeled connected components of this thresholded image
- ⋆ Use each labeled region as a 'mask' to isolate individual regions from the original Hough Transform image
- ⋆ Within each isolated region find the coordinates of the maximum value. These coordinates represent the equation of a dominant line. Convert these coordinates to a `rho` and `theta` value by 'inverting' the expressions given earlier.

  Note that you can use MATLAB's `max` function to find the indices of the peak. If matrix `X` is 2D it will return an array `maxval` giving the maximum value in each column of `X` and the second returned value will be an array giving the row where the maximum value was in each column. A second call to `max` on the array `maxval` will identify the absolute maximum

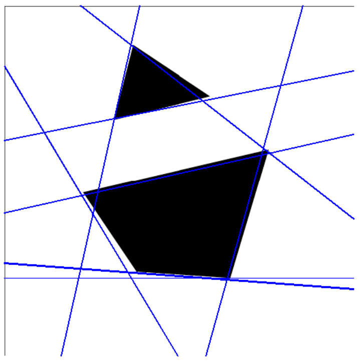


Figure 3: Output of `houghlines`

value in `X` and the column in which it occurs. Don't forget to convert these indices of the maxima back to θ and ρ coordinates.

- ⋆ To draw the lines, define a separate function

  ```
  [x1, y1, x2, y2] = thetarho2endpoints(theta, rho, rows, cols)
  ```

  Evaluate the value of `y` at `x=0` and `x=cols` from Equation 1. This would give you the coordinates x_1, y_1, x_2, y_2 to pass to the `line` function. If you have lines that are almost vertical, then set `y=0` and `y=rows` and solve for `x` from the equation instead.

  You should be able to make your Matlab script to handle the plotting of both vertical and horizontal lines automatically. Your output might look something like Figure 3.

- ⋆ Make your function return the relevant line parameters in homogeneous coordinates. You are going to have to work with these lines later, to provide better estimates. When you have two points on a line (x_1, y_1, x_2, y_2), it is easy to get the homogeneous line coordinates using the cross product. So the function will be:

  ```
  function [lines] =  houghlines(im, h, thresh)
  ```

  Where `lines` is an N-by-3 array of homogeneous line coordinates.

- **(3 points)** An alternative (and probably better) way of picking out the peaks in the Hough transform would be to use a dilation. Add this functionality to your `houghlines` function.

  This method was originally used for non-maximal suppression of corner strength images. What this code does is a greyscale dilation (look it up in wiki, it is related to Chapter 7 of the Lecture Notes) of the image (within the structuring element area every pixel is replaced with the maximum pixel value within the structuring element area). Where the greyscale dilated image matches the original image will correspond to local maxima. The size of your structuring element determines how close together your maxima are allowed to be.

4 Refining Hough

There is plenty of room for refinements. The Hough Transform 'surface' tends to be very rough. This is partly caused by the primitive way in which we 'draw' the sinusoidal votes into it. Note that if for some increment in θ the ρ value changes by two or more we should be 'drawing in' extra points to ensure that we do not skip the intermediate ρ values. One can get some improvement by smoothing the transform very slightly.

Theory Questions

4.1. **(0 points)** Most people would just do some Gaussian blur on the (ρ, θ) domain, without thinking. We would not do that, would we?

Consider a point at (x, y) in the original image. Assume that it has some Gaussian noise with standard deviation σ. For a fixed θ, what is the effect on the curve of Equation 1? You may most easily see that from a sketch of the geometry. The effect you observe determines what the proper filtering in the Hough domain should be!

4.2. **(0 points)** Another thing that is an easy improvement is to let the strength of the edge detector affect what you plot as the result of an edge point at (x, y) in the Hough domain. What would be a good measure of the significance of a detected edge?

4.3. **(0 points)** More sophisticated, you can know the direction of the gradient; therefore you can know pretty well what θ will result at that local edge point. How can you use that to 'focus' the transformation? (Note that this includes knowing the light/dark sides, and prevents an ambiguity of π in θ.)

Some morphological processing (Lecture Notes Chapter 7, which we skipped this year) may also be useful after the thresholding process.

(0 points) Actually implement some or all of the above-mentioned refinements.

5 Optimal Line Estimation

The Hough method has returned an indication of where the dominant lines in the image are. But because of reasons of computational efficiency, we had to use rather coarse bins for the line parameters ρ and θ. You should really view the Hough step as a grouping, but not as an estimation. This was also mentioned in Step 5 and 6 of Section 8.3.2 in the Lecture Notes.

So, use the outcome to select which points you want to group into a line, and from those determine the best line using a *least-square-fit*.

⋆ **(4 points)** Write a function that collects the points close to a line found by the Hough transform, within a distance of `epsilon`.

```
function [pts] = points_of_line[points, line, epsilon]
%    points  - an array containing all points
%    line    - the homogeneous representation of the line
%    epsilon - the maximum distance
% returns:
%    pts     - an array of all points within epsilon of the line
```

Note that it is really easy to find the distance between a line and a point if they are both in normalized homogeneous representation.

- ⋆ **(6 points)** Write a function

  ```
  function l = line_through_points(points)
  % returns:
  %    l       - homogeneous representation of the least-square-fit
  ```

  This is not quite least squares fit of a linear *function* as we treated in linear algebra. In that fit, you would have a set of axes, and typically minimize the sum of squared differences between a linear function $y = \alpha\, x + \beta$ and data points (x_i, y_i) by choosing α and β appropriately. But those differences are in the y-direction.

  We want to find the best line such ℓ such that the sum of squared distances of the data points $\mathbf{p}_i = (x_i, y_i)$ to this line ℓ are minimized - but those distance should be measured geometrically, perpendicular to the line. That is different!

  The solution can be derived, and we give it. First compute the *centroid* $\mathbf{c}$ of the data (the average of all data points). Then consider the distribution of the vectors $\mathbf{p}'_i = \mathbf{p}_i - \mathbf{c}$; the optimal direction of the line turns out to be the direction of largest covariance of this set $\{\mathbf{p}'_i\}$. So, make the *covariance matrix* of this set (remember your Statistics class, or read Section 4.1.2 of the Lecture Notes), and use Matlab to compute the eigenvector $\mathbf{u}$ of largest eigenvalue.

  Now you have found the optimal line ℓ: it passes through $\mathbf{c}$ and has direction $\mathbf{u}$. Convert it to your favorite line characterization (in what follows, we want the line as a vector in homogeneous coordinates, so that is a good choice now).

- If you did not determine the end points in the Hough transform, you could do so now.

- You may even use the RANSAC method to make the method more robust, deciding which of the points in the selected group used for the fit of each line are outliers.

6 Using the Lines

6.1 Better Straightening

In the straightening example, you have thus found 4 optimal lines using Hough followed by LSQ. Now you can use those as the automatic input to your straightening algorithm of the first assignment (Figure 1a).

Two methods come to mind. Both involve homogeneous coordinates of lines, explained in the BB writeup on Homogeneous Coordinates, Section 5.

- ⋆ **(4 points)** Method 1: you can use pairs of lines to produce their intersection. In 2D, there is a nice trick. If you have represented the lines using homogeneous coordinates as 3D (co)vectors ℓ_1 and ℓ_2, then the point of intersection is located at the homogeneous point $\ell_1 \times \ell_2$ (where $\times$ denotes the cross product, in Matlab `cross`). Implement this.

- **(2 points)** So: *intersection of 2D lines is taking the cross product of their homogeneous representatives!* Moreover, the homogeneous weight of the vector representing the intersection point is proportional to the sine of the angle between the two lines. Try to prove these facts.

- **(2 points)** The converse is also interesting: if you have two 2D points $\mathbf{p}_1$and $\mathbf{p}_2$ in their homogenous representation, $\mathbf{p}_1 \times \mathbf{p}_2$ is the homogeneous coordinate representation of the line connecting them. Prove this; and determine the geometric meaning of the homogeneous weight.

- **(4 points)** The second method is to use the estimated lines immediately, because the 4 lines you found also determine the projective transformation. Read Section 5 of the Homogeneous Coordinates writeup to see how this can be done, and implement it.

⋆ **(6 points)** Use the Hough transform to detect lines and straighten some of the images in the attachment. For some you might need the refinements in the previous section. It might require some tweaking of the thresholds and other parameters to get this right. If it doesn't work, explain why you think this is.

6.2 Better Pseudo-3D

You can of course also use this on the image (Figure 1b) of the calibration cube we used in the Pseudo-3D assignment. But you get many lines, and in order to do the addressing of the pseudo-3D-coordinates properly, you need to put them in the proper order. That is a rather tricky bit of coding, especially if you want to do it for general positions of the camera relative to the cube. But it can be done, in principle. We'll save you the trouble. We list 0 points for this exercise, but any reasonable attempt can get you bonus points. **(0 points)**
MAX POINTS: 68