# The Report of MP6

## I. Basic Concepts

### 1.1 Hough Transform

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the classical Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc. A generalized Hough transform can be employed in applications where a simple analytic description of a feature(s) is not possible. Due to the computational complexity of the generalized Hough algorithm, we restrict the main focus of this discussion to the classical Hough transform. Despite its domain restrictions, the classical Hough transform (hereafter referred to without the classical prefix) retains many applications, as most manufactured parts (and many anatomical parts investigated in medical imagery) contain feature boundaries which can be described by regular curves. The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise.

### 1.2 Algorithm and Implementation

The Hough technique is particularly useful for computing a global description of a feature(s) (where the number of solution classes need not be known a priori), given (possibly noisy) local measurements. The motivating idea behind the Hough technique for line detection is that each input measurement (e.g. coordinate point) indicates its contribution to a globally consistent solution (e.g. the physical line which gave rise to that image point).
Consider the common problem of fitting a set of line segments to a set of discrete image points. We can analytically describe a line segment in a number of forms. However, a convenient equation for describing a set of lines uses parametric or normal notion is shown as followed:

$$r = x cos\theta + y sin\theta$$

where $r$ is the length of a normal from the origin to this line and $\theta$ is the orientation of $r$ with respect to the X-axis. Obviously, for any point (x,y) on this line, $r$ and $\theta$ are constant.
In an image analysis context, the coordinates of the point of edge segments $(x_i, y_i)$ in the image are known and therefore serve as constants in the parametric line equation, while $r$ and $\theta$ are unknown variables we look for. If we plot the possible $(r, \theta)$ defined by each $(x_i, y_i)$ points in Cartesian image space map to curves in the polar Hough parameter space. This point-to-curve transformation is the Hough Transformation for straight lines. When viewed in Hough parameter space, points which are collinear in the Cartesian image space become readily apparent as they yield intersect at a common $(r, \theta)$ point.
The transform is implemented by quantizing the Hough parameter space into finite internals or accumulator cells. As the algorithm runs, each $(x_i, y_i)$ is transformed into a discretized $(r, \theta)$ curve and the accumulator cells which lie along this curve are incremented. Resulting peaks in the accumulator array represent strong evidence that a corresponding straight line exists in the image.

In this assignment, I get the edges of the input image using Canny edge detector at first. Then, I set up two matrices for rho and theta. Next, I compared such two matrices with other possible line segments in the image. The most common line segments found in the image are increased, which is a process of 'voting'.
In addition, the local maxima from the voting process are collected as candidates. These candidates are compared against the provided threshold, and those below the threshold are discarded, The remaining $(r, \theta)$ pairs are likely candidates for lines, and are subsequently drawn over the edge image after a conversion back to Cartesian coordinates.

### 1.3 The purpose of the functions

This project requires us to implement the Hough Transform Detector. The point of this algorithm is to detect the line segments that make up geometric figures in the given image.
In this homework, I design the function of Hough transform as following:
[found_rho, found_theta, accumulator] = houghline(img_in, rho_step, theta_step, threshold)
where the inputs are image_in, rho_step, theta_step and provided threshold.

## II. Results and Analysis

### 2.1 Test Results

The results of the three given images are shown as followed.

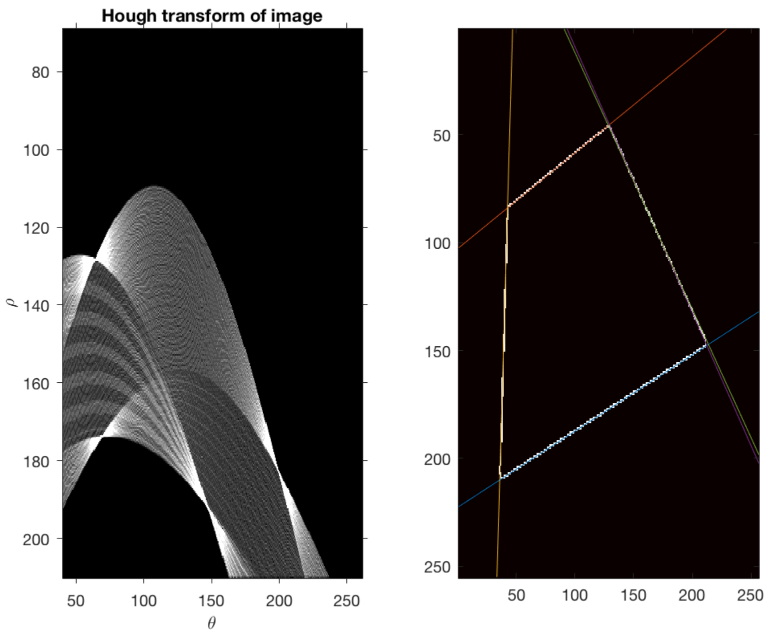

*Figure 1 test.bmp result (1,0.5,80)*

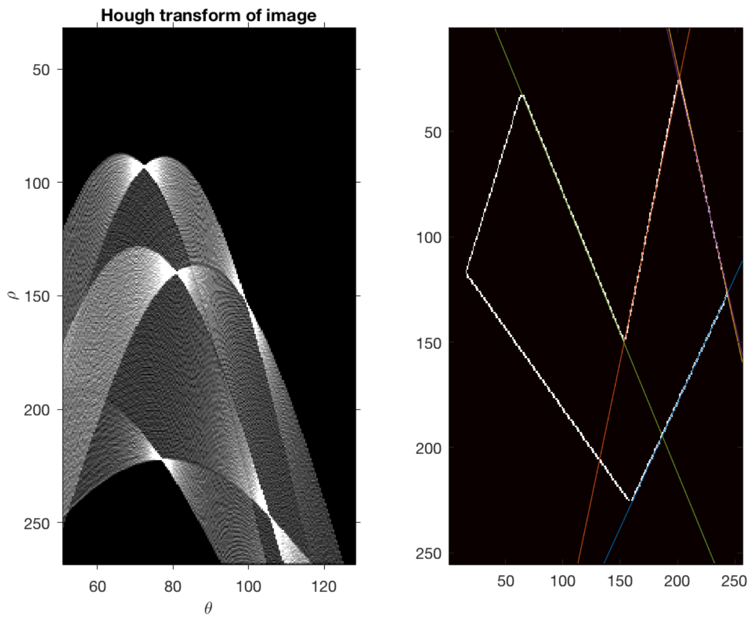

*Figure 2 Test2.bmp result (1,1,81)*

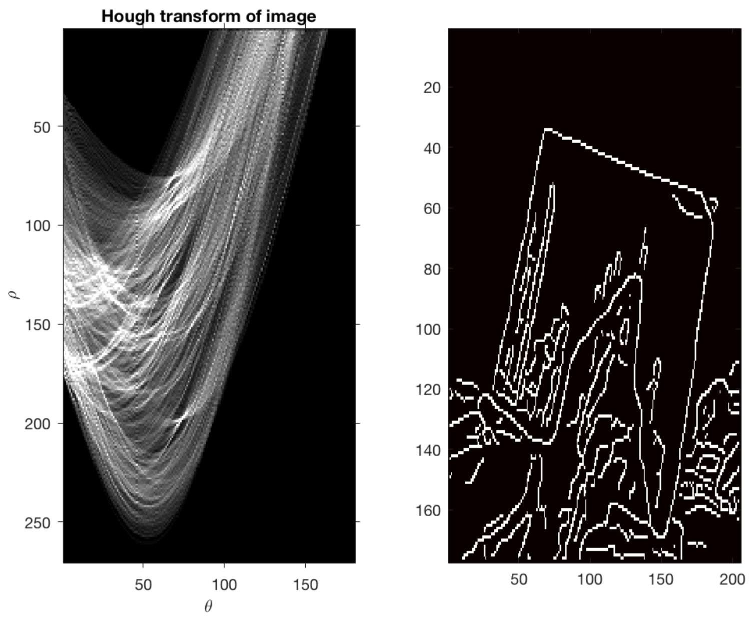

*Figure 3 Input.bmp (1,1,78)*

## 2.2 Result Analysis

From the results above, it could be safely drawn a conclusion that such Hough Transform Model is implemented well and the results are persuasive. The Hough Transform Line Detector in my project is able to find the shapes of the three basic images very well, except for the input.bmp. The reason might be the problem from segmentation of the image and edge detection. And I also test my model via playing different parameters. From the test, I found that when $(r, \theta)$ equals to (1,1), the results can be defined 'best' results.

### 2.3 Summary

Through such MP6, I got a better understanding of the core of Hough Transform Line Detector. And from the past six homework, I get a better command of the low-level basic of the Computer Vision. I am looking forward to learning more new interesting knowledge of such field next quarter.

## III. Matlab Codes

```
function [found_rho, found_theta, accumulator] = ...
    hough(img_in, rho_step, theta_step, threshold)

    % Convert the input image into a grayscale, binary format.
    mat = rgb2gray(imread(img_in));

    % Get the edge points for the image using a Canny edge detector.
    edge_mat = edge(mat, 'canny');

    % Setup the matrices for the values of rho and theta.
    rho = 1:rho_step:sqrt((size(edge_mat, 1) ^ 2) + (size(edge_mat, 2) ^ 2));
    theta = 0:theta_step:(180 - theta_step);

    % Initialize the accumulator array.
    accumulator = zeros(length(rho), length(theta));

    % Find the coordinates of the edge points of the image.
    [x, y] = find(edge_mat);

    % Iterate over each edge point, converting Cartesian to polar coordinates.
    for i = 1:numel(x)
        for j = 1:length(theta)
            test_theta = (theta(j) * (pi / 180));
            test_rho = ((x(i) * cos(test_theta)) + (y(i) * sin(test_theta)));

            % Voting procedure for valid values of rho.
            if ((test_rho >= 1) && (test_rho <= rho(end)))
                test_vote = abs(test_rho - rho);
                min_vote = min(test_vote);
                rho_vote = find(test_vote == min_vote);
                for k = 1:length(rho_vote)
                    accumulator(rho_vote(k), j) = ...
                    (accumulator(rho_vote(k), j) + 1);
                end
            end
        end
    end

    % Find [rho, theta] values for peaks in the accumulator matrix.
    peaks = imregionalmax(accumulator);
    [test_rho, test_theta] = find(peaks);

    % Apply the line length threshold to the accumulator matrix.
    test_accumulator = (accumulator - threshold);

    % Setup matrices for final [rho, theta] values
    % (and get rid of that stupid preallocation warning).
    found_rho = zeros(size(test_rho));
    found_theta = zeros(size(test_theta));
    index = 1;

    % Find final [rho, theta] values in the thresholded accumulator matrix.
    for i = 1:numel(test_rho)
        if (test_accumulator(test_rho(i),test_theta(i)) >= 0)
            found_rho(index) = test_rho(i);
            found_theta(index) = test_theta(i);
            index = index + 1;
        end
    end

    % Remove zero values.
    found_rho = found_rho(any(found_rho,2),:);
    found_theta = found_theta(any(found_theta,2),:);
    % Plot the Hough peaks and intersection points using a heat map.
    subplot(1,2,1);
    imshow(imadjust(mat2gray(accumulator)), 'XData', found_theta, 'YData', ...
                    found_rho, 'InitialMagnification', 'fit');
    title('Hough transform of image');
    xlabel('\theta'), ylabel('\rho');
    axis on, axis normal;
    colormap(hot);

    % Plot the resulting lines from the Hough transform.
    subplot(1,2,2);
    imagesc(edge_mat);
    hold on;
    for i = 1:size(found_rho)
        plot_theta = theta(found_theta(i));
        plot_rho = rho(round(found_rho(i)));

        % Convert polar coordinates to linear equations (y = m * x + b).
        m = -(cosd(plot_theta) / sind(plot_theta));
        b = (plot_rho / sind(plot_theta));
        x = 1:size(edge_mat);

        % Plot the resulting equations.
        plot(((m * x) + b), x);
        hold on;
    end
    hold off;

end
```