Chapter 1

Introduction

The **Hough Transform** is an algorithm presented by Paul Hough in 1962 for the detection of features of a particular shape like lines or circles in digitalized images. In its classical form it was restricted to features that can be specified in a parametric form. However, a generalized version of the algorithm exists which can also be applied to features with no simple analytic form, but it is very complex in terms of computation time.
Nevertheless the Hough Transform has a broad application today, e.g. it is used in traffic observation systems, land monitoring and robotics. At my home university (Humboldt University Berlin) in Germany we use the Hough Transform algorithm in order to detect field lines in autonomous robot soccer (**RoboCup**). A correct detection is crucial for the robot as this information is used for self-localisation.
Although Hough Transform is a standard algorithm for line or circle detection it has weak points, especially its computational complexity. However a faster version called **Fast Hough Transform** has been developed.

1.1 Overview

In the next chapter the classic Hugh Transform is presented. Chapter 3 provides an analysis of that algorithm and some improvement considerations. After that some applications of the Hugh Transform, especially the use in RoboCup are shown in chapter 4. Finally chapter 5 contains the description and results of the provided matlab implementation.

Chapter 2

Classic Hugh Transform

The **classic Hugh Transform** is a standard algorithm for line and circle detection. It can be applied to many computer vision problems as most images 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**, unlike edge detectors.

2.1 Line Detection

The simplest case of Hough transform is the linear transform for detecting straight lines. In the image space, the straight line can be described as $\mathbf{y = mx + b}$ and can be graphically plotted for each pair of image points (x, y). In the Hough transform, a main idea is to consider the characteristics of the straight line not as image points x or y, but in terms of its parameters, here the slope parameter m and the intercept parameter b.
For example, Figure 2.1 shows 3 points on a 5x6 bitmap. Given this bitmap, we can define points A(1,2), B(2,4) and C(3,6). Point A can have a family of lines passing through it,

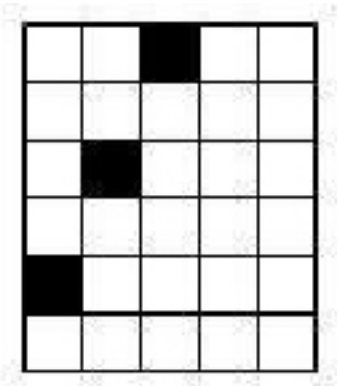

Figure 2.1: *Example of a line.*

and this can be expressed by the formula y = ax + b. In this formula, when we apply the values of A(1,2) to x and y, then a and b become parameters for the family of lines passing through A. Given this characterization of a line, we can then iterate through any number of lines that pass through a given point.
We incorporate the results in an **Accumulator Array**. At the beginning, the Accumulator Array is initialized to zero for all cells. After that, for every value of a with a corresponding value of b, the value of that cell is incremented by one.
If you draw all possible lines that pass through point A, some of these will also pass through points B and C. Similarly, some of the lines that pass through point B will also pass through points A and C. Cells are incremented in the accumulator array when lines pass through multiple points. At the end of the accumulation process, the cell that has the highest value represents the line that passes through the most number of points in the source image array.
However, this method has its drawbacks. If the line is horizontal, then a is 0, and if the line is vertical, then a is **infinite**. So, a more general representation of a line will be

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

2.2 Circle and Ellipse Detection

As stated before the Hugh transform can also be used for detection of circles and other parametrizable geometric figures. In theory any kind of curve can be detected if you can express it as a function of the form

$$f(a_1, a_2, \ldots, a_n, x, y) = 0.$$

For example a **circle** can be represented as

$$(x - a)^2 + (y - b)^2 - r^2 = 0$$

Then we have a n dimensional parameter space (three dimensional space for a circle).
This model has three parameters: two parameters for the centre of the circle and one parameter for the radius of the circle. For **ellipses** and other parametric objects the algorithm is quite similar, but the computation complexity (dimension of the Hough space) increases with the number of the variables.

2.3 Further Considerations

So how we obtain these points A, B, C used in the above example ?
This **candidate points** for a line must be computed with some other technique, for example with an edge detector. Applying the Hugh Transform to these points we obtain the line(s) which best fit the data. More precisely all candidate points **"vote"** for all possible lines passing through them, so since we hope that the real line produces many candidate points we can reconstruct it.

Chapter 3

Analysis

This chapter shows some advantages and disadvantages of the Hugh transform. Chapter 5 demonstrates these advantages and disadvantages on a real data set.

3.1 Advantages

One important difference between the Hugh Transform and other approaches is resistance of the former to noise in the image and its tolerance towards holes in the boundary line. Figures 3.1 and 3.2 compare the Hugh transform of a plain straight line with a dotted one. As can be seen there is almost no differences in the results. Chapter 5 illustrates these advantages on more examples.

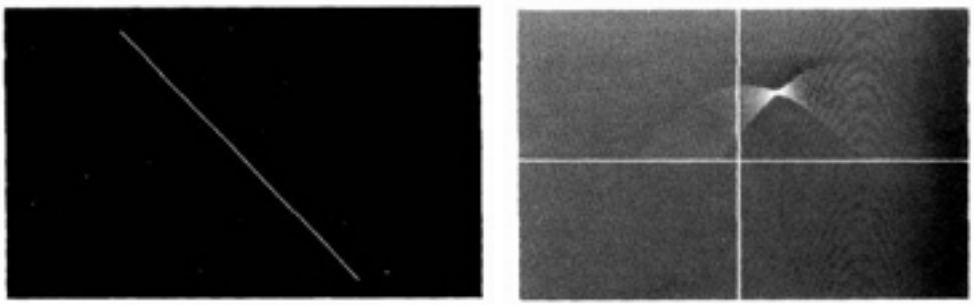

Figure 3.1: *Straight line and its Hough transform.*

3.2 Disadvantages

Since the Hugh Transform is a kind of Brute-Force method it is very complex in computation. It has two main drawbacks: large memory requirement and slowness. In order to find the plane parameters accurately, parameter space must be divided finely in all three directions, and an accumulator assigned to each block. Also it takes a long time to fill the accumulators when there are so many. The **Fast Hough Transform** gives considerable

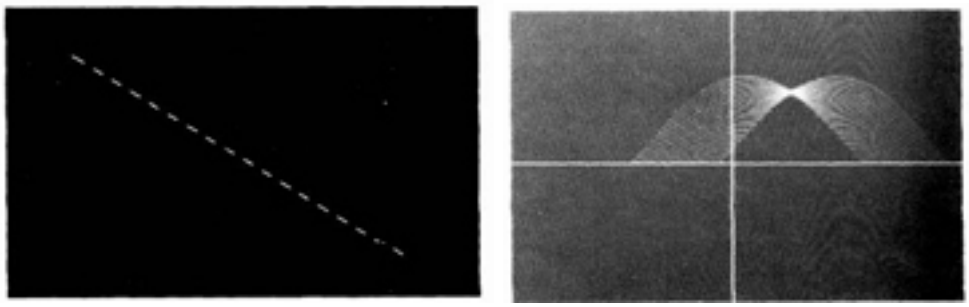

Figure 3.2: *Straight dashed line and its Hough transform.*

speed up and reduces memory requirement. Instead of dividing parameter space uniformly into blocks, the FHT homes in on the solution, ignoring areas in parameter space relatively devoid of votes. The relative speed advantage of the FHT increases for higher dimensional parameter spaces.
Another weakness of the Hugh Transform is that it often recognises many similar lines instead of the one correct one. The algorithm only returns a line without a starting and ending point. For that different postprocessing algorithms have to be used.

Chapter 4

Application

In the following I want to describe some application of the Hugh transform in **robot soccer**. I have worked in the AI lab of the Humboldt University zu Berlin for two years and was involved in the Aibo RoboCup project. Although the Hugh transform was not part of our world championship code mainly due to complexity reasons (we used a more simple heuristic approach) we did some experiments with it.
Since Hugh transform is a method of detecting lines it provides some useful information to the robot which can be used in **self-localization**. Knowing the field lines for example in the situation shown in figure 4.1 can improve the self-localization which was in our case a vision and odometry based stochastic monte-carlo process. Therefore each bit of information derived from the image is useful. Furthermore the knowledge about the position of lines can be helpful in many other situations e.g. for the goalie in order to decide in case of attack whether to go out of the goal and clear the situation or stay inside. Of special interest is the crossing point of two lines since the positions of it on the field are known by the robot and they are fixed. Another kind of information are the relationships between a line and other object. For example when the robot knows that the ball is outside the field it can behave differently e.g. prepare for the ball being put in by the referee. So we see that line detection can be quite important in such applications.

Figure 4.1: *The world from the robot's point of view*

Chapter 5

Results

5.1 Implementation

Before going into implementation details I would like to acknowledge the work of [HAA99] and [TAN06]. In my implementation some ideas are based on their work, especially the remapping mechanism from Hough to spacial space.
To perform the Hough Transform four steps are necessary. First of all the image has to be loaded, edges have to be computed and then the Hough algorithm is applied and finally the results are displayed e.g. detected lines in the image.

5.1.1 Loading Image

This step is straight forward. MATLAB provides methods for loading an image. Different kind of images can be loaded. Finally the image is converted into gray scale and is available for further use as `im`.

5.1.2 Edge Detection

Here I use the standard Canny edge detector provided by MATLAB. It must be noted that by varying the `threshold` and `sigma` we can influence the edge detection. This is important since a good edge detection is essential for the Hough Transform. The edges are automatically thinned by MATLAB, in my MATLAB version I cannot disable this. Nevertheless it is not a bad idea since it reduces the number of edge points and so the further computation complexity.

5.1.3 Hough Transform

I implemented three different types of Hough Transform. When choosing the detection type **Lines** we can detect straight lines in the image. Here the representation $\rho = x \cdot cos\theta + y \cdot sin\theta$ is used in order to avoid problems with vertical lines. The θ space is devided into `nTeta=200` values from 0 to $\pi - \pi/200$. The distance range is from 0 to `1.2*MaxEdgePoint`. These parameters are arbitrary and can be changed in the file `houghline.m`. By variation

of the parameter `threshold` we can influence the results since then only the entries in the accumulator are regarded which are greater then the threshold/10 times the maximum value. A relatively complex mechanism for locating and collecting the local maxima is used, thus we can detect multiple lines in a image.
The next type of detection is the **Circle + Radius**. Here the Hough circle detection algorithm is used, but the radius is fixed and can only be changed by the user. The mapping here is from x-y space to a-b space with $ax^2 + by^2 = r^2$. Here only one circle can be detected, the circle with the maximum votes. If different entries have the maximum vote then they are averaged.
Finally I implemented the detection type **Circle General**. Here the Hough circle detection algorithm is used, with a variable radius. The mapping here is from x-y space to a-b-r space with $ax^2 + by^2 = r^2$. Here only one circle can be detected, the circle with the maximum votes. If different entries have the maximum vote then they are averaged. That means that the best circle is computed with respect to a,b and r. The radius parameter is from 10 to 100 with steps of 4, this can be changed in the file `houghcircle.m`.

5.1.4 Object detection

Detection of objects from the Hough Transform is not an easy task. It can be compared to a clustering problem since we have the Hough space and have to detect regions of high votes in there. BUT we do not know how many clusters (objects) we have in the image and how big the clusters are. In other words we have to decide whether two point with high votes in the Hough space belong to the same line or to two different lines.
I tried to solve that problem for lines by using a threshold and applying some morphological operator. For the circles I only determined the maximum value in the accumulator and averaged all entries having this maximum value. However we could also used some kind of threshold here to determine the local maxima and so determine more then one circle.

5.2 Results

5.2.1 Single Line

First of all I will apply the Hough Transform to images containing one single lines. I will demonstrate that the line detection is very reliable in this case even with dotted lines or noise.
When applying the algorithm to the images `line.jpg, lineDotted.jpg` and `lineDottedStrong.jpg` we can see no difference in the detection, in all cases the correct line is detected. This is one main advantage of the Hough Transform which distinguishes it from other line detection mechanisms. It is **resistant to gaps in lines**.
In the other case when regarding lines with additional false points or an overall noise we can see another advantage of the Hough Transform, its **resistance to noise**. However in the case of noise and especially in images like `lineStep4Noise20.jpg, lineStep4Noise30.jpg` or `lineNoise2.jpg` it is quite difficult to detect only this one line. Of course is the Hough Transform **highly dependend on the candidate points** it gets, so changes in the edge

detection like increasing the threshold value can produce better results here. The Hough Algorithm does not need any knowledge about the image though it produces quite good results and is reliable to noise and gaps in the lines.

5.2.2 Multiple Line

Objects like in the image `image.bmp` can be detected quite well with this approach. The detection of lines in the image `lines.jpg` can be seen in figure 5.1. It works quite well for a threshold value of 7. Other images like `map.jpg` are harder to process since either the

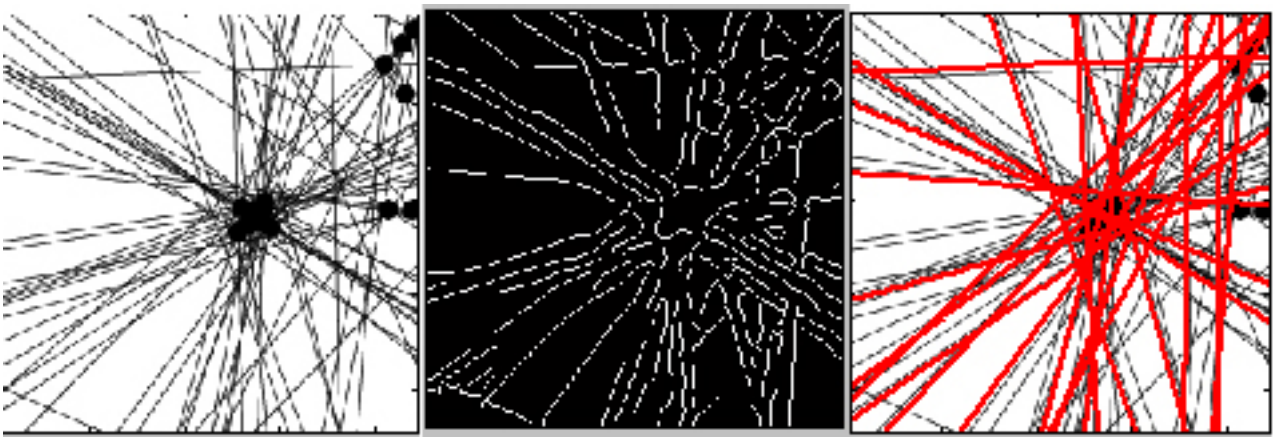

Figure 5.1: *Detection of lines in* `lines.jpg`.

number of edge points is quite huge resulting in a computation intensive Hough Transform or the edge points are rare and plotted all over the image which is due to **thinning** in the standard edge detecting function of MATLAB. Unfortunatelly I cannot deactivate this function in my MATLAB version.
The main problem in line detection here is not to perform the Hough Transform. It is far more difficult to determine **how many lines** are in the image and whether two points in the Hough space belong to the same line or not. This requires a detection of local maxima in the Hough space and **clustering of points** in order to avoid detecting the same line more than one time. This is not an easy task. Furthermore it is not clear from the Hough output where the line starts and where it ends, so a **post processing** is needed for real object detection.
At last we can see in figure 5.2 that the Hough Line Detection Algorithm can work in RoboCup.

5.2.3 Circles

The circle detection algorithm can be studied on the image `euro.jpg` quite well. When running the algorithm with a different radius value we can see that different coins are being detected. When running the general circle detection we detect the 50 Euro Cent coin and the Hough space is like in figure 5.3. This is the Hough Space for the detected

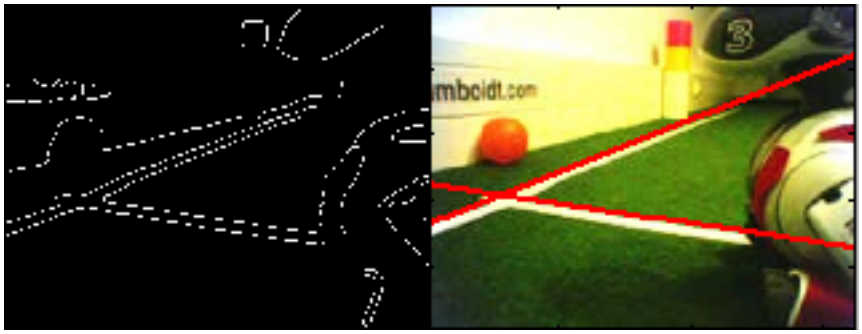

Figure 5.2: *Detection of lines in* `robot.jpg`.

radius, in reality the Hough space is three dimensional. The image `moon.jpg` was quite

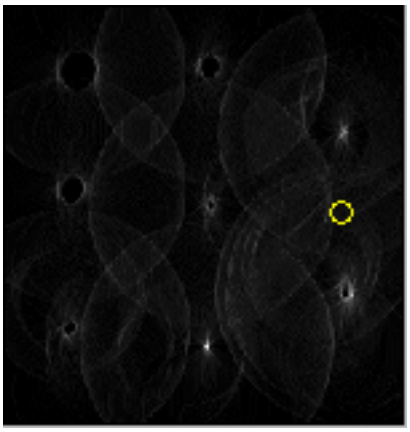

Figure 5.3: *Hough Transform for circles in* `euro.jpg`.

hard to detect at first but when performing the edge detection with a bigger sigma value e.g. 6 then we obtain a nice circle around the moon, which can be detected very easily with the Hough algorithm. So we see that the **edge detection is quite important** for the performance of the Hough Transform, on the one hand it influences the computational complexity since more edge points means more computation but on the other hand it must contain the important edges to allow detection.
So we see that although the Hough Transform is a standard algorithm there are **many other important issues** like candidate points computation, finding local maxima or post processing when detecting objects.