Chapter 10: Image Segmentation

Table of Contents

[Detection of Discontinuities 2](#_Toc138876667)

[Point Detection 2](#_Toc138876668)

[Line Detection 3](#_Toc138876669)

[Edge Detection 3](#_Toc138876670)

[Noisy Edges 5](#_Toc138876671)

[Diagonal Edges 6](#_Toc138876672)

[Smoothing Filters 7](#_Toc138876673)

[Hough Transformation 8](#_Toc138876674)

[-Plane 9](#_Toc138876675)

[Edge Linking and Boundary Detection 10](#_Toc138876676)

[Local Processing 10](#_Toc138876677)

[Global Processing via Hough Transformation 11](#_Toc138876678)

[Generalized Hough Transformation 11](#_Toc138876679)

[Global Processing via Graph Theoretic Techniques 13](#_Toc138876680)

**Segmentation** is the process of dividing an image into its constituent regions or objects. The process should stop when the objects of interest have been isolated. Segmentation algorithms are generally based on one of two properties:

* **Discontinuity** – The image is partitioned when there is an abrupt change in intensity, such as at edges.
* **Similarity** – The image is partitioned into sections that are similar according to a set of predefined criteria.

## Detection of Discontinuities

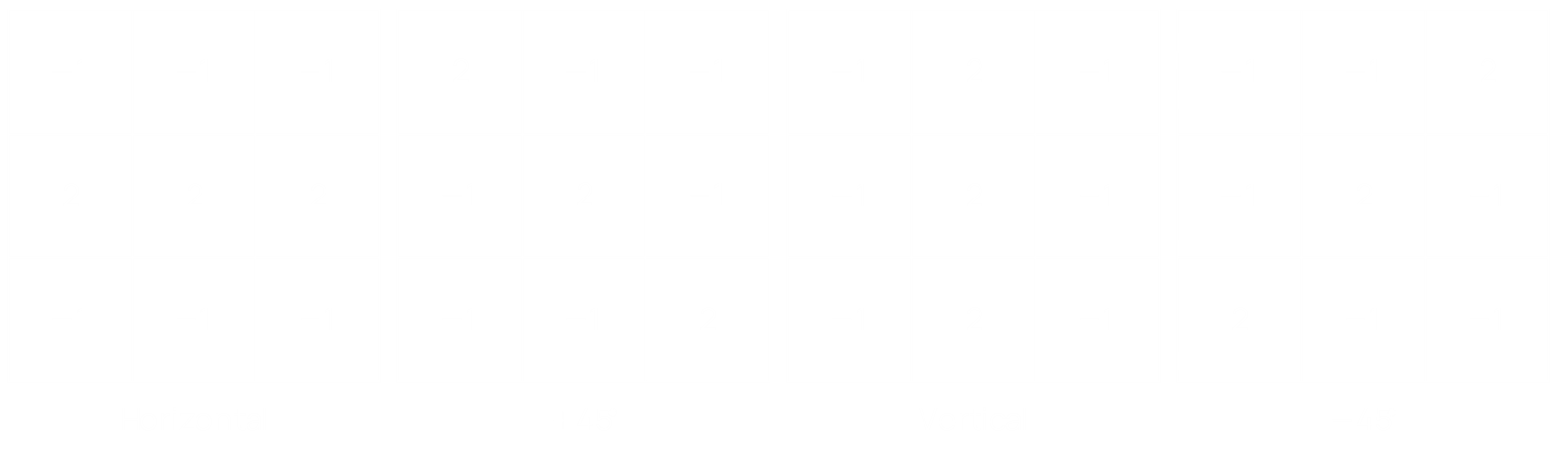
There are three types of discontinuities in gray-scale images, points, lines and edges. The most common way of detecting such discontinuities is to use a **mask**.

### Point Detection

When applying a mask on top of an image, a **point** is said to be detected if, at the center of the mask, , where is the response from the application of the mask and is the non-negative threshold. It is possible that will get multiple points from a single mask depending on the threshold. One possible mask that can be used in this scenario is a **Laplacian mask**.

### Line Detection

Several different masks can be used for line detection depending on the direction in which the line is.



To detect all the lines in a specific direction, we run the mask over the image and threshold the response.

### Edge Detection

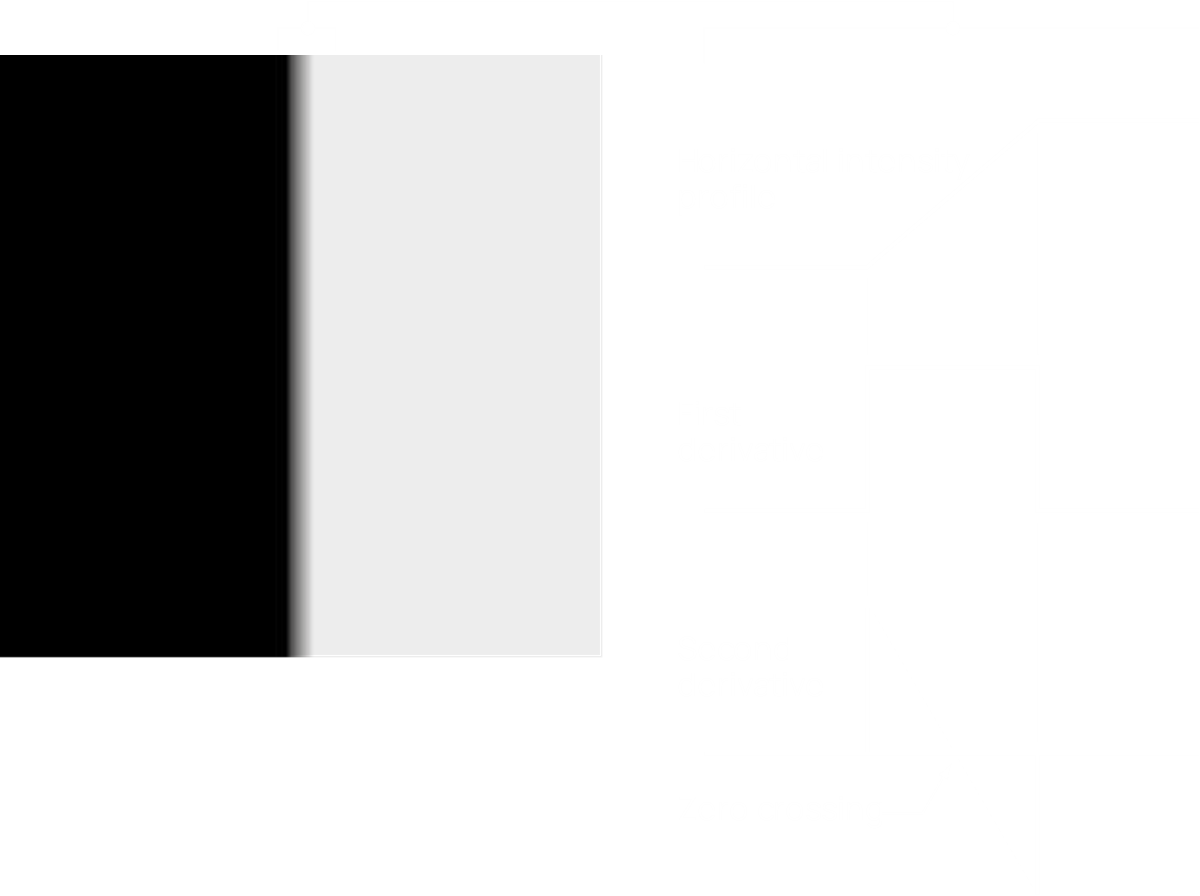
We have previously seen that edges can be detected using **gradient masks** and **Laplacian masks**. There are actually several types of edges which makes this process a little more complicated.

An **ideal edge** is one which has a single sharp cutoff. This is opposed to a **ramp edge** which has a more gradual change.



Generally, we will never have an ideal edge. Even if an edge seems to be ideal, if we zoom in far enough, we will see a ramp.

We have previously seen that the ramp edge causes two responses when using a Laplacian mask.



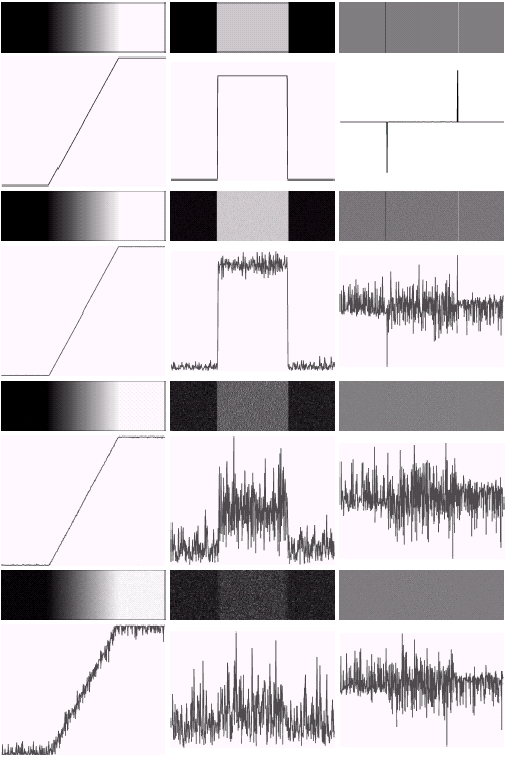
The response from the Laplacian mask will not be so clean generally. Instead, we will have a curve.



For such a curve, to get the exact point of the response, we draw a line from the top of the highest point to the bottom of the lowest point. The point at which this line crosses the horizontal axis is considered to be the point at which we have an edge. This is called the **Zero Crossing Property**.

### Noisy Edges

The amount of noise we have in our image has a huge effect on how well we will be able to detect edges. The image below shows the effect of adding increasing amounts of Gaussian noise on the results of the first and second order derivatives.



Notice that the second order derivative is **more noise sensitive** than the first order one. This is due to the fact that the original image is being differentiated twice to obtain this line. Suppose we have a point with a 10% chance of being corrupted by noise. Thus, has a 20% chance and has a 40% chance.

### Diagonal Edges

The images below show the response of **diagonal edge detection**.



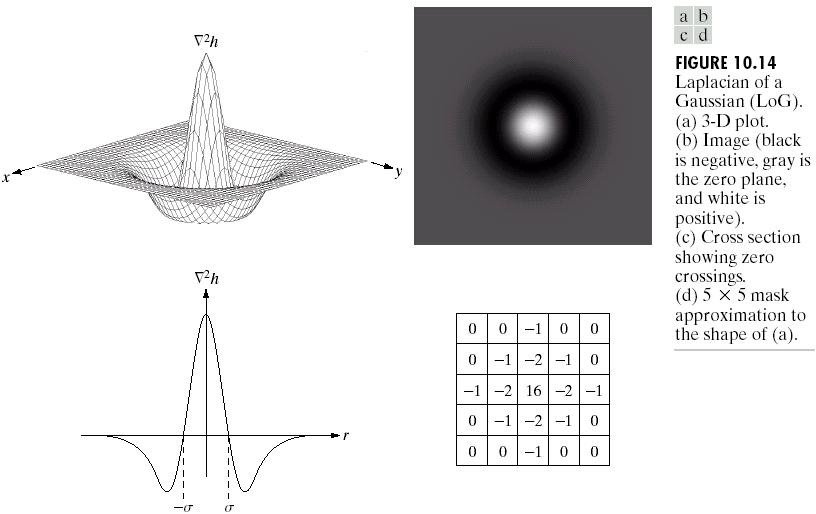
Notice that when we are detecting edges at , we have no responses in the opposite edge, i.e. , and vice versa. However, in both cases, we do have a response in the horizontal and vertical directions. These responses are less strong, but they do exist. This happens because diagonal edges have components in the vertical and horizontal directions, which cause these response. Similarly, for horizontal and vertical edges, we will in fact have some response in the diagonal directions.

### Smoothing Filters

In the image shown above, notice that we have some response coming from the walls of the building as well. This occurs because there are some small variations in the intensity values there as well. The region is not perfectly smooth. To get rid of these, we can use a **smoothing filter**. We are essentially removing noise from the results of the gradient mask.

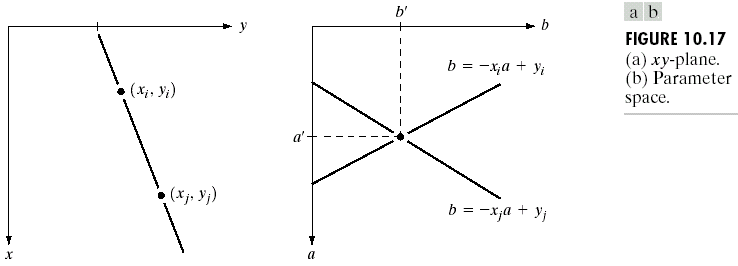
The same applies for Laplacian masks. To apply smoothing, we can use **Gaussian smoothing**. The Laplacian mask and the Gaussian smoothing mask are both linear operations, so they can be combined.

This combination results in a mask with a very distinctive shape, which is why it is also called the **Mexian hat**.



## Hough Transformation

**Hough Transformation** is a very old and very simple method of detecting **lines** in images. A line in the image will follow the equation . If we pick some point on the line, , there are an infinite number of possible values of and that will create lines that contain that point. Since it is not possible to keep track of all of these points, we can instead move to the **parameter space**.



In the parameter space, each line represents the possible values of and which will result in a line in the plane that crosses our chosen point.

We can repeat this process multiple times for different points in the original line in the image space. This will result in multiple lines in the parameter space. These lines will intersect, perhaps in multiple places due to noise. This means we need a mechanism to decide which of those intersections is the correct one.

To do this, we can increment the value of each point in the parameter space by 1 when a line is drawn over those points. Thus, the intersection points will have higher values. If there are more than intersections at a particular point, we choose the values of and at that point as the final result.

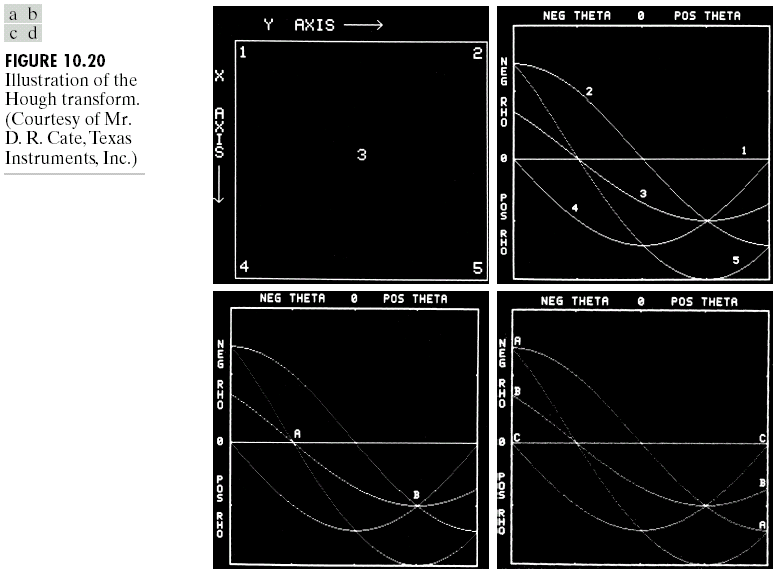
Since we are counting the number of intersections at different points, the parameter space is also called the **Accumulator space**.

The neat part about Hough Transformation is that it will work even if there are obstructions in the path of the line in the original image. It will even work for other shapes, such as circles. We just need to modify the equation. For example, if we use the equation , we will have a 3D parameter space in which we will have intersecting spheres. We can use the same process outlined above to find the correct intersection point.

### -Plane

The problem with the parameter space as described above is that each line we draw on it is **infinite**. To avoid this issue, we can instead use the equation to represent lines in the image space. The parameter space in this case is **finite**.

The lines in the parameter space in this case will not be straight lines, but rather curves. Regardless, the process of finding the correct parameter values remains the same.



## Edge Linking and Boundary Detection

The edge detection algorithms we have seen can be followed by **linking algorithms** which exists to link edges together. There are three approaches to doing this, local processing, global processing via Hough Transformation and global processing via graph-theoretic techniques.

### Local Processing

In **Local Processing**, we analyze the behaviour of pixels in a small neighbourhood around each detected edge. We will be linking edges together if they satisfy two criteria:

1. The response from some pixel is similar to the response from the chosen pixel , i.e., , where is a tunable threshold.
2. The direction of the gradient vector is similar to that of the chosen pixel, i.e., , where is a tunable threshold.

The issue with this process is that it does not work for **broken edges**.

### Global Processing via Hough Transformation

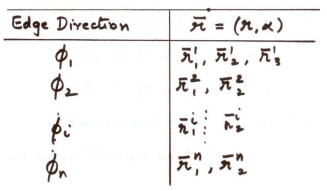
We have already seen the process of Hough Transform. Similar to how there was some degree of error in local processing for edge linking, there is also some error here. This error appears as a **non-overlapping lines** in the accumulator space. Essentially, two points on the same edge might result in lines that do not overlap at exactly the same point, but are still close enough that they are basically the same line. We can fix this issue by applying **smoothing** to the image beforehand, which will make the lines overlap.

### Generalized Hough Transformation

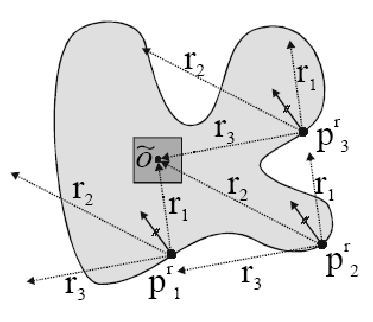
We have previously discussed that we can theoretically detect any shape using Hough Transformation. However, the shape we want to detect must still follow an equation, so that we can plot something in the accumulator space. For complicated shapes, this becomes practically infeasible. What we need is a **Generalized Hough Transformation**.



Suppose we have a shape such as the one shown above, and we take a **reference point** somewhere inside the shape, denoted by . Next, we sample several points on the boundary of the shape. From each point, we calculate two values. The first value is , the angle of the perpendicular at that point with the -axis. The second value is , the distance between the point and the reference point. We can store these values in a table. If we use the values as the keys of the table, we can have several values of for the same value of .



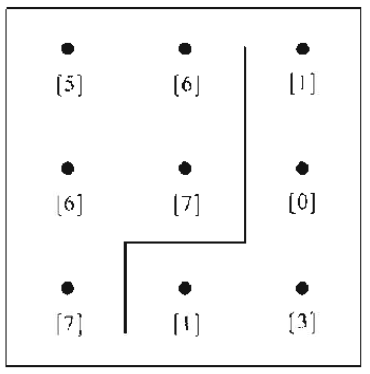
When trying to detect the shape, each point will give us a specific value. For this value, we will take all the possible values from our table and mark the positions we end up in with a vote. If we do this for large number of points, given that the shape is similar, we will end up with a large vote at a specific point. This is how we will know that it is the same shape.



The above image for example, is a different shape. As a result, we end up with a low vote, with the highest value being .

The drawback of this process is that the size of the shape must also match the original shape. The same shape in a different size will not work.

### Global Processing via Graph Theoretic Techniques



Suppose we have a setup such as the one shown above, where each dot represents a pixel and there is an edge that goes between pixels. We need to detect this edge.

Let be the maximum intensity in the entire image and and be the intensities of two pixels. The cost between the two pixels is given by

A **large difference** between the intensity values of two pixels indicates that there is an edge between them. In such a case, the resulting **cost** will be **small**.

Depending on the direction in which we are calculating, we will end up with different cost values. This can be used to create a **graph**.



The edge is the path with the lowest cost.