Chapter 12
Image Processing

The distance sensor on your self-driving car detects an object 100 m in front of your car. Are you following the car in front of you at a safe distance or has a pedestrian jumped into the road? The robotics algorithms presented so far have been based upon the measurement of physical properties like distance, angles and reflectance. More complex tasks require that a robot obtain detailed information on its surroundings, especially when the robot is intended to function autonomously in an unfamiliar environment.

For us, the obvious way to make sense of our environment is to use vision. We take vision for granted and don't realize how complex our visual system—our eyes and brain—really is. In fact, about 30% of the brain is used for vision. We can instantly distinguish between a moving car and a pedestrian crossing the road and react quickly.

For almost two hundred years it has been possible to automatically record images using a camera, but the interpretation of images remained a task for humans. With the advent of computers, it became possible to automatically process and interpret images. Digital images are familiar: weather maps from satellites, medical images (X-rays, CT and MRI scans, ultrasound images), and the photos we take with our smartphones. The field of *digital image processing* is one of the most intensely studied fields of computer science and engineering, but image processing systems have not yet reached the capability of the human visual system.

In this chapter, we present a taste of algorithms for digital image processing and describe how they are used in robotics systems. Sections 12.1 and 12.2 provide an overview of imaging systems and digital image processing. Sections 12.3–12.6 describe algorithms for image processing: enhancement by digital filters and histogram manipulation, segmentation (edge detection), and feature recognition (detection of corners and blobs, identification of multiple features).

For reasons of cost and computing power, few educational robots use cameras, so to study image processing algorithms you can implement the algorithms on a personal computer using images captured with a digital camera. Nevertheless, we propose some activities that demonstrate image processing algorithms on an educational

© The Author(s) 2018

M. Ben-Ari and F. Mondada, *Elements of Robotics*,
https://doi.org/10.1007/978-3-319-62533-1_12

robot. The robot moves over a one-dimensional image and samples are read by a ground sensor. This results in a one-dimensional array of pixels that can be processed using simplified versions of the algorithms that we present.

12.1 Obtaining Images

In this section we give an overview of design considerations for imaging systems.

Optics

The optical system of a camera consists of a lens that focuses light on a sensor. The wider the lens, the more light that can be collected, which is important for systems that need to work in dark environments. The longer the focal length (which is related to the distance between the lens and the sensor), the greater the magnification. That is why professional photographers carry heavy cameras with long lenses. Manufacturers of smartphones are faced with a dilemma: we want our phones to be thin and elegant, but that limits the focal length of the camera. For most robotics applications, magnification is not worth the size and weight required to achieve a long focal length.

Resolution

Once upon a time, images were captured on film by a chemical reaction caused by light hitting a sheet of plastic covered with an emulsion of tiny silver particles. In principle, each particle could react independently so the resolution was extremely high. In digital images, light is captured by semiconductor devices such as *charge-coupled devices (CCD)*. A digital camera contains a chip with a fixed number of elements in a rectangular array. Each element measures the light intensity independently and these measurements are called *pixels*. The more pixels captured by a chip of a given area, the higher the resolution. Currently, even inexpensive cameras in smartphones can capture millions of pixels in a single image.

The problem with high resolution images is the large amount of memory needed to store them. Consider a high-resolution computer screen with 1920×1080 pixels and assume that each pixel uses 8 bits to store intensity in the range 0–255. A single image requires about 2 megabytes (MB) of memory. An embedded computer could analyze a single such image, but a mobile robot may need to store several images per second.

Even more important than the amount of memory required is the computing power required to analyze the images. Image processing algorithms require the computer to perform a computation on each individual pixel. This is not a problem for an astronomer analyzing images sent to earth from a space telescope, but it is a problem for a self-driving car which needs to make decisions in a fraction of a second.

Color

Our visual system has the capability of distinguishing a range of wavelengths called *visible light*. We discern different wavelengths as different colors. Light of longer wavelengths is called *red*, while light of shorter wavelengths is called *violet*. The human eye can distinguish millions of different colors although we name only a few: red, orange, yellow, green, cyan, blue, violet, etc. Color is one of the primary tools that we use to identify objects.

Sensors are able to measure light of wavelengths outside the range we call visual light: *infrared* light of longer wavelengths and *ultraviolet* light of shorter wavelengths. Infrared images are important in robotics because hot objects such as people and cars can be detected as bright infrared light.

The problem with color is that it triples the requirements for storing and processing images. All colors can be formed by taking varying amounts of the three *primary colors*: red, green and blue (RGB). Therefore, a color image requires three bytes for each pixel. A single color image of resolution 1920×1080 requires over 6 MB of memory to store and the image processing takes at least three times as long.

12.2 An Overview of Digital Image Processing

The optical system of a robot captures images as rectangular arrays of pixels, but the tasks of a robot are expressed in terms of objects of the environment: enter a room through a door, pick up an item off a shelf, stop if a pedestrian walks in front of the car. How can we go from pixels to objects?

The first stage is *image enhancement*. Images contain noise that results from the optics and electronics. Furthermore, the lighting in the environment can cause an image to be too dark or washed out; the image may be accidentally rotated; the image may be out of focus. All these problems are independent of the content. It doesn't matter if an image that is out of focus shows a cat or a child. Image enhancement algorithms typically work by modifying the values assigned to individual pixels without regard to their meaning.[1]

Image enhancement is difficult because there is no formal definition of what it means to enhance an image. A blurred blob might be dirt on a camera's lens or an unknown galaxy. Section 12.3 presents two approaches to image enhancement: filtering removes noise by replacing a pixel with an average of its neighboring pixels and histogram manipulation modifies the brightness and contrast of an image.

Objects are distinguished by lines, curves and areas. A door consists of three straight edges of a rectangle with one short side missing. A traffic light consists of three bright disks one above another. Before a door or traffic light can be identified, image processing algorithms must determine which pixels represent lines, edges, etc. This process is called *segmentation* or *feature extraction* because the algorithms have to determine which pixels are part of a segment of an image.

[1]We limit ourselves to *spatial* processing algorithms that work on the pixels themselves. There is another approach called *frequency* processing algorithms, but that requires mathematical techniques beyond the scope of this book.

Segmentation would be easy if edges, lines and curves were uniform, but this is not what occurs in real images. An edge may be slanted at an arbitrary angle and some of its pixels may obscured by shadows or even missing. We are familiar with *captchas* where letters are intentionally distorted to make automatic recognition very difficult whereas humans can easily identify distorted letters. Enhancement algorithms can make segmentation easier, for example, by filling in missing pixels, but they may also introduce artificial segments. Section 12.4 demonstrates one segmentation technique: a filter that detects edges in an image.

The final phase of image processing is to recognize objects. In Sect. 12.5, we present two algorithms for detecting corners: by locating the intersection of two edges and by counting neighbors with similar intensities. Section 12.6 describes how to recognize *blobs*, which are areas whose pixels have similar intensities but which are not bounded by regular features such as lines and curves. Finally, Activity 12.6 demonstrates the recognition of an object that is defined by more than one feature, such as a door defined by two edges that are at an arbitrary distance from each other.

12.3 Image Enhancement

Figure 12.1a shows an image of a rectangle whose intensity is uniform horizontally and shaded dark to light from top to bottom. The representation of the image as a 6×10 rectangular array of pixels is shown in Fig. 12.2a, where each pixel is represented by a light intensity level in the range 0–100. Now look at Fig. 12.1b: the image is no longer *smooth* in the sense that there are three points whose intensity is not similar to the intensities of its neighbors. Compare the pixel array in Fig. 12.2b with the one in Fig. 12.2a: the intensities of the pixels at locations $(2, 3)$, $(3, 6)$, $(4, 4)$ are different. This is probably the result of noise and not an actual feature of the object being photographed.

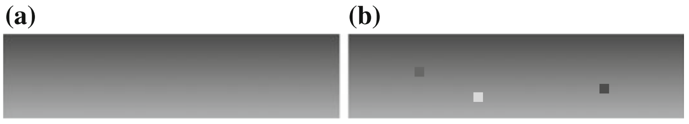


Fig. 12.1 **a** Image without noise. **b** Image with noise

(a)

	0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10	10
1	20	20	20	20	20	20	20	20	20	20
2	30	30	30	30	30	30	30	30	30	30
3	40	40	40	40	40	40	40	40	40	40
4	50	50	50	50	50	50	50	50	50	50
5	60	60	60	60	60	60	60	60	60	60

(b)

	0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10	10
1	20	20	20	20	20	20	20	20	20	20
2	30	30	30	**20**	30	30	30	30	30	30
3	40	40	40	40	40	40	**10**	40	40	40
4	50	50	50	50	**90**	50	50	50	50	50
5	60	60	60	60	60	60	60	60	60	60

Fig. 12.2 **a** Pixel array without noise. **b** Pixel array with noise

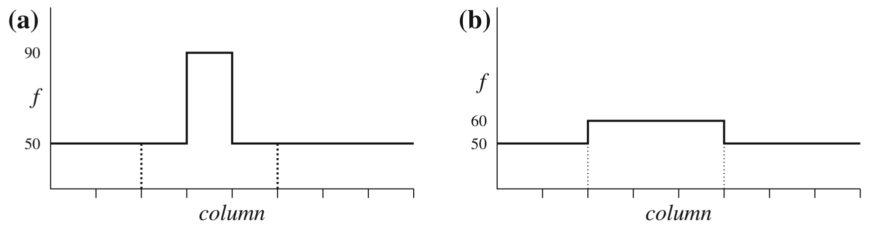


Fig. 12.3 **a** Intensity plot before averaging. **b** Intensity plot after averaging

It doesn't really matter where the noise comes from: from the object itself, dust on the camera lens, non-uniformity in the sensor or noise in the electronics. It is impossible to get rid of the noise entirely, because we can never be sure whether a pixel is noise or an actual feature of the object, but we do want to enhance the image so that the noise is no longer noticeable.

12.3.1 Spatial Filters

Consider row 4 in the pixel array in Fig. 12.2b:

$$50,\ 50,\ 50,\ 50,\ 90,\ 50,\ 50,\ 50,\ 50,\ 50\,.$$

Figure 12.3a is a plot of the light intensity f for the pixels in that row. It is clear that one of the pixels has an unlikely value because its value is so different from its neighbors. A program can make each pixel more like its neighbors by replacing the intensity of the pixel with the average of its intensity and the intensities of its neighbors. For most of the pixels in the row, this doesn't change their values: $(50 + 50 + 50)/3 = 50$, but the noise pixel and its two neighbors receive new values: $(50+90+50)/3 \approx 60$ (Fig. 12.3b). Averaging has caused two pixels to receive "wrong" values, but overall the image will be visually enhanced because the intensity of the noise will be reduced.

Taking the average of a sequence of pixels is the discrete version of integrating a continuous intensity function. Integration smooths out local variation of the function. The dotted lines in Fig. 12.3a, b indicate a three-pixel sequence and it can be seen that the areas they bound are about the same.

The averaging operation is performed by applying a *spatial filter* at each pixel of the image.[2] For the two-dimensional array of pixels, the filter is represented by a 3×3 array, where each element of the array specifies the factor by which the pixel and its neighbors are multiplied. Each pixel has four or eight neighbors, depending on whether we include the diagonal neighbors. Here, we include the diagonal pixels in the filters.

[2]The mathematical term for applying a function g at every point of a function f is called (discrete) *convolution*. For continuous functions integration is used in place of the addition of averaging.

(a)	0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10	10
1	20	20	18	18	18	20	20	20	20	20
2	30	30	28	28	28	26	26	26	30	30
3	40	40	38	43	43	41	36	36	40	40
4	50	50	50	54	54	51	46	46	50	50
5	60	60	60	60	60	60	60	60	60	60

(b)	0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10	10
1	20	20	19	19	19	20	20	20	20	20
2	30	30	29	25	29	28	28	28	30	30
3	40	40	39	41	41	40	25	38	40	40
4	50	50	50	52	70	50	48	48	50	50
5	60	60	60	60	60	60	60	60	60	60

Fig. 12.4 **a** Smoothing with the box filter. **b** Smoothing with a weighted filter

The *box filter* is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

The results of the multiplications are added and the sum is divided by 9 to scale the result back to an intensity value.

The application of the filter to each pixel (r, c) can be written explicitly as:

$$\begin{array}{llll} g(r,c) = (& & & \\ & f(r-1,c-1) & + f(r-1,c) & + f(r-1,c+1) + \\ & f(r,c-1) & + f(r,c) & + f(r,c+1) \quad + \\ & f(r+1,c-1) & + f(r+1,c) & + f(r+1,c+1) \\ &) \,/\, 9\,. & & \end{array}$$

The result of applying the box filter to the noisy image in Fig. 12.2b is shown in Fig. 12.4a.[3] The intensity values are no longer uniform but they are quite close to the original values, except where a noise pixels existed. The second row from the bottom shows that the noise value of 90 no longer appears; instead, all the values in the row are close together in the range 46–54.

The box filter gives equal importance to the pixel and all its neighbors, but a *weighted filter* uses different factors for different pixels. The following filter gives much more weight to the pixel itself than to its neighbors:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

It would be appropriate to use this filter if we think that a pixel almost certainly has its correct value, but we still want its neighbors to influence its value. After applying this filter, the result must be divided by 16 to scale the sum to an intensity value. Figure 12.4b shows the result of using the weighted filter. Looking again at the second

[3]The filter is not applied to the pixels in the boundary of the image to avoid exceeding the bounds of the array. Alternatively, the image can be padded with extra rows and columns.

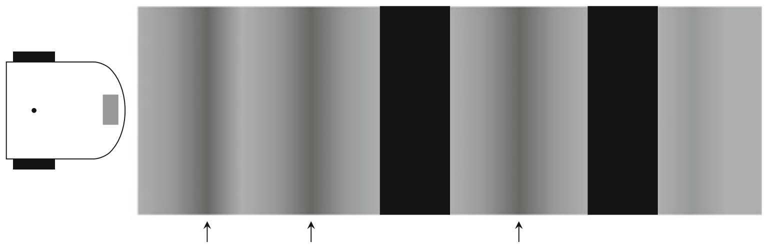


Fig. 12.5 One-dimensional image enhancement

row from the bottom, the value of 90 has only been reduced to 70 because greater weight is given to the pixel relative to its neighbors.

Activity 12.1: Image enhancement: smoothing

- Print a sheet of paper with a gray-level pattern like the one shown in Fig. 12.5. The pattern has two black lines that we wish to detect but also three dark-gray areas (indicated by the arrows) that are likely to be incorrectly detected as lines.
- Program the robot so that it moves from left to right over the pattern, sampling the output of the ground sensor. Examine the output and set a threshold so that the robot detects both the black lines and the dark-gray areas. Modify the program so that in its second pass, it indicates (by light or sound) when it has detected a black line and a dark area.
- Modify the program so that it replaces every sample by the average of the intensity of the sample and its two neighbors. The robot should now detect the two black lines but not the gray areas.
- Experiment with different weights for the average.
- Experiment with different sampling rates. What happens if you sample the ground sensor at very short intervals?

12.3.2 Histogram Manipulation

Figure 12.6a shows the pixels of a binary image: an image where each pixel is either black or white.[4] The image shows a 3×5 white rectangle on the black background. Figure 12.6b shows the same image with a lot of random noise added. By looking

[4]The values 10 for black and 90 for white have been used instead of the more usual 0 and 100 for clarity in printing the array.

(a)	0	1	2	3	4	5	6	7	8	9
0	10	10	10	10	10	10	10	10	10	10
1	10	10	10	10	10	10	10	10	10	10
2	10	10	10	**90**	**90**	**90**	**90**	**90**	10	10
3	10	10	10	**90**	**90**	**90**	**90**	**90**	10	10
4	10	10	10	**90**	**90**	**90**	**90**	**90**	10	10
5	10	10	10	10	10	10	10	10	10	10

(b)	0	1	2	3	4	5	6	7	8	9
0	19	17	37	19	26	11	46	27	37	10
1	11	24	17	30	14	43	29	22	34	46
2	31	37	38	63	72	86	65	64	27	47
3	33	38	49	73	63	66	59	76	40	10
4	47	13	44	90	86	56	63	65	18	44
5	10	34	29	14	35	31	26	42	15	25

Fig. 12.6 **a** Binary image without noise. **b** Binary image with noise

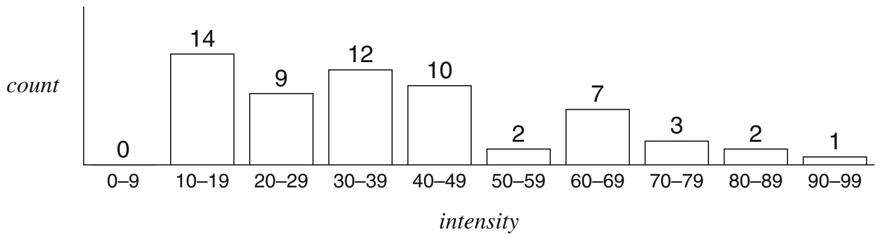


Fig. 12.7 Histogram of the noisy image

at the image it is possible to identify the rectangle, but it is very difficult to do and smoothing the image won't help.

Let us now construct a *histogram* of the intensities (Fig. 12.7). A histogram is constructed of *bins*, where each bin stores a count of the pixels having a range of intensities. The histogram in the figure contains ten bins for intensities in the ranges 0–9, 10–19, ..., 91–99. If we assume that the white rectangle is small relative to the background, it is easy to see from the histogram that there are two groups of pixels, those that are relatively dark and those that are relatively bright. A threshold of 50 or 60 should be able to distinguish between the rectangle and the background even in the presence of noise. In fact, a threshold of 50 restores the original image, while a threshold of 60 correctly restores 13 of the 15 pixels of the rectangle.

The advantage of histogram manipulation is that it is very efficient to compute even on large images. For each pixel, divide the intensity by the number of bins and increment the bin number:

```
for each pixel p
    bin_number ← intensity(p) / number_of_bins
    bins[bin_number] ← bins[bin_number] + 1
```

Compare this operation with the application of a 3×3 spatial filter which requires 9 multiplications, 8 additions and a division at each pixel. Furthermore, little memory is needed. We chose 10 bins so that Fig. 12.7 could display the entire histogram, but a full 8-bit grayscale histogram requires only 256 bins.

Choosing a threshold by examining a plot of the histogram is easy, and if you know roughly the fraction of the background covered by objects, the selection of the threshold can be done automatically.

Algorithms for histogram manipulation can perform more complex enhancement than the simple binary threshold we described here. In particular, there are algorithms for enhancing images by modifying the brightness and contrast of an image.

Activity 12.2: Image enhancement: histogram manipulation

- Modify the program in Activity 12.1 so that it computes the histogram of the samples.
- How does the histogram change if the number of samples is increased?
- Examine the histogram to determine a threshold that will be used to distinguish between the black lines and the background.
- Compute the sum of the contents of the bins until the sum is greater than a fraction (perhaps one-third) of the samples. Use the index of the last bin to set the threshold.

12.4 Edge Detection

Medical image processing systems need sophisticated image enhancement algorithms for modifying brightness and contrast, removing noise, etc. However, once the image is enhanced, interpretation of the image is performed by specialists who know which lines and shadows correspond with which organs in the body, and if the organs are normal or not. An autonomous robot does not have a human to perform interpretation: it needs to identify objects such as doors in a building, boxes in a warehouse and cars on the road. The first step is to extract features or segments such as lines, edges and areas.

Consider the 6 × 6 array of pixels in Fig. 12.8a. The intensity level of each row is uniform but there is a sharp discontinuity between the intensity level of rows 2 and 3. Clearly, this represents an edge between the dark area at the top and the light area at the bottom. Averaging will make the intensity change smoother and we lose the sharp change at the edge.

(a)

	0	1	2	3	4	5
0	30	30	30	30	30	30
1	30	30	30	30	30	30
2	30	30	30	30	30	30
3	**50**	**50**	**50**	**50**	**50**	**50**
4	**50**	**50**	**50**	**50**	**50**	**50**
5	**50**	**50**	**50**	**50**	**50**	**50**

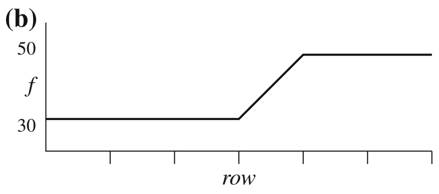


Fig. 12.8 **a** Image with an edge. **b** Intensity of edge

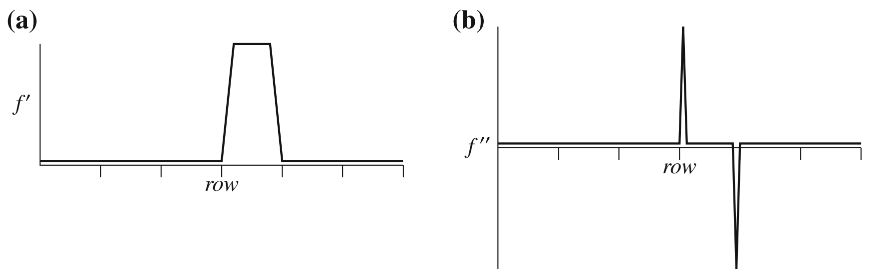


Fig. 12.9 **a** First derivative of edge intensity. **b** Second derivative of edge intensity

Since averaging is an integrating operator that *removes* abrupt changes in intensities, it is not surprising that the differential operator can be used to *detect* abrupt changes that represent edges. Figure 12.8b plots the intensity against the row number along a single column of Fig. 12.8a, although the intensities are shown as lines instead of as discrete points. The intensity doesn't change for the first three pixels, then it rapidly increases and continues at the higher level. The first derivative f' of a function f is zero when f is constant, positive when f increases and negative when f decreases. This is shown in Fig. 12.9a. An edge can be detected by searching for a rapid increase or decrease of the first derivative of the image intensity.

In practice, it is better to use the second derivative. Figure 12.9b shows a plot of f'', the derivative of f' in Fig. 12.9a. The positive spike followed by the negative spike indicates a transition from dark to light; if the transition were from light to dark, the negative spike would precede the positive spike.

There are many digital derivative operators. A simple but effective one is the *Sobel filter*. There are two filters, one for detecting horizontal edges (on the left) and other for detecting vertical edges (on the right):

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}.$$

A characteristic of a derivative filter is that the sum of its elements must equal zero. The reason is that if the operator is applied to a pixel whose intensity is the same as that of all its neighbors, the result must be zero. Look again at Figs. 12.8b and 12.9a where the derivative is zero when the intensity is constant.

When the Sobel filters are applied to the pixel array in Fig. 12.8a, the result clearly detects that there is a horizontal edge (Fig. 12.10a) but no vertical edge (Fig. 12.10b).

Sobel filters are very powerful because they can not only detect an edge but also compute the angle of the edge within the image. Figure 12.11 shows an image with an edge running diagonally from the upper left to the lower right. The results of applying the two Sobel filters are shown in Fig. 12.12a, b. From the magnitudes and signs of the elements of these arrays, the angle of the edge can be computed as described in [3, Sect. 4.3.1].

(a)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	80	80	80	80	0
3	0	80	80	80	80	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

(b)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

Fig. 12.10 **a** Sobel horizontal edge. **b** Sobel vertical edge

	0	1	2	3	4	5
0	30	30	30	30	30	30
1	**50**	30	30	30	30	30
2	**50**	50	30	30	30	30
3	**50**	**50**	**50**	30	30	30
4	**50**	**50**	**50**	**50**	30	30
5	**50**	**50**	**50**	**50**	**50**	30

Fig. 12.11 Diagonal edge

(a)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	**60**	20	0	0	0
2	0	**60**	**60**	20	0	0
3	0	20	**60**	**60**	20	0
4	0	0	20	**60**	**60**	0
5	0	0	0	0	0	0

(b)

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	**−60**	−20	0	0	0
2	0	**−60**	**−60**	−20	0	0
3	0	−20	**−60**	**−60**	−20	0
4	0	0	−20	**−60**	**−60**	0
5	0	0	0	0	0	0

Fig. 12.12 **a** Sobel horizontal filter on a diagonal edge. **b** Sobel vertical filter on a diagonal edge

Activity 12.3: Detecting an edge

- Print out a pattern with a sharp edge (Fig. 12.13).
- Adapt the program from Activity 12.1 to cause the robot to sample and store the ground sensor as the robot moves over the pattern from left to right. Apply a derivative filter to the samples.
- During a second pass over the pattern, the robot indicates when the value of the derivative is not close to zero.
- What happens if the robot moves over the pattern from right to left?
- When applying the filter, the results must be stored in a separate array, not in the array used to store the pixels. Why?

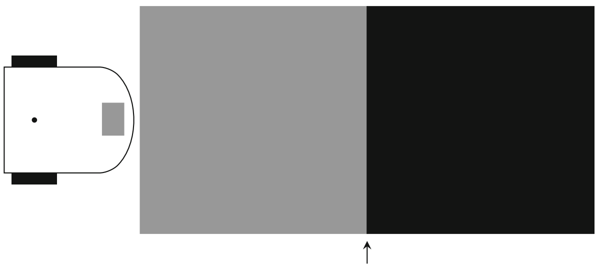


Fig. 12.13 Edge detection activity

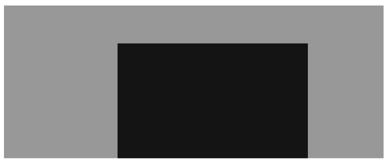
12.5 Corner Detection

The black rectangle within the gray background in Fig. 12.14a is more than just a set of edges. The vertical edges form two corners with the horizontal edge. Here we describe two algorithms for identifying corners in an image. For simplicity, we assume that the corners are aligned with the rectangular image.

We know how to detect edges in an image. A corner is defined by the intersection of a vertical edge and a horizontal edge. Figure 12.14b is the 6 × 10 pixel array for the image in Fig. 12.14a. If we apply the Sobel edge detectors to this pixel array, we obtain two vertical edges (Fig. 12.15a) and one horizontal edge (Fig. 12.15b).

The intersection is defined for pixels for which the sum of the absolute values in the two Sobel edge arrays is above a threshold. With a threshold of 30, the edges intersect in the pixels (2, 3) and (2, 7) which are the corners.

(a)



(b)

	0	1	2	3	4	5	6	7	8	9
0	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	30	30	30	30	30	30
2	30	30	30	**50**	**50**	**50**	**50**	**50**	30	30
3	30	30	30	**50**	**50**	**50**	**50**	**50**	30	30
4	30	30	30	**50**	**50**	**50**	**50**	**50**	30	30
5	30	30	30	**50**	**50**	**50**	**50**	**50**	30	30

Fig. 12.14 **a** Image of a corner. **b** Pixel array of a corner

(a)

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	**20**	**20**	0	0	0	**−20**	**−20**	0
2	0	0	**60**	**60**	0	0	0	**−60**	**−60**	0
3	0	0	**80**	**80**	0	0	0	**−80**	**−80**	0
4	0	0	**80**	**80**	0	0	0	**−80**	**−80**	0
5	0	0	0	0	0	0	0	0	0	0

(b)

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	**20**	**60**	**80**	**80**	**80**	**60**	**20**	0
2	0	0	**20**	**60**	**80**	**80**	**80**	**60**	**20**	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0

Fig. 12.15 **a** Vertical edges. **b** Horizontal edge

$$\begin{array}{c|cccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 8 & 7 & 6 & 5 & 5 & 5 & 6 & 7 & 0 \\
2 & 0 & 8 & 6 & \mathbf{3} & 5 & 5 & 5 & \mathbf{3} & 6 & 0 \\
3 & 0 & 8 & 5 & 5 & 8 & 8 & 8 & 5 & 5 & 0 \\
4 & 0 & 8 & 5 & 5 & 8 & 8 & 8 & 5 & 5 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Fig. 12.16 Similar neighbors

A uniform area, an edge and a corner can be distinguished by analyzing the neighbors of a pixel. In a uniform area, all the neighbors of the pixel have approximately the same intensity. At an edge, the intensities of the neighbors of the pixel are very different in one direction but similar in the other direction. At a corner, the intensities of the neighbors of the pixel show little similarity. To detect a corner, count the number of similar neighbors for each pixel, find the minimum value and identify as corners those pixels with that minimum value. Figure 12.16 shows the counts for the pixels in Fig. 12.14b. As expected, the corner pixels (2, 3) and (2, 7) have the minimum number of similar neighbors.

Activity 12.4: Detecting a corner

- Implement corner detection by intersecting edges using a robot with two ground proximity sensors. The robot moves from the bottom of the image in Fig. 12.14a to the top. If placed over the black rectangle it does not detect a corner, while if it is placed so that one sensor is over the black rectangle and the other over the gray background it does detect the corner.
- Implement corner detection by similar neighbors. Repeatedly, check the current samples from the left and right sensors and the previous samples from the left and right sensors. If only one of the four samples is black, a corner is detected.

12.6 Recognizing Blobs

Figure 12.17a shows a *blob*: a roughly circular area of 12 pixels with high intensity on a background of low intensity. The blob does not have a well-defined boundary like a rectangle. The next to the last row also shows two artifacts of high intensity that are not part of the blob, although they may represent distinct blobs. Figure 12.17b shows the pixels after adding random noise. The task is to identify the blob in the presence of noise, without depending on a predefined intensity threshold and ignoring the artifacts. The independence of the identification from the overall intensity is

(a)	0	1	2	3	4	5	6	7	8	9
0	30	30	30	30	30	30	30	30	30	30
1	30	30	30	30	**80**	**80**	30	30	30	30
2	30	30	30	**80**	**80**	**80**	**80**	30	30	30
3	30	30	30	**80**	**80**	**80**	**80**	30	30	30
4	**80**	30	30	30	**80**	**80**	30	30	30	**80**
5	30	30	30	30	30	30	30	30	30	30

(b)	0	1	2	3	4	5	6	7	8	9
0	46	42	40	50	46	44	40	33	30	34
1	32	46	46	46	**67**	**73**	39	47	39	30
2	33	40	40	**73**	**68**	**63**	**73**	44	42	31
3	35	41	50	**67**	**60**	**71**	**60**	37	30	49
4	**68**	46	32	44	**61**	**77**	48	42	45	**62**
5	39	37	38	34	33	40	35	37	34	32

Fig. 12.17 **a** Blob. **b** Blob with noise

(a)	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	67	73	0	0	0	0
2	0	0	0	73	68	63	73	0	0	0
3	0	0	0	67	60	71	60	0	0	0
4	68	0	0	0	61	77	0	0	0	62
5	0	0	0	0	0	0	0	0	0	0

(b)

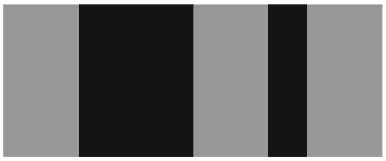


Fig. 12.18 **a** Blob after threshold. **b** Blob to detect

important so that the robot can fulfill its task regardless of the lighting conditions in the environment.

To ignore noise without predefining a threshold, we use a threshold which is defined in terms of the average intensity of the image. To separate the blobs from one another, we first find a pixel whose intensity is above the threshold and then *grow* the blob by adding neighboring pixels whose intensity is above the threshold. For the noisy image in Fig. 12.17b, the average intensity is 54. Since the blob presumably occupies a relatively small part of the background, it might be a good idea to take a threshold somewhat higher than the average, say 60.

Figure 12.18a shows the image after assigning 0 to all pixels below the threshold. The blob has been detected but so have the two artifacts. Algorithm 12.1 is an algorithm for isolating a single blob. First, search for some pixel that is non-zero; starting from the top left, this will be pixel $p_1 = (1, 4)$ with intensity 67. Now grow the blob that adding all neighbors of p_1 whose intensities are non-zero; these are $p_2 = (1, 5)$, $p_3 = (2, 3)$, $p_4 = (2, 4)$, $p_5 = (2, 5)$. Continue adding non-zero neighbors of each p_i to the blob until no more pixels are added. The result will be the 12-pixel blob without the artifacts at $(4, 0)$, $(4, 9)$.

The algorithm works because the first non-zero pixel found belonged to the blob. If there were an isolated non-zero pixel at $(1, 1)$, this artifact would have been detected as a blob. If there is an estimate for the minimum size of a blob, the algorithm should be followed by a check that the blob is at least this size.

Algorithm 12.1: Detecting a blob

```
        integer threshold
        pixel p
        set not-explored ← empty-set
        set blob ← empty-set

1: set the threshold to the average intensity
2: set pixels below threshold to zero
3: find a non-zero pixel and add to not-explored
4: while not-explored not empty
5:     p ← some element of not-explored
6:     add p to blob
7:     remove p from not-explored
8:     add non-zero neighbors of p to not-explored
```

Check that Algorithm 12.1 is not sensitive to the intensity level by subtracting the constant value 20 from all elements of the noisy image (Fig. 12.17b) and rerunning the algorithm. It should still identify the same pixels as belonging to the blob.

Activity 12.5: Detecting a blob

- Write a program that causes the robot to sample the ground sensor as it moves from left to right over the pattern in Fig. 12.18b.
- Compute the average intensity and set the threshold to the average.
- On a second pass over the pattern, after detecting the first sample from the black rectangle that is below the threshold, the robot provides an indication (by light or sound) as long as it moves over the rectangle.
- The robot should consider the second black rectangle as an artifact and ignore it.

Activity 12.6: Recognizing a door

- In Fig. 12.19a, the gray rectangle represents an open door in a dark wall represented by the black rectangles. Figure 12.19b represents a dark wall between two gray open doors. If you run the program from Activity 12.3, you will see that two edges are detected for both patterns. Modify the program so that the robot can distinguish between the two patterns.

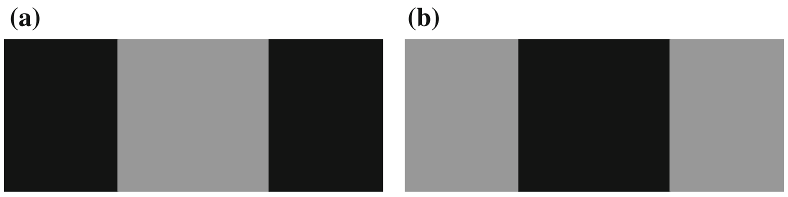


Fig. 12.19 **a** Recognize the door. **b** This is not a door

12.7 Summary

In human beings and most animals vision is the most important sensor and a large portion of the brain is devoted to interpreting visual signals. Robots can use vision to perform advanced tasks within an environment that is constantly changing. The technology of digital cameras is highly advanced and the cameras can transfer high-resolution pixel arrays to the robot's computer. Algorithms for digital image processing enhance and interpret these images.

Enhancement algorithms remove noise, improve contrast and perform other operations that do not depend on what objects appear in an image. They use spatial filters that modify the intensity of each pixel based on the intensities of its neighbors. Histogram modification uses the global distribution of intensities in an image to modify individual pixels.

Following image enhancement, algorithms identify the objects in the image. They start by detecting simple geometric properties like edges and corners, and then proceed to identify the objects that appear in the image.

12.8 Further Reading

Gonzalez and Woods [1] is a comprehensive textbook on digital image processing that includes the mathematical fundamentals of the topic. Russ [2] is a reference work on image processing. Szeliski [4] is a book on computer vision which goes beyond image processing and focuses on constructing 3D models of images. For applications of image processing in robotics, see [3, Chap. 4].

References

1. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Pearson, Boston (2008)
2. Russ, J.C.: The Image Processing Handbook, 6th edn. CRC Press, Boca Raton (2011)
3. Siegwart, R., Nourbakhsh, I.R., Scaramuzza, D.: Introduction to Autonomous Mobile Robots, 2nd edn. MIT Press, Cambridge (2011)
4. Szeliski, R.: Computer Vision: Algorithms and Applications. Springer, Berlin (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.