AA274A: Principles of Robot Autonomy I
Course Notes

Oct 22, 2019

11 Image Processing, Feature Detection and Description

Once an autonomous system is able to collect visual information from its surroundings, it must perform a series of operations to develop a representation of the structure and surroundings captured in this raw visual input. This allows the system to interpret its surroundings, and the ability to do so accurately is fundamental to successful execution of downstream autonomous operations such as localization, mapping, and decision-making. In this lecture, several key steps will be outlined which contribute to the extraction of such information from images. This lecture will describe some of the most common steps of the image processing pipeline, including filtering, differentiation, feature detection, and feature description[1].

Image processing is a form of signal processing where the input signal is an image (such as a photo or a video) and the output is either an image or a set of parameters associated with the image. Most image-processing techniques treat the image as a two-dimensional signal $I(x, y)$ where x and y are the spatial image coordinates and the amplitude of I at any pair of coordinates (x, y) is called intensity or gray level of the image at that point. Image processing is a huge field and typical operations, and therefore we focus only on the most important image processing operations that are relevant for robotics, which are image filtering operations such as smoothing and edge detection.

This section will first describe a workflow by which raw image data (in the form of pixels and color values) can be transformed into a workable representation of the image. From this representation, local features can be used to make inferences on more complex image properties such as similarity of images, object detection, depth / structure identification, and tracking of objects between frames.

[1] Unless stated otherwise, most of this lecture note is a direct excerpt from [SNS11])

11.1 Image Filtering

Image filtering is one of the principal tools in image processing. The word filter comes from frequency domain processing, where "filtering" refers to the process of accepting or rejecting certain frequency components. For example, a filter that passes low frequencies is called a *lowpass* filter. The effect produced by a lowpass filter is to blur (smooth) an image, which has the main effect of reducing image noise. Conversely, a filter that passes high frequencies is called *highpass* filter and is typically used for edge detection. Image filters can be implemented both in the frequency domain and in the spatial domain. In the latter case, the filter is called *mask* or *kernel*. In this section, we will review the fundamentals of spatial filtering (direct excerpt from [SNS11]).

We start by considering an image as a function I: $[a, b] \times [c, d] \to [0, L]$, where $I(x, y)$ represents the grayscale pixel intensity at (x, y) for $a \leq x \leq b$ and $c \leq y \leq d$. Assuming we have a two-dimensional grayscale image, the image is represented as a two-dimensional matrix with r rows and c columns. Each entry is equal to the discretized brightness level; often we use 256 levels, where 0 corresponds to black and 255 to white. (Note that 256 is chosen because it corresponds to the number of levels that can be represented in a single byte, $2^8 = 256$). For a color image, each entry in the matrix would be a vector function with three components, one each for the red, green, and blue color channels.

OpenCV implements a number of image filtering algorithms. More information can be found in [opea].

11.2 Spatial Filters

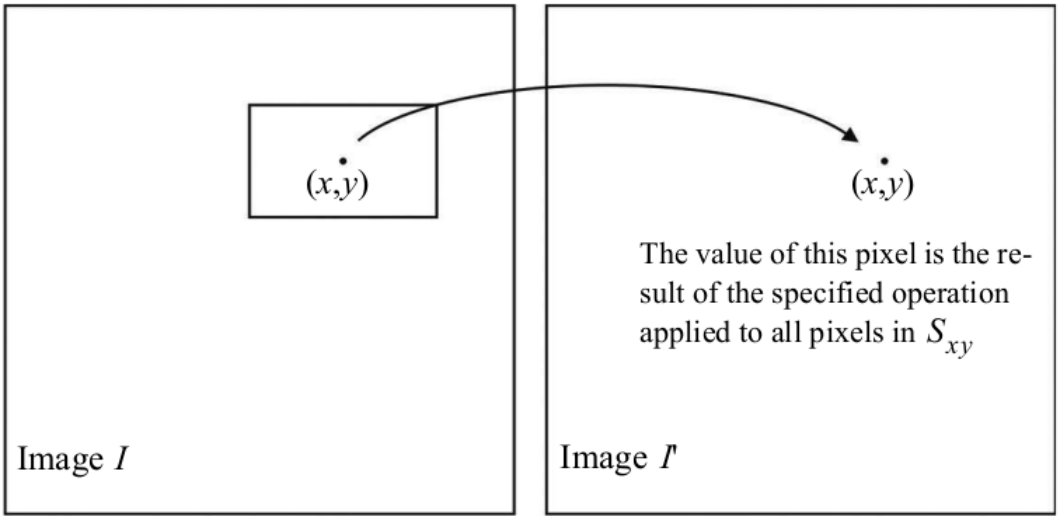


Figure 1: Illustration of the concept of spatial filtering [SNS11].

In Figure 1, the basic principle of spatial filtering is explained[2]. A spatial filter consists of (1) a neighborhood of the pixel under examination (typically a small rectangle), and (2) a predefined operation T that is performed on the image pixels encompassed by the neighborhood. Let S_{xy} denote the set of coordinates of a neighborhood centered on an arbitrary point (x, y) in an image I. Spatial filtering generates a corresponding pixel at the same coordinates in an output image I' where the value of that pixel is determined by a specified operation on the pixels in S_{xy}. For example, suppose that the specified operation is to compute the average value of the pixels in a rectangular window of size $m \times n$ centered on (x, y). The locations of pixels in this region constitute the set S_{xy}. Figure 1 illustrates the process. We can express this operation in equation form as

$$I'(x, y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} I(r, c) \tag{1}$$

where r and c are the row and column coordinates of the pixels in the set S_{xy}. The new image I' is created by varying the coordinates (x, y) so that the center of the window moves from pixel to pixel in image I.

The filter used to illustrate the example above is called *averaging filter*. More generally, the operation performed on the image pixels can be *linear* or *nonlinear*. In these cases, the filter is called either a linear or nonlinear filter. Here, we concentrate on linear filters. In general, linear spatial filtering of an image with a filter F of size $(2n+1) \times (2m+1)$ is given by the expression

$$I'(x, y) = F \circ I = \sum_{i=-n}^{n} \sum_{j=-m}^{m} F(i, j) I(x + i, y + j). \tag{2}$$

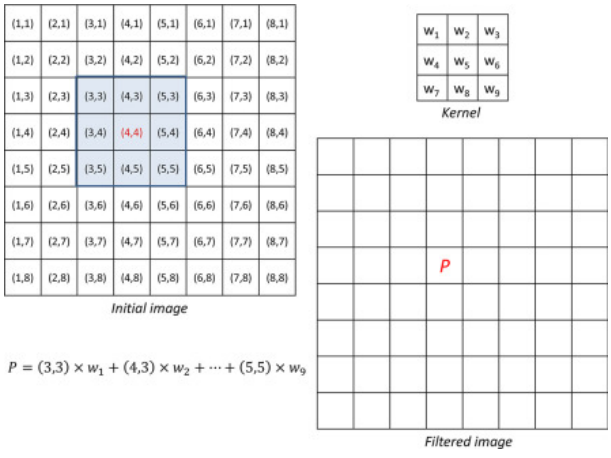


Figure 2: Correlation of initial image with a kernel to produce a filtered image

[2]Much of this section is a direct excerpt from Ch4.3 of [SNS11]

In equation 2, I' represents the filtered image, F represents the filter mask (sometimes called the kernel or window), and I represents the original image. The filter mask F is of size $(2n+1) \times (2m+1)$, where each entry represents the weighting of a particular neighbor pixel from the original image. We apply this filter mask to every pixel (x, y) by multiplying each intensity value in the neighborhood of (x, y) by a certain weight $F(i, j)$ in order to get to the filtered image I'. This procedure is referred to as a *correlation* with the kernel F, and is illustrated in figure 2.

As we apply this filter, we have to decide what to do at the boundaries of our image. There are a number of options, including padding the image, cropping it, extending it, or wrapping it. However, as images are generally quite large, the exact treatment chosen for the edges does not vary the final result significantly.

Generating linear spatial filters requires that we specify the mn coefficients of the kernel. These coefficients are chosen based on what the filter is supposed to do. In the next section, we will see how to select these coefficients.

11.2.1 Average

The average filter that returns the average of the pixels in the mask is an example of a smoothing filter. It achieves a smoothing effect by removing sharp features. Assuming a 3×3 mask, the filter can be written as

$$F = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

where all the coefficients sum to 1. This normalization is important to keep the same value as the original image if the region by which the filter is multiplied is uniform. Also note that, instead of being 1/9, the coefficients of the filter are all 1s. The idea is that the pixels are first summed up and the result is then divided by 9. Indeed, this is computationally more efficient than multiplying each element by 1/9.

This filter is commonly used to reduce noise and smooth or blur an image, as seen in Figure 3. The *normalization* is done so that the overall brightness of the image remains constant. The OpenCV implementation of average filter can be found at [Opeb].

11.2.2 Gaussian Smoothing

Many image-processing algorithms make use of the second derivative of the image intensity. Because of the susceptibility of such high-order derivative algorithms to changes in illumination in the basic signal, it is important to smooth the signal so that changes in intensity are due to real changes in the luminosity of objects in the scene rather than random variations due to imaging noise. A standard approach is the use of a Gaussian averaging filter, which is another example of linear spatial filter. Gaussian filter is often preferred over a simple average filter because it gives higher weights to pixels closer to the center of the filter and

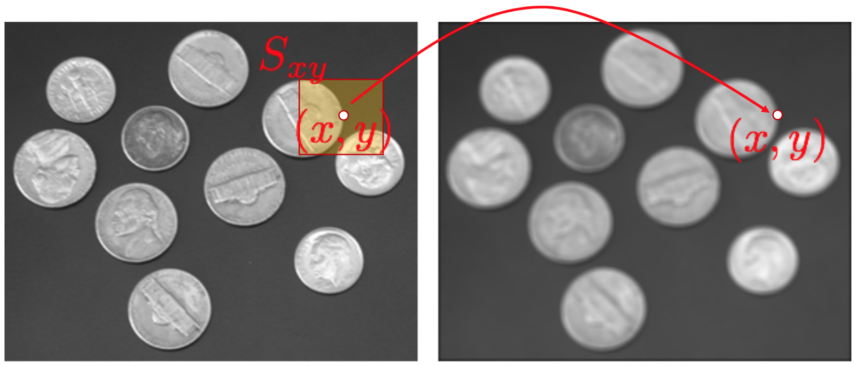


Figure 3: Example of an average filter

lower weights to those along the edge. Weights of Gaussian filter are given by

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(- \frac{x^2 + y^2}{2\sigma^2} \right). \tag{3}$$

The function is sampled about its center to obtain the mask. For example, for a normalized 3×3 mask with $\sigma = 0.85$,

$$G = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

11.2.3 Convolution

Convolution is a linear filter that is similar to correlation, but has reverse image indices. Correlation and convolution are identical when the filter is symmetric. The convolution equation is shown in 4.

$$I'(x, y) = F * I = \sum_{i=-n}^{n} \sum_{j=-m}^{m} F(i, j) I(x - i, y - j) \tag{4}$$

Convolution is associative, meaning that $F * (G * I) = (F * G) * I$. We commonly use this property when trying to smooth an image and then take the derivative of the result to obtain the gradient. Instead of doing these sequentially, we can combine the two operations into one by first convolving the derivative filter with our smoothing filter (e.g. a Gaussian filter), then convolving the resulting filter with the image. This second approach is advantageous because it is less computationally intensive.

11.2.4 Separable Masks

A mask is separable if it can be broken down into the convolution of two kernels $F = F_1 * F_2$. If a mask is separable into "smaller" masks, then it is often cheaper to apply F_1 followed by

F_2, rather than by F directly. One special case of this is when the mask can be represented as outer product of two vectors (meaning it is equivalent to the 2D convolution of those two vectors). If the mask is of shape $M \times M$, and the input image has size $w \times h$, then the computational complexity of directly performing the convolution is $O(M^2wh)$. However, if one were to separate the mask into its component vectors and convolve these components with the input image, the computational cost would be $O(2Mwh)$, which is linear, not quadratic in M.

The average mask can be separated as shown below.

$$F = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \frac{1}{9}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$$

The Gaussian mask is also separable because the exponential function can be separated out. The Gaussian smoothing separable mask is shown below in Equation 5.

$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= g_\sigma(x)\cdot g_\sigma(y) \end{aligned} \tag{5}$$

Decomposing a mask into its separable components does not alter the results but can drastically improve the computational speed of your code.

11.3 Image Differentiation

Taking the derivative of an image is similar to differentiating a function with a two-dimensional domain. On a basic level, the derivative of an image represents changes in pixel intensity in both the vertical and horizontal direction. Because images are discrete functions, the traditional method for differentiating continuous functions can not be used. One popular method to differentiate an image is known as the central difference method; the equations for this method are shown below in Equation 6. These equations find the change in intensity between the neighboring pixels.

$$\begin{aligned} \frac{\partial I}{\partial x} &= \frac{I(x+1,y)-I(x-1,y)}{2} \\ \frac{\partial I}{\partial y} &= \frac{I(x,y+1)-I(x,y-1)}{2} \end{aligned} \tag{6}$$

Here, I is the intensity of the image, and x and y define the location of the pixel of interest. Note that applying a normalization factor to these equations does not effectively change the result, as the background intensities will be scaled by the same factor. The derivative with respect to x is the derivative in the horizontal direction, and the derivative with respect to

y is in the vertical direction. The central difference method looks at the neighboring pixels on both sides of the pixel of interest. It is also valid to just consider one side of each pixel to calculate the derivative, for example $\frac{\partial I}{\partial x} = I(x+1, y) - I(x, y)$. While this method would be correct, using both neighboring sides is generally better at estimating the derivative right at the pixel of interest.

It is also possible to differentiate an image using convolution operations, such as with a Sobel mask. Sobel masks work similarly to the masks discussed earlier in this lecture. The Sobel masks for the x and y direction are given below.

$$S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel masks are similar to the central difference method described above in that both approaches use the intensity of neighboring pixels to find the derivative, but Sobel masks use more neighboring pixels when calculating the derivative, applying a weighting factor to each pixel. We can see that more weight is given to the pixels directly adjacent to the pixel of interest.

For example, S_x weights the rows above and below the row of interest to calculate the horizontal derivative, and S_y weights the columns to the left and the right of the column of interest to find the vertical derivative. In essence, the S_x operator takes the central difference of the the row of interest, as well as the rows above and below the pixel location, and then takes a weighted average of those central differences to make the differentiation more robust.

11.4 Similarity Measures

Filtering can also be used to find similar features in different images. These are particularly used to solve the correspondence problem in structure from stereo and structure from motion. The methods we are about to describe are all area-based: these algorithms consider a small patch (window) in one image and look for the most similar patch in the second image by means of an appropriate correlation measure. Suppose that we want to compare a $m \times n$ patch in image I_1 centered on (x, y) with another patch of the same size centered on (x', y') in image I_2 . We assume that these are odd integers, therefore $m = 2a + 1$ and $n = 2b + 1$. The similarity is then computed between the gray intensity levels of the two patches. Two useful examples of similarity measures are the sum of absolute differences (SAD) and the sum of squared differences (SSD). The equations for each are given below in (7) and (8), respectively.

$$SAD = \sum_{i=-n}^{n} \sum_{j=-m}^{m} |I_1(x+i, y+j) - I_2(x'+i, y'+j)| \tag{7}$$

$$SSD = \sum_{i=-n}^{n} \sum_{j=-m}^{m} [I_1(x+i, y+j) - I_2(x'+i, y'+j)]^2 \tag{8}$$

The SAD is the simplest among these similarity measures. It is calculated by subtracting pixels between the reference image I_1 and the target image I_2 followed by the aggregation of absolute differences within the patch. The SSD has a higher computational complexity compared to the SAD, since it involves numerous multiplication operations (i.e., squared). Notice that if the left and right images match perfectly, the resultant of SAD and SSD will be zero. SAD and SSD tend not to be very robust, but they do provide a starting point for comparing image features.

11.5 Image Feature Extraction

In this section, we define the concept of the local feature and review some of the most consolidated feature extractors. As the computer vision literature in this field is very large, we will only describe in detail the two most popular feature detectors, namely Harris and SIFT, and will briefly introduce the others by explaining the main advantages and disadvantages and domain of application.

A local feature is an image pattern that differs from its immediate neighborhood in terms of intensity, color, and texture. Local features can be small image patches (such as regions of uniform color), edges, or points (such as corners originated from line intersections). In the modern terminology, local features are also called interest points, interest regions, or keypoints.

Depending on their semantic content, local features can be divided into three different categories. In the first category are features that have a semantic interpretation such as, for instance, edges corresponding to lanes of the road or blobs corresponding to blood cells in medical images. This is the case in most automotive applications, airborne images, and medical image processing. Furthermore, these were also the first applications for which local feature detectors have been proposed. In the second category are features that do not have a semantic interpretation. Here, what the features actually represent is not relevant. What matters is that their location can be determined accurately and robustly over time. Typical applications are feature tracking, camera calibration, 3D reconstruction, image mosaicing, and panorama stitching. Finally, in the third category are features that still do not have a semantic interpretation if taken individually, but that can be used to recognize a scene or an object if taken all together. For instance, a scene could be recognized counting the number of feature matches between the observed scene and the query image. In this case, the location of the feature is not important; only the number of matches is relevant. Application domains include texture analysis, scene classification, video mining, and image retrieval (see, for instance, Google Images, Microsoft Bing Images or Youtube). This principle is the basis of the visual-word-based place recognition.

11.6 Detectors

Detectors detect local features. We will focus on the most common detectors in robotics, i.e., edge detectors, corner detectors and blob detectors.

11.6.1 Edge Detection

Introduction

Let us define an edge as "a region in an image where there is a significant change in intensity values along one direction, and negligible change along the orthogonal direction". In one dimension, an edge corresponds to a point where there is a sharp change in intensity, or more explicitly, a large first derivative and a small second derivative. Many edge detectors rely on differentiating images and looking for spikes in the derivative.

Some criteria necessary for robust edge detection include accuracy, localization, and single response. To elaborate, good accuracy implies few false positives or negatives (missed edges). Good localization implies that the detected edge should be exactly where the true edge is in the image. A single response implies one edge is detected for each real edge. For example, one real edge should not be detected as two separate edges. Noise and discretization may challenge our ability to detect edges.

Most edge detection methods rely on two key steps: smoothing and differentiation. Differentiation is performed in both the vertical and horizontal directions to find locations with high intensity gradients in just one of the two directions. However, differentiation alone is vulnerable to false positives due to image noise, which may be gaussian additive noise or otherwise such as salt & pepper noise (where random pixels jump to either complete white or complete black) [SJV09]. In order to lower the noise, many algorithms will smooth the image before differentiating it. The next section explores edge detection further.

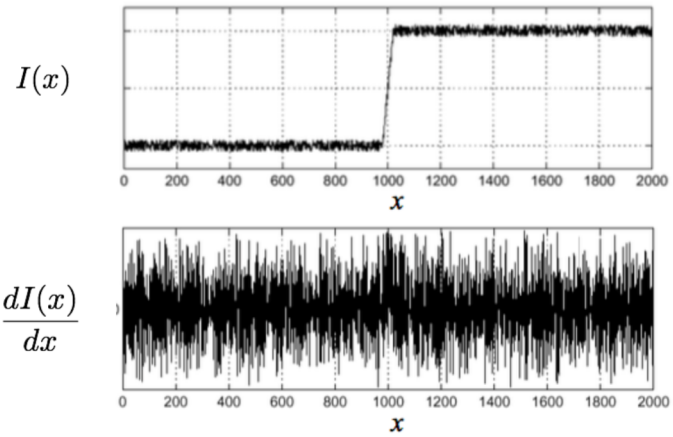


Figure 4: Differentiation of a noisy signal

Techniques. In this part we will discuss classic edge detection techniques. They involve the use of convolution with a kernel to extract edges. For example, differentiating every point in our signal should only give non zero values (or at least high values) at each edge, allowing us to sort them easily. However we will see that this is not always the case as our signal/image is usually not as smooth as we would like it to be.

Let us first explore the 1D case to understand the principle of edge detectors and how they behave under high noise. For example, looking at Figure 4, we see that simply differentiating the signal without performing any sort of filtering will yield a completely useless result, as noise causes random values of the derivative at every point. There is no detection threshold that will allow us to detect edges from this differentiated signal.

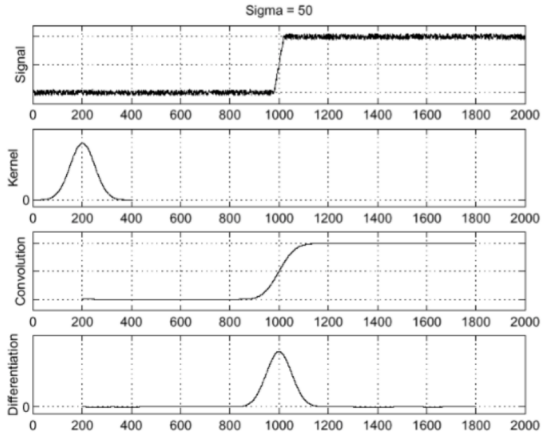


Figure 5: Edge detection through convolution with Gaussian kernel, followed by differentiation

We can remedy this problem by filtering the signal. For example we may use a Gaussian filter to apply a smoothing mask, the result of which can be differentiated to identify edges (by identifying the highest value or the point where the second derivative cancels). To do so, we perform the following convolutions:

$$s(x) = g_\sigma(x) * I(x)$$
$$s'(x) = \frac{d}{dx} * s(x)$$

Where I is the original signal, g_σ is the Gaussian, and s' is the final mask. The performance of such a procedure may be seen in Figure 5.

One issue with this method is that it requires that we compute two convolutions on our image, which can be computationally expensive. It is possible to exploit the associativity of the convolution to perform the same transformation with a single convolution:

$$s' = \frac{d}{dx} * (g_\sigma * I) = (\frac{d}{dx} * g_\sigma) * I$$

Therefore the function $g'_\sigma = \frac{d}{dx} * g_\sigma$ may be used for much faster computation. An illustration of this process may be seen in Figure 6.

Now we might take inspiration from this 1D case to design a 2D edge detector. Let G_σ be the two dimensional Gaussian distribution, i.e. $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(- \frac{x^2+y^2}{2\sigma^2} \right)$. An illustration of this function and its derivative may be found in Figure 7.

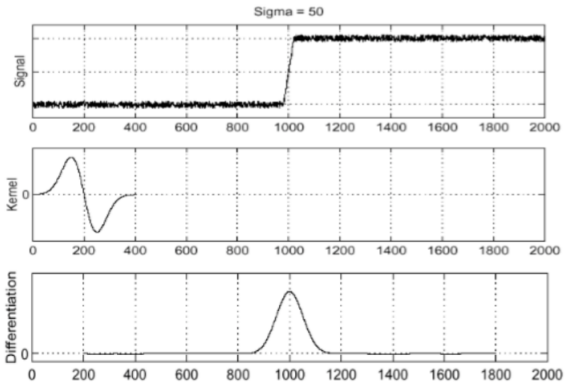
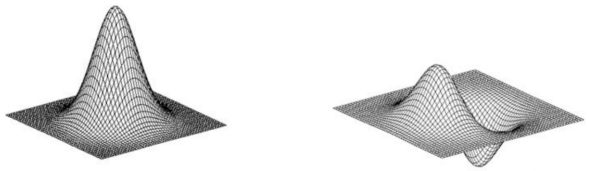


Figure 6: Edge detection through convolution with a differentiated Gaussian kernel



$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \qquad \frac{\partial G_\sigma(x,y)}{\partial x}$$

Figure 7: 2D Gaussian function

The gradient of the smoothed image may be written as:

$$\nabla S = \begin{bmatrix} \frac{\partial}{\partial x} * G_\sigma * I \\ \frac{\partial}{\partial y} * G_\sigma * I \end{bmatrix} = \begin{bmatrix} G_{\sigma,x} * I \\ G_{\sigma,y} * I \end{bmatrix} = \begin{bmatrix} S_x \\ S_y \end{bmatrix}$$

We notice that once again we are able to reduce the number of convolutions to 1 by building the partial derivatives of the Gaussian: $G_{\sigma,x}$ and $G_{\sigma,y}$. As in the 1D case, our next step is to check $|\nabla S| = \sqrt{S_x^2 + S_y^2}$ against an aptly chosen threshold to detect edges. We may additionally filter out points that are above the threshold but not local maxima to guarantee thin edges.

Finally, an application of this method on a picture classically used to demonstrate image processing techniques may be found in Figure 8.

11.6.2 Corner Detector

A corner in an image is defined as an intersection of two or more edges, or as defined by Moravec [Mor77], a point where there is a large intensity variation in every direction. As shown in Fig. 9, for a window which is centered at a pixel in a region of uniform intensity (a "flat" region), its adjacent windows in all directions will be identical. For a window centered at a pixel on an edge, the adjacent windows will look the same except for the ones in the

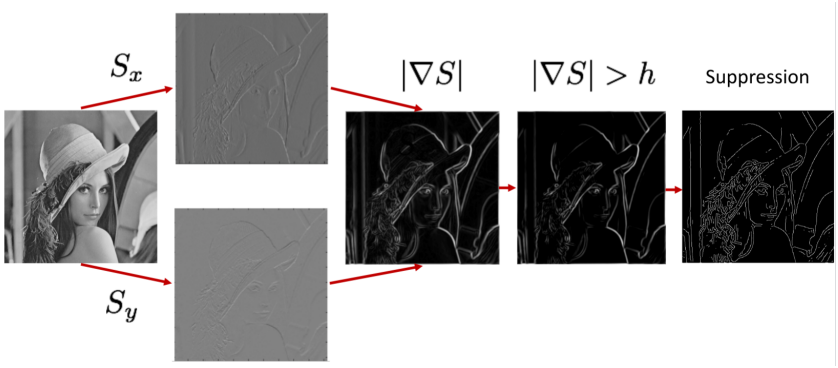


Figure 8: Classic example of 2D edge detection

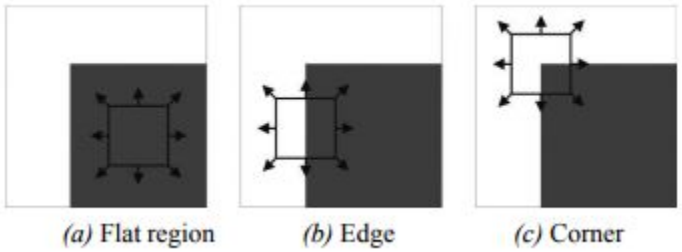


Figure 9: a) "Flat" region: no change in all directions. b) "Edge": no change along the edge direction. c) "Corner": significant changes in all directions. [SNS11]

direction perpendicular to the edge. For a window centered at a pixel on a corner, the adjacent windows in any direction will look different.

Moravec's method identifies a corner where the Sum of Squared Difference (SSD) is at its local maxima. The SSD between image patch centered on pixel coordinate (u, v) and the image patch offset by (x, y) is given by:

$$SSD = \sum_{i=-n}^{n} \sum_{j=-m}^{m} [I_1(x+i, y+j) - I_2(x'+i, y'+j)]^2 \tag{9}$$

where I denotes the intensity of a grayscale image.

11.6.3 Desired Properties of Corner Detectors

The most important desired properties of corner detectors are repeatability and distinctiveness. Having detectors with "repeatability" means that the same features taken at the same scene under different viewpoint and illumination conditions can be found in multiple images, which requires the features to be invariant to geometric and photometric transformations. Having detectors with "distinctiveness" means the information carried by the patch surrounding the keypoints are highly distinctive, therefore a reliable correspondence can be

established, and feature points can be distinguished and matched. Good repeatability and distinctiveness are both desirable for point matching and applications such as panorama stitching and 3D reconstruction.

Harris Corner Detector Instead of the shifting windows, the Harris corner detector [HS88] uses partial derivatives of the SSD which improves upon Moravec's corner detector. Harris and Stephens first approximated $I(u+x, v+y)$ with a first-order Taylor expansion; the SSD can be expressed as follows:

$$SSD(x,y) \approx \sum_u \sum_v (I_x(u,v)x + I_y(u,v)y)^2 \tag{10}$$

where I_x and I_y are the partial derivatives in the x, y direction respectively. The quadratic form in Eq. 10 can be written in a matrix form:

$$SSD(x,y) \approx \begin{bmatrix} x & y \end{bmatrix} M \begin{bmatrix} x \\ y \end{bmatrix} \tag{11}$$

where M is the second moment matrix

$$M = \begin{bmatrix} \sum\sum I_x^2 & \sum\sum I_x I_y \\ \sum\sum I_x I_y & \sum\sum I_y^2 \end{bmatrix} \tag{12}$$

Since M is symmetric, M can be decomposed as:

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \tag{13}$$

where λ_1 and λ_2 are the eigenvalues of M.

As shown in Fig. 10, if both λ_1 and λ_2 are small, the change in intensity in both directions is small, and therefore a flat region can be identified; if both λ_1 and λ_2 are large, indicating a large SSD in both directions, then a corner is detected; if one eigenvalue is significantly higher than the other, the SSD is dominant in one direction, which indicates an edge. In order to reduce the complexity of the computation (i.e., avoiding calculation of eigenvalues), Harris and Stephens introduced the "cornerness function," C, so that only the determinant and the trace of matrix M need to be computed.

$$C = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(M) - \kappa \cdot trace^2(M) \tag{14}$$

Here κ is an empirically determined tunable sensitivity parameter, usually in the range of 0.04–0.15. The Harris corner detector's last step is then to extract local maxima of the cornerness function using non-maxima suppression (an algorithm that preserves only the maxima in a region).

The Harris corner detector is invariant to 2D rotations, as the eigenvalues of the second moment matrix M don't change under pure rotation. The Harris corner detector is also

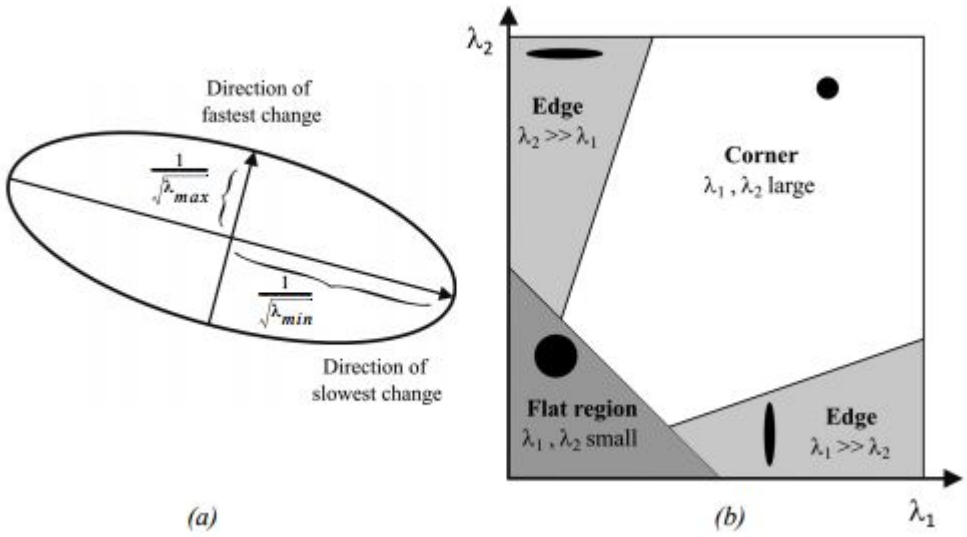


Figure 10: a) Visualization of direction of the fastest and slowest change of intensity built from matrix M. b) Harris and Stephens' classification of corners and edges [SNS11]

invariant to affine intensity changes ($I' = aI + b$) because the eigenvalues and the cornerness function are rescaled by a constant factor and the locations of local maxima remain unchanged. However, the Harris corner detector is not invariant to geometric affine transforms and image scaling due to the discretization of the image into pixels. Under a geometric affine transform, the neighborhood of the feature along x and y directions can be distorted, varying the curvature differently in the x and y directions. When the image is scaled, the region of the coner detector may become too large or too small. For example, it is possible for a corner to be identified as a corner at a lower scale, but an edge at a higher scale.

11.7 Descriptors

Descriptors describe keypoints (local features) to be compared across images or used for object detection or matching. Like the detectors, it is also desirable for descriptors to be repeatable and distinct. A naive descriptor is an $n \times m$ window of pixel intensities centered at a keypoint, which can be normalized to be illumination invariant; however, it is not invariant to pose and scale, and is poorly distinctive.

11.7.1 SIFT Features

SIFT stands for Scale Invariant Feature Transform and is a method to detect and match robust keypoints, which was invented in 1999 by Lowe [Low04]. The uniqueness of SIFT is that these features are extremely distinctive and can be successfully matched between images with very different illumination, rotation, viewpoint, and scale changes. Its high repeatability and high matching rate in very challenging conditions have made SIFT the

best feature detector so far. It has found many applications in object recognition, robotic mapping and navigation, image stitching (e.g. panoramas, mosaics), 3D modeling, gesture recognition, video tracking, and face recognition.

The main advantage of the SIFT features in comparison to all previously explained methods is that a "descriptor" is computed from the region around the interest point, which distinctively describes the information carried by the feature. This descriptor is a vector that represents the local distribution of the image gradients around the interest point. Since SIFT not only incorporates all the properties of the scale-affine-invariant Harris, but is also extremely distinctive and robust to image noise, small illumination variations, and viewpoint changes, it is known for its high repeatability and high matching rate even in very challenging conditions, and it has many applications in object, gesture and face recognition, robotic perception and 3D modeling. SIFT features are essentially local distributions of the image gradient around keypoints with highly informative content. The following briefly introduces the main steps of the SIFT algorithm. More details can be found in [Low04].

1. Scale-space extrema detection: Keypoints are detected by searching for stable features across the scale space (a continuous function of scale). The stable keypoint locations are identified by computing the difference of Gaussian (DoG) of two nearby scale space image produced from repeatedly convolving Gaussians with the initial image, and selecting the local extrema where the sample point is the largest or smallest among its eight neighbors in the current image and nine neighbors in the scale above and below.

2. Keypoint localization: The keypoint localization step fits the nearby data for location, orientation, scale and ratio of principal curvatures around keypoints candidates, and rejects the ones that are below some contrast threshold or poorly localized along an edge.

3. Orientation assignment: This step assigns an orientation to a keypoint as the peak in the histogram of gradient orientation in the neighborhood of the keypoint. The keypoint descriptor is then represented relative to the dominant orientation assigned, which allows the descriptor to be rotation invariant. In the case when there are multiple peaks that are more than 80% of the highest peak, multiple keypoints will be created all additional peaks, at the same location but with different assigned orientations.

4. Keypoint descriptor: The previous steps define local image regions by location, scale and orientation, and this step computes a highly distinctive and repeatable descriptor for this local image region. Each keypoint descriptor is built by stacking gradient histogram with 8 orientation bins for each 4×4 subregions of the neighboring region of the keypoints, and is therefore $4 \times 4 \times 8 = 128$ in length. The length of keypoint descriptors can be changed by tuning the block size and orientation bin size; however, the 128 element vector is empirically determined to be the most robust to image variations.

References

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *the 4th Alvey Vision Conference*, 1988.

[Low04] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91, 2004.

[Mor77] H. P. Moravec. Towards automatic visual obstacle avoidance. In *5th International Joint Conference on Artificial Intelligence*, 1977.

[opea] Image filtering — opencv 3.0.0-dev documentation. `https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html`. (Accessed on 10/22/2019).

[Opeb] Opencv: Image filtering. `https://docs.opencv.org/3.1.0/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37`. (Accessed on 10/22/2019).

[SJV09] S. Esakkirajan S. Jayaraman and T. Veerakumar. *Digital Image Processing*. Tata McGraw-Hill Education, 2009.

[SNS11] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.