Lecture Notes, Week 5: Derivatives, Edges, Corners

1 Local Features

We will start talking about various local image representations that give rise to more robust image representations then raw pixels. For example, one of such fundamental representations are edges (or contours). Edges are largely invariant to lighting as they represent boundaries between regions. A brightness/lighting change will generally change the grayscale values, but would not change the boundaries between the regions.

2 Image Derivatives

Remember that we can treat a continuous image incident on the camera sensor as a function $i(x, y)$. This also allows us to compute derivatives of the image. Given that the function is of two spatial variables, we will get two partial derivatives that jointly form a *gradient* vector. Formally, with simple definition from calculus we get:

$$\frac{\partial i(x,y)}{\partial x} = \lim_{\delta x \to 0} \frac{i(x+\delta x, y) - i(x,y)}{\delta x}$$
$$\frac{\partial i(x,y)}{\partial y} = \lim_{\delta y \to 0} \frac{i(x, y+\delta y) - i(x,y)}{\delta y}.$$

For the discretized image, that results after the camera sensor array, we can employ a discrete approximation, where $\delta x = 1$ and $\delta y = 1$ (because this is the smallest distance between pixels). This gives rise to the following discrete numerical approximation to the derivatives above:

$$I_x(X,Y) = \frac{\partial I(X,Y)}{\partial X} \approx \frac{I(X+1,Y) - I(X,Y)}{1} = I(X+1,Y) - I(X,Y)$$
$$I_y(X,Y) = \frac{\partial I(X,Y)}{\partial Y} \approx \frac{I(X,Y+1) - I(X,Y)}{1} = I(X,Y+1) - I(X,Y).$$

This approximation is linear and can be implemented using a simple *correlation* filter $[-1, 1]$ for I_x and $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ for I_y. Some examples of computing derivatives using these filters are in slides. However, it is worth mentioning that this is not the only numerical approximation of derivative and others exist:

$$\begin{array}{rcll} I_x(X,Y) & \approx & I(X+1,Y) - I(X,Y) & \text{forward difference} \\ I_x(X,Y) & \approx & I(X,Y) - I(X-1,Y) & \text{backward difference} \\ I_x(X,Y) & \approx & \dfrac{I(X+1,Y) - I(X-1,Y)}{2} & \text{central difference} \end{array}$$

the top two differ only by whether the $[-1, 1]$ is applied from left or the right of the image; while the last would correspond to the filter $[-1, 0, 1]$. Please note that *any* derivative filters will sum up to 0. The proof for this is in slides. Further, any of these filters can also be implement using convolution by flipping them 180 degrees. By convention, unless otherwise stated, we will use $[-1, 1]$ derivatives applied as correlation and with respect to the coordinate system that is at the top left corner of the image.

Consider the following row of pixel values and corresponding derivatives as an example

$$\left[\begin{array}{cccccccc} 0 & 0 & 50 & 100 & 200 & 50 & 0 & 0 \\ 0 & 50 & 50 & 100 & -150 & -50 & 0 & \times \\ \times & 0 & 50 & 50 & 100 & -150 & -50 & 0 \\ \times & 25 & 50 & 75 & -25 & -100 & -25 & \times \end{array}\right] \begin{array}{l} \\ \leftarrow \text{forward difference} \\ \leftarrow \text{backward difference} \\ \leftarrow \text{central difference} \end{array}$$

Image Gradient. It is convenient to think of the two derivatives as a tuple, a 2D vector, known as a *gradient* $= [I_x, I_y]$. It is also convenient to decompose this gradient in terms of its magnitude and direction.

Image Gradient Magnitude. Image gradient magnitude is simply defined as: $\sqrt{I_x^2 + I_y^2}$ and can be interpreted as the strength of the gradient or the edge (*i.e.*, the edginess measure of the pixel).

Image Gradient Direction. Gradient direction is defined as: $\text{atan}\left(\frac{I_y}{I_x}\right)$. The gradient direction always points toward greatest increase in brightness, *i.e.*, from dark to light. Note that gradient direction is defined with respect to the coordinate system in which the derivative is computed. Further note the relationship between the gradient direction and the edge; the gradient direction will always be perpendicular to the edge. This observation will become useful later when we discuss Canny edge detection.

Smoothing. Derivative filters are notoriously susceptible to noise. The noise in pixel values is amplified by a derivative operator. A second derivative, which is a direct extension of the first derivative above, would magnify the noise even more. Therefore smoothing is generally required before derivative operations. For convinence it is often customary to pre-convolve derivative filters with Gaussian smoothing to obtain a single filter to then apply to an image. For example, we typically apply x-derivative of a Gaussian and y-derivative of a Gaussian to obtain the derivatives I_x and I_y directly.

3 Edge Detection

Equipped with ability to compute image derivatives, we will now discuss three edge detection algorithms. In general, the properties we would like from edge detection are: *robustness/detection* – we want to reliably detect edges, *i.e.*, have low false positive and false negative rate for an edge; *localization* – we want to detect edges precisely at the pixel locations where there exists a boundary; and *single response* – we want the edges to be localized precisely to a pixel as opposed to having a number of responses near the edge.

Sobel. The simplest edge detection approach comes from realization that gradient magnitude in itself is a good measure of "edginess of a pixel". With this in mind a Sobel edge detector simply thresholds a gradient magnitude image partition the pixels into *edge* and *no edge* groups. The only slight caveat is that instead of smoothing by 2D Gaussian, Sobel only smooths in one dimension orthogonal to the gradient. This improves localization. Overall Sobel is fairly accurate, but is not very robust because results will generally change as a function of the threshold chosen. In addition, it tends to produce wide ridges around the edge. The reason is that smooth gradient (increasing or decreasing greyscale values) that is above the threshold will tend to produce a series of edge pixels (often referred to as the "ridge") along the edge.

Marr/Hildreth. Marr/Hildreth edge detection, instead, tries to localize the edge better by finding the extrema by considering the zero-crossings (transitions from positive to negative response or vice versa) of the second derivative. This gives rise to a Laplacian of Gaussian (LoG) operator, where the Laplacian is the second image derivative and the pre-convolved Gaussian is necessary to deal with noise. Note, that values used for smoothing here would typically be larger than for Sobel, because of the fact that second derivative is more susceptible to noise than the first. Similar to Sobel, LoG is a pre-convovled operator, so no additional smoothing is needed. The benefit of Marr/Hildreth approach is that it requires no threshold, making it easier and more robust to use. However, because the amount of smoothing tends to be more excessive some of the edge information, particularly around corners could be lost.

Canny. Canny is the most robust and accurate edge detector. It similarly to Marr/Hildreth tries to find local extrema, but does so by analyzing the first derivative, instead of resulting to dealing with the second. The approach can be best understood as an algorithm with 4 steps.

1. Computing derivatives. Compute directional x- and y- derivatives of a Gaussian for an image.
2. Compute gradient magnitude and direction. Compute gradient magnitude and gradient direction for each pixel using equations in Section 2.
3. Non-maxima suppression. The goal here is thin out the "ridges" you would get (*e.g.*, in Sobel) to single pixels that are local maxima. This is done by comparing gradient magnitude of a pixel to those of its neighbors. The two neighbors that are considered in this process are those that lie in positive and negative gradient direction (*i.e.*, neighbors across or orthogonal to the edge). In cases where gradient direction does not directly point to the neighbors (*e.g.*, not 45 or 90 degrees), the values are interpolated for better accuracy. If a gradient magnitude of a pixel in question is higher than both (interpolated) neighbors, it is kept as a potential edge pixel. If it is not, it is no longer considered as a potential edge pixel. In effect, this step culls all non-local maxima pixel candidates.
4. Linking and thresholding. Once the edges are "thinned" we could just threshold the gradient magnitude values of remaining potential edge pixels to obtain edges. However, similar to Sobel, this may be sensitive to specific threshold chosen. Instead, Canny defines two thresholds, k_{low} and k_{high}. Any remaining potential edge pixels are filtered (labeled as not an edge) if its gradient magnitude is lower than k_{low}. Similarly, any remaining potential edge pixel who's gradient magnitude is above k_{high} is labeled as an edge pixel and is considered *strong* edge pixel. Any pixels that end up between those two thresholds are *weak* edge pixels and have to be resolved through linking, where it is assumed that if any such pixels align with an edge, then they are likely to be edge pixels as well. In practice this requires starting and labeling chains of edge pixels. We start a chain from each *strong* edge pixel and then look in the direction orthogonal to the gradient direction (along the edge) to see if any *weak* edge pixels can be encountered. If there are, those are included in the final edge output. Those *weak* pixels that are not able to be linked are disregarded.

Note that Steps 1 and 2 for Canny are the same as those for Sobel (though Sobel has no use for gradient direction). The "secret souse" of Canny is in Steps 3 and 4 that address limitations of Sobel. Step 3 ensures that edges are one-pixel width in general; while Step 4, through the use of double threshold and linking procedure, ensures that results are much less sensitive to exact values of those thresholds (to a limit).

4 Corner Detection

Edges are useful because they allow us to define features that are robust to lighting changes. However, if we think about potentially localizing and matching local features across images, matching edges is fundamentally difficult, because locally edges are not distinct. Corners and blobs are good features to consider when one is interested in locally distinct features.

Autocorrelation. One way to measure local distinctiveness is by computing a sum of squared differences (or correlating) a small image patch with its local neighborhood. Consider extracting a patch of size $M \times M$ and then shifting it ± 5 pixels to the right/left/top/bottom and correlating at each location. This will result in an array of 11×11 correlation values. If the patch is locally distinct then these correlation values should drop off sharply in all directions as you move away from a center pixel. For an edge you will generally see a ridge in the autocorrelation 11×11 map. For a flat region, values in autocorrelation map will all be close to 0 for SSD and 1 for correlation. Harris corner detection proposes an algorithm for identifying pixels that are locally distinct, *i.e.*, for which autocorrelation drops off sharply in each direction. This will generally be true for features that look like corners in the image.

Harris. The derivation of Harris from autocorrelation is beyond this class, but it is derived by doing a Taylor expansion on autocorrelation function with SSD distance defined above. In practice, Harris can be characterized as an algorithm with four distinct steps:

1. Compute derivatives. This is done similarly as in the edge detection case, resulting in I_x and I_y.
2. Compute covariance matrix. This is done by averaging products of the gradients in the small neighborhood $\mathcal{R}$ (e.g., 11×11 in the example above) around each pixel location (X, Y). Formally this is defined as follows:

$$\mathbf{C}_{X,Y} = \begin{bmatrix} \sum\limits_{(i,j)\in\mathcal{R}} w_{i,j} I_x(X+i, Y+j) I_x(X+i, X+j) & \sum\limits_{(i,j)\in\mathcal{R}} w_{i,j} I_x(X+i, Y+j) I_y(X+i, Y+j) \\ \sum\limits_{(i,j)\in\mathcal{R}} w_{i,j} I_x(X+i, Y+j) I_y(X+i, X+j) & \sum\limits_{(i,j)\in\mathcal{R}} w_{i,j} I_y(X+i, Y+j) I_y(X+i, Y+j) \end{bmatrix}$$

   For the examples in slides the weighting $w_{i,j}$ was assumes to be equal to 1. This was for convenience of exposition. In practice, $w_{i,j}$ is defined by a Gaussian which allows pixels closer to the center pixel to contribute more to the sums, emphasizing local structure. In practice, covariance matrix is is a fit to autocorrelation function discussed above.

3. Characterizing cornerness. Once we have a covariance matrix we need to characterize extent to which corner exists at this pixel. This can be done by analyzing the eigenvalues of this matrix. A locally distinct corner-like structure would correspond to two eigenvalues that are large in magnitude. Existence of only one eigenvalue (the second being zero or close to zero) would characterize an edge. Both eigenvalues being zero would characterize a flat patch. We can ascertain the cornerness of the pixel by defining a function of eigenvalues (λ_1, λ_2) that would be high when both of these are high and low otherwise. There are a number of choices for such functions, which lead to different corner detection algorithms. Common choices include: $\min(\lambda_1, \lambda_2)$ and $\lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$. The latter is what Harris originally proposed. The benefit of this function is that it can be computed without explicitly computing the eigenvalues, since $\lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(\mathbf{C}) - \kappa trace^2(\mathbf{C})$.
4. Thresholding. This step is trivial and amounts to simply thresholding the values computed in the previous step to obtain a set of locally distinct corners. For a $\kappa = 0.04$ (a typical parameter choice for Harris) a threshold of 0 works well, *i.e.*, corners will produce a positive value, while non-corner pixels will typically produce a negative value.

Properties. Corner detection has some nice properties. Mainly, it is rotationally invariant. While rotation of an image will cause a change in eigenvectors of the covariance matrix, it will not change the eigenvectors. Meaning that cornerness measure and detected corners will remain exactly the same. Harris is also invariant to shifts in brightness due to, for example, certain changes in illumination. The reason is that shift in brightness will not change the gradients and hence eigenvalues, cornerness and detected corners. Changes in contrast (*i.e.*, multiplicative changes in brightness) will scale the gradients and will similarly scale the eigenvalues. As a result Harris will have some susceptibility to such changes.

5 Blob Detection

Blob detection stems from similar motivations to corner detection, but focuses on slightly different structure in images. Instead of detecting corners, blob detection focuses on detecting circular flat regions in the image. Such regions will also be locally distinct, however, can be characterized much more easily by simply looking at, and thresholding, a response of a Laplacian filter.

6 Characteristic Scale

Both corner and blob detection are *not* image scale independent. A pixel may look like a corner at a certain image scale and as an edge at another scale. Same is true for blobs. Hence it is often useful to figure out at which scale does the locally distinct feature actually exists. This can be done easily by running Harris or blob detection across multiple scales (each level of a Gaussian pyramid) and picking a response as one that is a maxima both spatially and across scales. This has similar motivation to non-maxima suppression in Canny edge detection. In practice, for each location and level in the Gaussian pyramid for which cornerness or Laplacian response is computed, we would compare it to it's 8 neighbors on the same level, as well as 9 neighbors on the higher and lower levels. In other words, we would compare it to 26 neighbors in total. We keep the response as a corner/blob detection only if it is higher than all the 26 neighbors in question and is above the specified threshold.