Computer Robot Vision: Lecture 3

Sam Barrett

March 4, 2021

1 Image Gradients & Edges

1.1 Edge Detection

The goal of edge detection is to take an image and map it from a 2D array of pixel values to a set of curves, line segments or contours. In so doing we filter out less relevant information whilst preserving important structural properties.

The main idea is to look for strong gradients in the pixel values. This allows us to loosly define an edge as a *place of rapid change in the image intensity function*. The first derivative of the intensity function over a horizontal section of pillar should show two spikes corresponding to each edge of the object.

We can express the partial derivative $f(x, y)$ for a 2D function as:

$$\frac{\delta f(x,y)}{\delta x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

For discrete data, such as images, we can approximate this using *finite differences*:

$$\frac{\delta f(x,y)}{\delta x} = \frac{f(x+1, y) - f(x,y)}{1}$$

We can also efficiently perform this operation using matrix convolution. We can think of an image gradient as being separate in the x and y direction, as such we must calculate the image gradient of both, $\frac{\delta f(x,y)}{\delta x}$ and $\frac{\delta f(x,y)}{\delta y}$

The filter for constructing the x derivative is simply:

-1	1

The filter for the y direction is: **check**

-1
1

The image gradient can be obtained as the partial derivative of the intensity values of an image in the x and y direction and can be denoted as:

$$\nabla f = \left[\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}\right]$$

With this value we can define:

- The **gradient direction**, or orientation of edge normal. It is given by:

$$\theta = \tan^{-1}(\frac{\delta f}{\delta y} / \frac{\delta f}{\delta x})$$

- The edge or gradient **strength** is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\delta f}{\delta x}\right)^2 + \left(\frac{\delta f}{\delta y}\right)^2}$$

1.1.1 Effects of noise

For any real-life image, if we consider a single row or column of the image and plot its intensity as a function we see a noisy signal. This noise effects our derivative as this noise is amplified by derivation.

Different filters have different sensitivity to noise. What can we do to tackle it?

One solution is to *smooth* our image first. One way of doing this is using the Gaussian kernel which can be seen in Figure 1

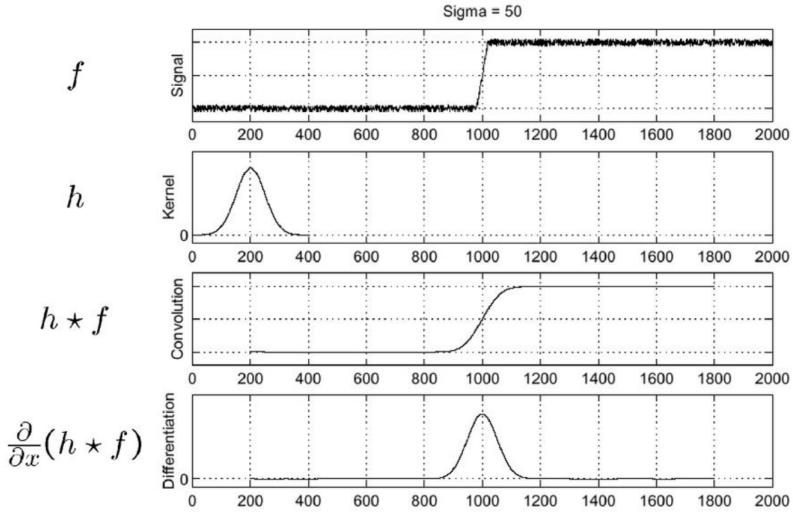

Figure 1: Effect of Gaussian smoothing on noisy function

In order to find edges in such an example, simply look for peaks in $\frac{\delta}{\delta x}(h \star f)$, where $h \star f$ is f convolved with the kernel h.

Convolution has many nice properties including being transitive and composable, allowing us to say that $\frac{\delta}{\delta x}(h \star f) \equiv (\frac{\delta}{\delta x}h) \star f$, this allows us to store the Gaussian derivative as a filter and perform one less step when finding the edges in a noisy function f.

There are other popular kernels including Prewitt, Sobel and Roberts.

Every smoothing mask has the following properties:

- Positive values
- all values sum to 1, as they take averages from surrounding pixels this makes sure the overall intensity of the image stays the same in areas of constant intensity.
- Amount of smoothing is proportional to the size of the mask
- They remove *high frequency* components, meaning the filter also acts as a *low-pass* filter.

We sometimes want to go further, we can take the second derivative by finding the derivative of our Gaussian derivative filter, this is known as the **Laplacian of Gaussian** filter $\nabla^2 h_\sigma = \frac{\delta^2}{\delta x^2}h$. When this filter is applied the edge is where the resulting graph crosses the x axis. These results are often more precise than the 1st derivative.

The size, σ of our kernel effects the sharpness and the structures found, choosing σ depends on the task at hand.

1.2 Laplacian Pyramid

This operation can be performed recursively and works by taking an image L_i and convolving it with our Gaussian filter t produce G_i, this resulting smoothed image is then up-scaled by injecting 0s in between each row and column before applying the Laplacian operator to produce an image L_i. As $i \to n$, the edges in L_i get thicker.

As i increases, only the thicker edges are present.

2 Edge detection and matching

We have seen the first step in edge detection in the previous section, smoothing. We will now look at the subsequent steps of **Edge enhancement** whereby we increase the contrast between edges and the background (via differentiation) and **Edge localisation** where we determine which local maxima are edges and which are noise.

2.1 Thresholding

The process of thresholding is as follows:

- Choose a threshold t
- set any pixels less than t to 0
- Set any pixels greater than t to 1

The standard thresholding procedure can also be defined formally as:

$$E(x,y) = \begin{cases} 1 & \text{if } \|\nabla f(x,y)\| > t \text{ for some threshold } t \\ 0 & \text{otherwise} \end{cases}$$

This process can only select *strong* edges and does not guarantee *continuity.*

2.2 Hysteresis Thresholding

An alternative method of thresholding is known as **Hysteresis thresholding**.
It uses 2 thresholds t_l, t_h, to represent a high and low boundary.

Note: usually $t_h = 2t_l$.

$\|\nabla f(x,y)\| \geq t_h$	definitely an edge
$t \geq \|\nabla f(x,y)\| < t_h$	maybe an edge, depends on the neighbouring pixel classifications
$\|\nabla f(x,y)\| < t_l$	definitely not an edge

Results of hysteresis thresholding often have better contour connectivity, i.e. more likely for entirely of edge to be preserved.

foreach *position* (x,y) **do**
  if *if* $\|\nabla f(x,y)\| < t_h$ **then**
    discard pixel (x,y)
  else
    keep pixel (x,y) as it is **strong**, and definitely an edge
  end
end
foreach *kept pixel* **do**
  foreach *connected pixel* (x,y) *(different notions of connectedness see Section 3.5)* **do**
    If $\|\nabla f(x,y)\| \geq t_l$ re-add the pixel, it is a part of an edge
  end
end

Algorithm 1: Hysteresis Thresholding

2.3 Non-maximum suppression

This is a thinning operator, and transforms wide *ridges* to a single pixel wide.

It does this by removing non-maximal pixels whilst preserving edge connectivity.

It does this by:

- Checking if a pixel is a local maxima along the gradient direction
- Selecting a single maximum across the width of the edge

  This sometimes requires interpolation if the gradient direction points towards the middle of 2 pixels.

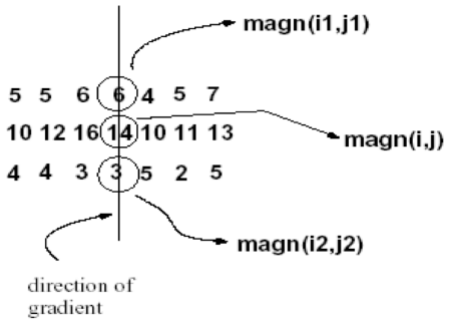

Figure 2: Non-maximum suppression

for *each pixel* (x, y) **do**
  if $magn(i,j) < magn(i_1,j_1)$ *or* $magn(i,j) < magn(i_2,j_2)$ **then**
    $I_N(i,j) = 0$
  else
    $I_N(i,j) = magn(i,j)$
  end
end

Algorithm 2: Non-Maximum Suppression

2.4 Designing an Edge Detector

We have the following criteria for an *optimal* edge detector:

1. **Good detection**

   An optimal detector will minimise the probability of false positives i.e. *"edges"* caused by noise. They will also minimise false negatives, the case in which true edges are missed

2. **Good localisation**

   Detected edges should be close to or at the same position as the true edges

3. **Specificity**

   A detector should return only a single point per true edge, i.e. edges should be sharp and not blurred over many pixels.

2.5 Canny Edge Detector

Canny came up with it, it isn't a particularly shrewd model.

This is the most popular edge detector in computer vision.

Canny showed that the first derivative of a Gaussian well approximates the operator that optimises a trade-off between signal-to-noise ratio and localisation.

The process of the Canny edge detector is as follows:

- Filter image using the derivative of Gaussian
- Calculate the gradient magnitude and direction
- Perform non-maximum suppression (See Section 2.3)
- Linking and thresholding using hysteresis thresholding

  We apply the high threshold to initialise contours, we then trace these contours until the magnitude falls below the low threshold.

2.5.1 Canny Characteristics

The Canny operator gives images with edges which are single pixel wide with good continuation between adjacent pixels. It is one of the most widely used edge detection operators and there are many sub-types. It is very sensitive to parameters, requiring extensive tuning for different domains.

3 Binary Image Analysis

Binary images are black and white, silhouette images.

Advantages:

- They are easy to acquire/take
- They require low amount of space to store
- They are (relatively) simple to process

Disadvantages:

- They have limited utility
- They cannot generally capture 3D scenes

- Specialised equipment is required to capture silhouettes.

The basic steps of binary image analysis are as follows:

1. Convert the image into a binary format

   This can be done by thresholding

2. Clean up the thresholded image

   This is performed using **morphological operators**

3. Extact separate *blobs*

4. Describe the blobs with region properties

3.1 Converting to binary form

We do this by thresholding, this is where we take a greyscale image and convert it to a binary mask.

We can use standard thresholding, hysteresis thresholding or a mask:

Where we have a set of values Z which define the intensity values which we are interested in:

$$E(x,y) = \begin{cases} 1 & \text{if } \|\nabla f(x,y)\| \in Z \\ 0 & \text{otherwise} \end{cases}$$

3.2 Morphological operators

These are operators used to change the shape of the foreground regions via intersection/ union operations between a scanning structuring element and a binary image.

This is a useful operation to *clean up* the results from thresholding.

The basic operations used are **Dilation** and **Erosion**.

3.2.1 Dilation

This operation expands connected components, grows features and fills in holes in the image. An example of what this operation does can be seen in Figure 3

At each position, if the current pixel is *on* i.e. is 1, then set all the output pixels corresponding to structuring element to 1.

This operation can be seen in a stream of binary pixels in Figure 4. You can see that the *objects* have become larger as gaps have been filled in. This procedure can scale to 2D images through the use of 2D mask.

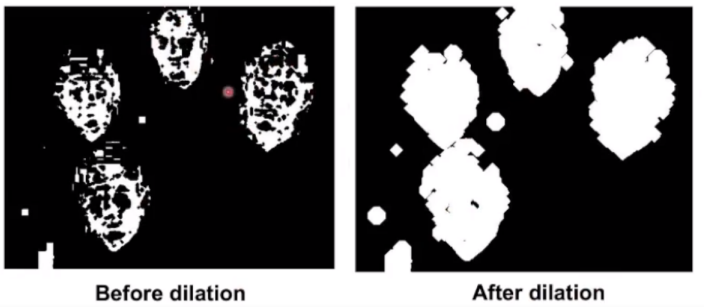

Figure 3: Before and after Dilation operation

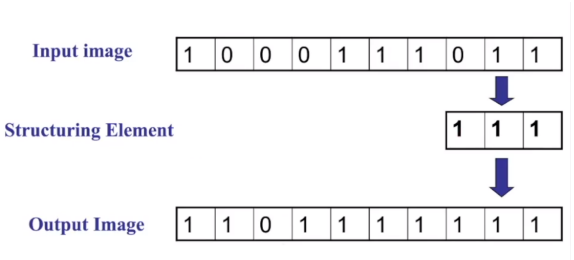

Figure 4: Dilation on a stream of binary pixels

3.2.2 Erosion

This operation *erodes* connected components, shrinks features and removes *bridges, branches* and noise.

An example of what it does to the result of our Dilation operation can be seen in Figure 5

To perform the erosion operation:

If every pixel under the structuring element is 1, then set the output pixel corresponding to the current pixel to 1, see Figure 6. Notice that the *object* get smaller.

3.3 Opening

Opening is the process of performing erosion and then dilation to a binary image. This process removes small objects but keep the original shape.

3.4 Closing

Closing is the process of performing dilation followed by erosion to a binary image. This fills in holes and keeps the original shape.

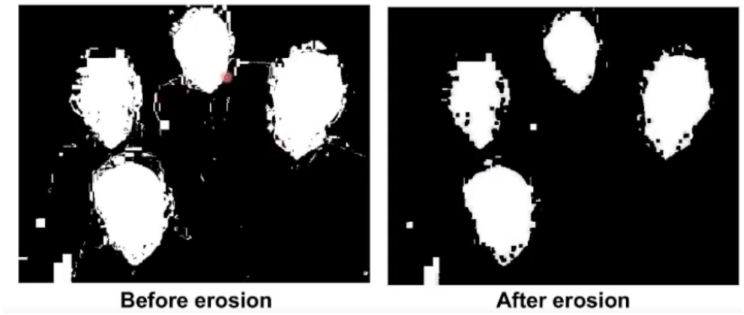

Figure 5: Erosion Example

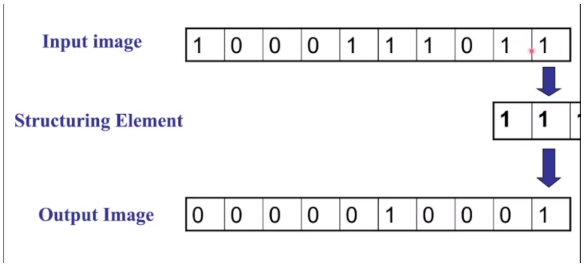

Figure 6: Erosion applied to binary pixel stream

3.5 Connectedness

How do we determine what pixels are *connected* to eachother when attempting to filter out noise without removing object detail?

Connectedness is the process of defining which pixels are considered neighbours.

3.5.1 4-Connected

This is the notion that a pixel is connected to others if it shares *northerly, easterly, southerly* or *westerly* border with them.

3.5.2 8-Connected

This is an extension of 4-Connectedness, here a pixel is deemed connected to all 8 of the pixels surrounding it, adding the diagonally connected pixels to the 4-Connected notion.

3.6 Region Properties

Given connected components, we can compute many simple *features* per blob including:

- Area, defined as the number of pixels in the region
- The centroid, the average x and y position of pixels in the region
- The bounding box, the minimum and maximum coordinates in both directions.