Segmenting with ITK (I)
Some notes

February 15, 2020

Contents

1. What is this session about?

In the first part of this session, we go on with filtering in ITK. Firstly, we will begin with *gradient calculation* because it is a common operation over images, especially as a first step in *edge detection* algorithms. Next, we will practice *thresholding* filters that are a rough method for image segmentation, and finally we will see filters for *edge detection*[1].

1.1. Learning outcomes

- To understand gradient computation and how it influences more complex processing algorithms.
- To understand thresholding technique an how to use it for basic image segmentation.
- To understand how edge detection works and use this technique with some example images.

2. Gradient computation

In ITK the term "gradient" may refer in some contexts to the gradient vectors and in others to the magnitude of the gradient vectors, hence ITK filters include the magnitude term when appropriate. Gradient vector gives us a clue about where a directional change in the intensity field of an image is produced.

Mathematically, the gradient of a 2D/3D function at each point is a 2D/3D vector with components given by the partial derivatives in each direction indicated by the basis vectors. At each point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction.

Since the intensity function of a digital image is only known at discrete points[2], derivatives of this function cannot be defined unless we assume that there is an underlying continuous intensity function which has been sampled at the image points. Approximations of these derivative functions can be defined at varying degrees of accuracy. The most common way to approximate the image gradient is to convolve an image with a kernel, such as the *Sobel operator*. Remember from the previous session that a kernel (mask) is what we use in images spatial (kernel) filtering for calculating an output image from an input image by using the mask over each and every pixel of the input image.

Each pixel of a *gradient image* measures the change in intensity of that same point in the original image, in a given direction. To get the full range of directions, gradient images in the x and y directions are computed.

2.1. Gradient magnitude

The magnitude of the image gradient is extensively used in image processing, mainly to help in the determination of region contours and the separation of homogeneous regions. The `GradientMagnitudeImageFilter` computes the magnitude of the image gradient at each pixel location using a simple *finite differences* approach[3]. This filter does not apply any smoothing to the image before computing the gradients. The results can therefore be very sensitive to noise. For solving this problem we can apply a noise reduction filter as a previous step in the pipeline or use the following filter.

[1] For further reading you can see the chapter on 'Filtering' in the ITK *guide2* [1].

[2] Remember that this discrete points are represented by *pixels* (squares) or *grid points* (centers of pixels).

[3] See Section 'Gradients' in the ITK *guide2*.

2.2. Gradient Magnitude With Smoothing

The `GradientMagnitudeRecursiveGaussianImageFilter` computes the magnitude of the image gradient at each pixel location. The computational process is equivalent to first smoothing the image by convolving it with a Gaussian kernel and then applying a differential operator. The user selects the value of σ.

Internally this is done by applying an IIR filter that approximates a convolution with the derivative of the Gaussian kernel. Traditional convolution will produce a more accurate result, but the IIR approach is much faster, especially using large σ values.

Attention should be paid to the image type chosen to represent the output image since the dynamic range of the gradient magnitude image is usually smaller than the dynamic range of the input image.

Task 1

Write a program that reads an image from a file and displays the result of applying the previous filters for calculating gradient magnitude to the input image. Try this program with the input images: `BrainProtonDensitySlice256x256.png` , `saltAndPepperNoise1.jpg` , `saltAndPepperNoise2.jpg` , and `gaussianNoise.jpg` . You may read the source codes `GradientMagnitudeImageFilter.cxx` and `GradientMagnitudeRecursiveGaussianImageFilter.cxx` placed at ' `ITK_DIR/Examples/Filtering` ' to get a clue about how to program this task.

3. Thresholding

The thresholding operation is used to identify pixel values based on specifying one or more values (called the **threshold value**). This is a very simple way of segmenting an image. Both sections in the *guide2*, 'Binary Thresholding' and 'General Thresholding', describe how to perform thresholding operations using ITK. Try the examples pointed out over an input image.

Binary thresholding, `BinaryThresholdImageFilter` , tranforms an image into a binary image by transforming each pixel according to whether it is inside or outside a specified range. The user chooses lower and upper threshold values to process. If a pixel is inside of this range, it is assigned an 'inside' value. Otherwise it is assigned an 'outside' value. This filter gets as input an image and produces as output another image. Note that the lower and upper thresholds are values of the type of the input image pixels, while the inside and outside values are of the type of the output image pixels. Formally:

$$\text{Output}(x_i) = \begin{cases} InsideValue & \text{if} \quad lowerThreshold \leq x_i \leq upperThreshold \\ OutsideValue & \text{otherwise} \end{cases}$$

The `ThresholdImageFilter` can be used to transform the intensity levels of an image in three different ways.

- The user can define a single threshold and all the pixels with values below this threshold will be replaced by an 'outside' value. The corresponding method is `ThresholdBelow()` .
- The user can define a single threshold and all the pixels with values above this threshold will be replaced by an 'outside' value. The corresponding method is `ThresholdAbove()` .
- The user can define two thresholds and all the pixels with values outside the range between these two thresholds will be replaced by an 'outside' value. The corresponding method is `ThresholdOutside()` .

Task 2

Write a program that reads an image from a file and displays the result of applying the previous thresholding filters to the input image. The idea is to obtain a segmentation of some region of interest that you can identify in the image. Try this program with the input images: `BrainProtonDensitySlice256x256.png`, `saltAndPepperNoise1.jpg`, `saltAndPepperNoise2.jpg`, and `gaussianNoise.jpg`. You may read the source codes `BinaryThresholdImageFilter.cxx` and `ThresholdImageFilter.cxx` placed at '`ITK_DIR/Examples/Filtering`' to get a clue about how to program this task.

4. Edge Detection

As it was described in a previous lecture, a common operation in image processing is *edge detection*. We are going to practice the Cany Edge Detection filter provided by ITK toolkit[4]. Canny edge detection is widely used for edge detection since it is the optimal solution satisfying the constraints of good sensitivity, localization and noise robustness. The filter implementation requires a sequence of steps which involves Gaussian smoothing to remove noise, calculation of gradient magnitudes to locate edge features, non-maximum suppression to remove spurious features, and finally thresholding to yield a binary image.

Try the example code `CannyEdgeDetectionImageFilter.cxx`. In this example, three parameters of the Canny edge detection filter may be set via the `SetVariance()`, `SetUpperThreshold()`, and `SetLowerThreshold()` methods. Taking into account the steps used by this filter, we can infer that the `variance` is used for Gaussian smoothing and `upperThreshold` and `lowerThreshold` control which edges are selected in the final step.

Task 3

Write a program that reads an image from a file and displays the result of applying the `CannyEdgeDetectionImageFilter` to the input image. The idea is to obtain several results depending of the free parameters of the filter, together with using a previous noise reduction over an input image or providing the filter with the input image without any previous noise reduction step. Try this program with the input images: `BrainProtonDensitySlice256x256.png`, `saltAndPepperNoise1.jpg`, `saltAndPepperNoise2.jpg`, and `gaussianNoise.jpg`. You may read the source code `CannyEdgeDetectionImageFilter.cxx` placed at '`ITK_DIR/Examples/Filtering`' to get a clue about how to program this task.

Exercise

Write a report on the different results you have obtained concerning the different types of images you have provided to your programs for thresholding and Canny edge detection segmentation, together with the different parameter values you have used. Please, illustrate the results showing the images you have obtained in each pipeline, detailing the parameters you have used.

[4] You can read Section 'Edge Detection' in the ITK *guide2*.

References

[1] Johnson, H. J., McCormick, M. M., and Ibanez, L. *The ITK Software Guide Book 2: Design and Functionality-Volume 2 4th ed.*, 2017.