Live-Wire-ing the Insight Toolkit with Intelligent Scissors

Release 0.00

Nicholas J. Tustison, Paul A. Yushkevich and James C. Gee

May 21, 2008

Penn Image Computing and Science Laboratory
University of Pennsylvania
Philadelphia, Pennsylvania, USA

Abstract

Semi-automatic image segmentation algorithms depend on interaction with the user to accurately define a region of interest within an image. Once such method is a dynamic programming approach called *Intelligent Scissors* developed by Mortenson and Barret [5, 4, 1, 2]. Standard interaction involves the user-placement of a seed point on or near the boundary of the object to be extracted. Using a gradient-based cost function, a *live-wire* image path from the seed point to a subsequently placed boundary point is determined. As this free point is manipulated with the mouse cursor, the live-wire boundary, which extends from the seed point to the varying free point, locks onto nearby edges within the image. Although some applications do not require such user-interaction (e.g. [3]) warranting inclusion within the toolkit, incorporating live-wire capabilities into ITK-SNAP [6] has been discussed and, therefore, we would like to, as a preliminary step, vet the code within the ITK community.

Contents

1 Intelligent Scissors: Theory

Mortensen formulated the boundary finding problem as a single-source shortest path problem on a weighted graph [5]. Each pixel in the image constitutes a node in Mortensen's graph and the edges of the graph are the links between the pixel neighbors. The minimum weight path from a source pixel to a target pixel is the path which minimizes a specified cost function determined by Dijkstra's algorithm. This cost function is a

combination of three components: the gradient of an image (f_g), the zero-crossings of an image (f_z), and a boundary smoothness constraint associated with the gradient direction at each pixel (f_s). The directional edge between any two neighboring pixels has an associated cost based on a weighted combination of each of these quantities. Given two pixel neighbors p and q, the cost of the edge between the corresponding two nodes in the graph is

$$C_{\vec{pq}} = w_g f_g(q) + w_z f_z(q) + w_s f_s(p,q) \tag{1}$$

where w represents the weight of the respective component.

The gradient of an image provides a measurement of local edge strength. If I represents the image, the gradient G of a 2-D image is calculated to be

$$G = \sqrt{I_x^2 + I_y^2 + I_z^2}. \tag{2}$$

Since the goal is to associate large edge strength values with low cost and to maintain gradient values in the range $[0,1]$, we invert and scaled the gradient image such that

$$f_g = 1 - \frac{G}{\max G} \tag{3}$$

The second component of the cost function is the Laplacian zero crossing feature, f_z, which is a binary edge-feature image derived from I. It is calculated by convolving I with a Laplacian edge operator. The binary edge features are calculated by looking for sign changes in the convolved image indicating an edge so

$$f_z(q) = \begin{cases} 0 & \text{if } q \text{ is an edge pixel} \\ 1 & \text{if } q \text{ is not an edge pixel} \end{cases} \tag{4}$$

The final component of the cost function, f_s, promotes a smooth boundary by giving a high cost to edges which deviate sharply from the boundary direction. This is calculated from

$$f_s(p,q) = 1 - \frac{1}{\pi}\left[\arccos(d_{\min}(p)) + \arccos(d_{\min}(q))\right] \tag{5}$$

where

$$d_{\min}(p) = \min\left(\frac{\nabla I(p)}{||\nabla I(p)||} \cdot \frac{\vec{pq}}{||\vec{pq}||}, \frac{\nabla I(p)}{||\nabla I(p)||} \cdot \frac{\vec{qp}}{||\vec{qp}||}\right) \tag{6}$$

and $d_{\min}(q)$ is calculated similarly. Readers will notice that this formulation is slightly different from that prescribed in the original live-wire formulation. This is due to the fact that the original formulation was limited to 2-D images and we wanted to allow the possibility of finding the minimum path through n-D images which is accommodated by our modification.

2 Intelligent Scissors: Implementation

We realize that there are many implementations and variations of the original formulation. However, instead of pursuing the difficult process of determining which implementation provides optimal performance over all others, we decided to implement the original incarnation proposed by Mortensen and Barret [5, 4, 1, 2] while making minimal changes. The two changes from the original algorithm are as follows:

- we allow for finding the minimal path in n-D images and
- we use a priority queue (instead of a bucket heap) in the implementation of Dijkstra's algorithm.

This allows other users to take the bare-bones implementation and incorporate whatever additional features they choose.

Typical interaction is an input image is specified along with an anchor point presumably located on an image. The livewire path can then be determined from any pixel within the region of interest. Therefore, our implementation inherits from the `itk::ImageFunction` class which provides querying functionality (e.g. `EvaluateAtPoint()` and `EvaluateAtIndex()`). In our case, what is returned is a smart pointer to a `itk::PolyLineParametricPath`.

The various user-changeable parameters which are as follows:

- `GradientMagnitudeWeight`: see w_g in Equation (1). Default value is 0.43.
- `GradientDirectionWeight`: see w_s in Equation (1). Default value is 0.14.
- `ZeroCrossingWeight`: see w_z in Equation (1). Default value is 0.43.
- `ZeroCrossingImage`: see f_z in Equation (1). Defaults to the output of the filter `itk::ZeroCrossingBasedEdgeDetectionImageFilter`.
- `UseImageSpacing`: determines the scaling factor for the neighborhood weighting. Default value is `true`.
- `UseFaceConnectedness`: determines the local neighborhood. Default value is `true`.
- `MaskImage`: if specified it determines the region of interest. Default value is `NULL`.
- `InsidePixelValue`: determines the label value which designates the region of interest in the `MaskImage`.

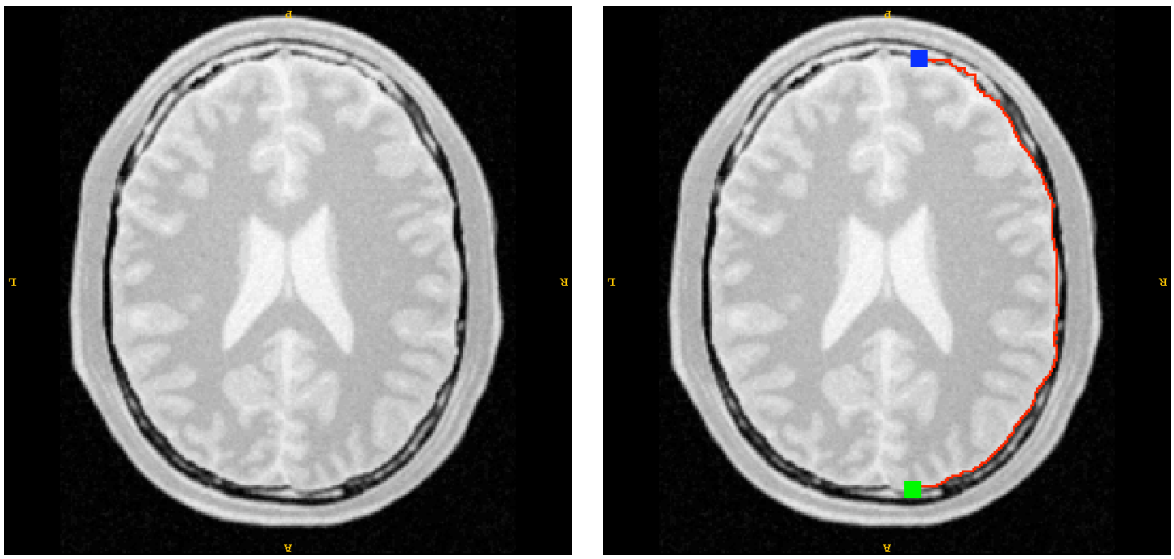
Setting up and using the class is relatively simple. Setting up and instantiation of the class `itk::LiveWireImageFunction` is performed in the usual way: where the `labelImage` is used to hold the resulting live-wire paths to be displayed in ITK-SNAP.

The following code snippet illustrates the setting of the anchor seed and subsequent extraction of the live-wire path evaluated from the specified image index.

3 Sample Results

The first sample results are taken from a brain slice which is available in the ITK Data directory and are illustrated in Figure 1. The output image is obtained with the function call

```
>> itkLiveWireImageFunctionTest BrainProtonDensitySlice256x256.nii
  outputBrainProtonDensitySlice256x256.nii 67 52 46 127
```



(a) (b)

Figure 1: Sample results from a brain slice image available in the `Data` directory. (a) Input brain slice. (b) The green box denotes the anchor seed and the red image path represents the output value of evaluating the live-wire function at the blue square.

where the anchor seed indices are $(67,52)$ and the path returned is the evaluation performed at index $(46,127)$. The results can be visualized in snap where the label path image is superimposed over the original image.

The second sample results are taken from a binary image (also available in the ITK Data directory). The results are illustrated in Figure 2. The output image is obtained with the function call

```
>> itkLiveWireImageFunctionTest BinaryImage.nii outputBinaryImage.nii 24 113 161 39
  maskBinaryImage.nii
```

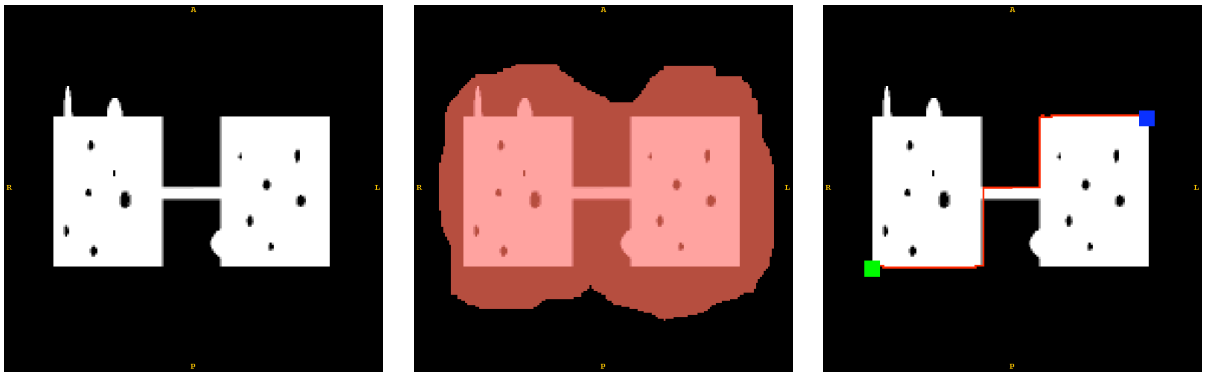
where the anchor seed indices are $(24,113)$ and the path returned is the evaluation performed at index $(161,39)$. For this example, a mask image is also used. The results can be visualized in snap where the label path image is superimposed over the original image.

References

[1] W. A. Barrett and E. N. Mortensen. Fast, accurate and reproducible live-wire boundary extraction. In *Visualization in Biomedical Computing*, pages 183–192, 1996. (document), 2

[2] W. A. Barrett and E. N. Mortensen. Interactive live-wire boundary extraction. *Medical Image Analysis*, 1(4):331–341, September 1997. (document), 2

[3] Shiying Hu, Eric A. Hoffman, and Joseph M. Reinhardt. Automatic lung segmentation for accurate quantitation of volumetric x-ray ct images. *IEEE Transactions on Medical Imaging*, 20(6):490–498, 2001. (document)



(a) (b) (c)

Figure 2: Sample results from a brain slice image available in the `Data` directory. (a) Input binary image. (b) The region of interest specified via a mask image. (c) The green box denotes the anchor seed and the red image path represents the output value of evaluating the live-wire function at the blue square.

[4] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *PRoc. ACM SIGGRAPH 95: Computer Graphics and Interactive Techniques*, pages 191–198, August 1995. (document), 2

[5] E. N. Mortensen, B. S. Morse, and W. A. Barrett. Adaptive boundary detection using 'live-wire' two-dimensional dynamic programming. In *IEEE Proc. Computers in Cardiology*, pages 635–638, 1992. (document), 1, 2

[6] Paul A Yushkevich, Joseph Piven, Heather Cody Hazlett, Rachel Gimpel Smith, Sean Ho, James C Gee, and Guido Gerig. User-guided 3d active contour segmentation of anatomical structures: significantly improved efficiency and reliability. *Neuroimage*, 31(3):1116–1128, Jul 2006. (document)