A Matlab-based Implementation of the Livewire Interactive Segmentation Tool

Omer Ishaq

Centre for Image Analysis, Uppsala University, Uppsala, Sweden
omer@cb.uu.se

Abstract. Image segmentation is the process of delineating an image into foreground and background regions. Typically, the computer-assisted image segmentation techniques are classified as either fully automated or interactive segmentation methods. These interactive segmentation techniques are typically graph-driven and are further sub-categorized as either region- or boundary-based algorithms. Livewire is one of the earliest interactive image segmentation algorithm and has been extensively applied to image segmentation problems. As part of a course on Graph-based Methods for Image Analysis, an implementation of the Livewire was done in Matlab. This implementation is the focus of this report. Implementation details and results are provided. Note, there is no novel contribution in the report.

1 Introduction

In medical image analysis (MIA) segmentation is vitally important for localizing, delineating, quantifying and visualizing anatomical structures. Typically, the segmentation approaches in MIA can be broadly categorized as: (i) Manual segmentation; (ii) Fully automated segmentation; (iii) Interactive segmentation. Traditionally, segmentation has been performed through manual tracing by experts, however, such an approach is very time consuming and engenders large inter- and intra-rater variation. The other end of the spectrum comprises fully automated segmentation techniques. Although these techniques are fast and reproducible, yet they suffer from a significant reduction in segmentation accuracy. Where as, interactive (semi-automated) segmentation techniques have emerged as a robust mechanism for performing accurate computer-aided image segmentation using minimal human intervention.

The project focuses on a Matlab implementation of the classic Livewire interactive segmentation tool [1] for two dimensional images.

2 Methods

This methods section is organized as follows, Section 2.1 details the generation of a weighted-graph representation of a 2D image. Sections 2.2 and 2.3 discuss the shortest path selection in a weighted graph and the generation of a closed curve, respectively. The details of the matlab implementation are provided in the Section 2.4.

2.1 Generation of the Cost Graph

Livewire is a graph-based method for interactive image segmentation. This necessitates the generation of an undirected weighted-graph W from an input image. Any n-dimensional image can be represented as an undirected graph. The graph nodes represent the image pixels. The edges represent the connections/relationships between the neighboring pixels. We use a 4-connectivity model where every pixels is connected to four of its closest neighboring pixels, that is, the top, bottom, left and right pixels. Subsequently, we assign the weights to the graph edges. The weights assigned are a function of the edge strength between two pixels. More specifically, for each pixel $p_{i,j}$ image derivatives/gradients G_x and G_y are calculated along the horizontal and the vertical directions, such that,

$$G_x = p_{i,j} - p_{i+1,j} \tag{1}$$

$$G_y = p_{i,j} - p_{i,j+1} \tag{2}$$

The magnitude G and direction θ of the gradient are calculated as,

$$G = \sqrt[2]{(G_x)^2 + (G_y)^2} \tag{3}$$

$$\theta = tan^{-1}\frac{G_y}{G_x} \tag{4}$$

The orientation θ calculated from the Equation 4 is normal to the image edge. Since, the objective of a Livewire algorithm is to *follow* along an image edge/boundary, therefore, the recovered directional vector θ is rotated through ninety degrees to align with the edge. We will refer to the re-aligned vector as θ^a. For the current project, we followed the convention of rotating the vector θ clockwise. An anti-clockwise rotation yields the same results. The re-aligned vector θ^a may have any orientation between 0 degrees and 360 degrees with reference to the horizontal. On the other hand, the edges connecting a pixel $p_{i,j}$ with its neighbors occur only along the orientations of 0, 90, 180 and 270 degrees, therefore, the re-oriented gradient vector with the orientation θ^a and magnitude G is decomposed to its horizontal and vertical components G_x^a and G_y^a. These components are shown in the Equations 5 and 6,

$$G_x^a = G\cos\theta^a \tag{5}$$

$$G_y^a = G\sin\theta^a \tag{6}$$

The inverse of these components G_x^a and G_y^a are assigned as the edge weights W_x and W_y in the cost graph W,

$$W_x = \frac{1}{G_x^a} \tag{7}$$

$$W_y = \frac{1}{G_y^a} \tag{8}$$

The inverse is calculated so that the strongest edges in the image have lowest cost in W, and vice versa. The process is repeated for all the pixels in an image.

2.2 Selection of the Shortest Path

Given an undirected weight graph W, generated from Section 2.1, the goal of Livewire is to find the strongest edge between two user specified control points C_k and C_{k+1}. As mentioned in Section 2.1, the stronger the edge strength in the image I, the lower its weight in W, therefore, the task of finding the strongest edge between points C_k and C_{k+1} is the same as finding the lowest cost path (i.e., the shortest path) between the nodes representing the fore-mentioned control points in the graph W. We use the $Dijkstra's$ algorithm for finding this path.

Therefore, given an ordered set of control points C_1, C_2, C_3, ... C_k, ... C_n, a connected curve can be generated by finding the shortest path between all pairs of neighboring control points, that is, a set of n points would generate a set of $n-1$ paths. These paths are connected together to generate the resultant curve. The generated curve may or may not be closed. A mechanism for generating a closed curve is dicussed in the Section 2.3.

2.3 Generation of a Closed Curve

The goal of an image segmentation algorithm is to generate a *closed* delineating curve which separates the foreground and the background pixels. The curve generated from the Section 2.2 may or may not be closed. There are three possible scenarios,

1. The curve γ defined over the interval $[a, b]$ self intersects at only one point, such that $\gamma(a) = \gamma(b)$. This is a closed curve with no other self intersections and provides a delineation between the fore- and the back-ground.
2. The curve γ has no self intersection points. This is an open curve and can be closed by finding the shortest path (i.e., using the Dijkstra's algorithm) between the extreme points a and b.
3. The curve γ self intersects at point c, where c is different from a and b . The curve is corrected by discarding the curves over the intervals $[a, c)$ and $(c, b]$.

2.4 Implementation Details

The algorithm is implemented in the Matlab package. The weighted connectivity graph W is represented by a sparse adjacency matrix. The sparse matrix is selected because of the memory constraints. All the weights calculated in the Section 2.1 are assigned to the sparse matrix in a single step, on the other hand, if the weights were assigned one at a time, it would result in considerable computation overhead since the matrix would need to grow with each assignment.

The shortest path between any pair of user specified control points is generated by the *graphshortestpath* function which is based on the Dijkstra's algorithm. The interactive user interface is implemented by capturing the mouse and the keyboard events.

3 Results

The implemented algorithm was used on multiple images. The results for one of these images is shown in Figure 1. Figure 1(a) shows a minimum cost curve/path (shown in yellow), between a static user-specified control point and the moving mouse pointer. More specifically, the user clicks on the image to specify a static control point and then moves his/her mouse pointer to generate potential boundary delineations. An appropriate boundary delineation can be selected/fixed by clicking a second time on the image. Subsequently, the selected curve segment is saved, as shown in blue color in the Figure 1(b). In this manner, $n-1$ curves can be selected by specifying n control points. An open curve (shown in Figure 1(b)) can be auto-closed (Figure 1(c)) by using the method in Section 2.3.

As discussed in the Section 2.4, the edge weights are assigned to the sparse adjacency matrix in a single step, therefore, the memory for the adjacency matrix is allocated only once. The weights calculation and assignment step takes 8.9 seconds for a 2D image with 225 pixels along each dimension. However, if the assignment is done individually for each weight, then it can take as much as 113 seconds for the same image. Consequently, we assign all the weights in one step.

(a) Shortest path between a control point and the moving mouse pointer.

(b) A selected/fixed curve segment.

(c) An auto-closed curve.

Fig. 1. Different execution stages of the algorithm: (a) Generation of a potential curve; (b) A fixed/selected curve; (c) An auto-closed curve.

4 Conclusions

We have implemented a Matlab-based Livewire algorithm. The algorithm has reasonable computation time. An interactive graphical user interface is provided.

References

1. A.X. Falco, J.K. Udupa, S. Samarasekera, S. Sharma, B.E. Hirsch, R.A. Lotufo, "User-steered image segmentation paradigms: Live-wire and live-lane", Graphical Models and Image Processing, 60(4):233-260, 1998.