Homework #3: Contour-Detection

Due Tuesday, March 10th, 2007.

Professor Davi Geiger

Consider the images

Fig 1 (a), (b), (c), and (d) are test images

The goal is to extract contours from the images as described in class and show the results as images, with contours in red overlaid over the gray level images. The best way to display is to first assign the grey value to all three color channels, red, green and blue. Then assign the value 255 to the red color channel wherever there is a contour pixel, but still let the other color channels to be assigned the corresponding image grey value.

Consider the energy function $E(v_1,...,v_C) = \sum_{t=1}^{C} E_t(v_{\tau-t},v_\tau) = \frac{1}{C} \sum_{t=1}^{C} f_s(v_\tau) + e_{\tau,\lambda}(v_{\tau-t},v_\tau)$ where $e_{\tau,\lambda}(v_\tau,v_{\tau+1}) = \gamma |\theta_\tau - \theta_{\tau-1}| = \gamma |\theta_\tau - \theta_{\tau+1}|$ and $f_s(p_\tau,\theta_\tau) = T(s) \|D \hat{I}(p_\tau,\theta_\tau, s)\| + 1$. Note that in HW2, you displayed the image of $I(p_\tau) = \arg\min_{\theta_\tau} f_5(p_\tau,\theta_\tau)$ but in this homework you will consider, for each pixel, all the 8 angles values of $f_5(p_\tau,\theta_\tau)$ and let dynamic programming selects the best angle at each stage. We have neglected the parameter $\lambda$ in $f_5(p_\tau,\theta_\tau)$ as it is a constant value and will not have any consequence to the optimization results.

Note that we will eight different angles $\theta_\tau = -\pi/4, -3\pi/4, 0, \pi/4, 3\pi/4, \pi, 5\pi/4, 7\pi/4$ but we have for each pixel $p_\tau$ only four derivatives since the magnitude of the derivatives are the same for $\theta_\tau$ and for $\theta_\tau + \pi$. Thus, when writing the program, make sure to consider $\|D \hat{I}(p_\tau,\theta_\tau + \pi, s)\| = \|D \hat{I}(p_\tau,\theta_\tau, s)\|$. Also the function $|\theta_\tau - \theta_{\tau+1}|$ should be replaced by $\min(|\theta_\tau - \theta_{\tau+1}|, 2\pi - |\theta_\tau - \theta_{\tau+1}|)$ to account for the circular nature of angles. We may call $|\theta_\tau - \theta_{\tau+1}|_{\text{circular}} = \min(|\theta_\tau - \theta_{\tau+1}|, 2\pi - |\theta_\tau - \theta_{\tau+1}|)$

The parameter $\gamma$ will have to be estimated. To that end, the first step is to understand the range of values of $f_5(p_\tau,\theta_\tau)$ and assume that the term $\gamma |\theta_\tau - \theta_{\tau+1}|_{\text{circular}} \leq \gamma \pi$ should be just a bias to
differentiate similar contours, i.e., it should not dominate the data term $f_3(p, \theta)$. Trial and error is the best way to fine tune the estimation.

**QUESTION: CONTOUR FOLLOWER VIA DYNAMIC PROGRAMMING**

Write a dynamic program where given an image, the user chooses a starting pixel and after the program runs, it requests the user to select the end pixel. Once the user selects the end pixel, the program returns the optimal contour between the initial and end pixel in red color overlaid over the black and white image.

Estimating good values for the parameter $\gamma$ is part of the homework, i.e., please report the values used. They may not be the same for the synthetic black and white images (circle/trapezoid) than for the real images. Consider using different values of $\gamma$ for these two different type of images.

Details: In dynamic programming we will have 8 starting nodes corresponding to the user selected initial pixel and its 8 possible directions. The user interface can input the pixel either by typing it or by clicking on the image, however you choose to implement it. Set $C$ to be a high number, say $W$ where $W=\text{Image Width}$ (number of pixels in the image along the width). Also, after the input pixel, the program run dynamic programming and at the end should prompt/allow the user interface to select the “end” pixel in the image. Again, the user can input this final pixel just like it did for the initial one, either by typing its coordinates or using the mouse, however you find best. Then the program will output the best contour from the starting point to the “end” selected final point.

**NOTE:** Make sure that the dynamic programming runs before the user selects the end point. By the time the user selects the end point dynamic programming will already have computed the solution for all contours to all points up to length $C=W$.

Here is a discussion and version of a pseudo-code for this problem

```plaintext
Initialization of the selected pixel $p^*$: For all angles, the nodes $u^*=(p^*, \theta)$ should be assigned $\Phi_{\tau=1^*}(u^*)=0$. Keep track of all nodes, i.e., for each $v$ store the minimum cost value $\Phi_{\tau^*}(v)$, the node $v$ as well as its corresponding length $\tau^*$. Once the program ends at $\tau=C$, and for each pixel “$p$”, select the optimal cost corresponding to the best node $v^*(p)$ and its optimal length $\tau^*(p)$ and optimal angle $\theta^*(p)$. Finally backtrack the optimal path starting at this optimal node $\tau^*$ back to $u^*_{\tau=1}$, again using $v^*_{\tau=1}=\text{back}_\tau(v^*_{\tau})$ recursively.

The criteria is the mean contour value, so we use the variable $\text{Optimal}(p)$ to store the average contour value for each pixel $p$ and its corresponding optimal length $\tau^*(p)$ and angle $\theta^*(p)$. More precisely

Contour-Detection DP( Image, C, p*, $\gamma$) /* $u^*$ are the set of eight nodes corresponding to initial pixel p* */
Initialize
Create the Graph $G(V,E)$ from the image
Create the Graph $\Phi(T=C, V, E)$ : effectively C copies of the graph $G(V,E)$

loop for $v$ in $V$
   $\Phi_{\tau=1^*}(v)=\infty; /*$ first column */
```

2
p = ExtractPixel(v); /* function that extracts pixel p from node v=(p,θ) */

Optimal (p)=∞; /* function carrying the optimal cost for pixel p of node v=(p,θ) */
end loop

loop for θ   /* 8 angle values */
Φτ=1 *(p*,θ)=0 /* initializing at all nodes u*: Φτ*[τ=1, u*]=0, where u*=(p*,θ) */
end loop

/* precomputation of the transition costs can be stored in an array of size (8, 8*N) for a 8 neighbor structure */

loop for v in V
  loop for u such that e(u,v)≠∞ /*u represents the 8 neighbors that can reach v */
    E(u, v) = f_v (u) + γ |θ_u - θ_v |circular ;
  end loop
end loop

Main loop
loop for τ=2, 3,..., C
  loop for v in V
    F = ∞;
    loop for u such that e(u,v)≠∞ /* consider only neighbors u that can reach v */
      if (Φτ-1 *(u) + E(u,v) < F); { 
        F= Φτ-1 *(u) + E(u,v);
        backτ(v) =u; }
    end loop
  end loop
  Φτ *(v)= F;
  p = ExtractPixel(v); /* extract pixel p from node v=(p,θ) */
  /* finding the best node, average cost, for pixel p */
  if ( 1/τ Φτ *(p,θ) < Optimal(p) ) { 
    Optimal(p)= 1/τ Φτ *(p,θ); /* getting the best average cost path */
    τ*(p) = τ; 
    θ*(p) = θ; }
end loop
end loop

In order to obtain the optimal contour for any selected end pixel p, find the corresponding optimal node v∗τ=(p,θ∗(p)) at length τ=τ*(p) and angle θ=θ*(p); and then backtrack using the recurrent formula v∗τ-1 = backτ (v∗τ).