Homework #3: Contour-Detection

Due Wednesday, February 28th, 2007.

Professor Davi Geiger

Consider the images (work with 8 neighbors, not 16, to make it simple)

![Images](image1.jpg)

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

The goal is to extract contours from the images as described in class, using only 8 neighbors, and show the results as images, with contours in red overlaid over the black and white images.

Consider the energy function $E(v_1, ..., v_C) = \sum_{t=1}^{C} E_t(v_{t-1}, v_t) = \sum_{t=1}^{C} \hat{f}_t(v_t) + e_{\gamma, \lambda}(v_{t-1}, v_t)$

Where $\hat{f}_t(v_t) = \hat{f}_5^+ (p_t, \theta_t)$ is the one from HW2. Note that in HW2, you displayed the image of $I(p_t) = \min_{\theta_t} f_5^+ (p_t, \theta_t)$ but in this homework you will consider, for each pixel, all the 8 angles values of $\hat{f}_5^+ (p_t, \theta_t)$ and let dynamic programming selects the best angle at each stage. The values of $\hat{f}_5^+ (p_t, \theta_t)$ are normalized as in HW2, so the values vary from 0 to 255. Also, $|\theta_t - \theta_{t-1}|$ is a periodic function and the exact definition for $e_{\gamma, \lambda}(v_{t-1}, v_t)$ is

$$e_{\gamma, \lambda}(v_{t-1}, v_t) = \gamma \min(|\theta_t - \theta_{t-1}|, 2\pi - |\theta_t - \theta_{t-1}|)$$

So, from the class material, $\mu = 0$ and $g(x) = x$ and $s=5$, and we have neglected the parameter $\lambda$ as it is a constant value and will not have any consequence to the optimization results.

The parameter $\gamma$ will have to be estimated.

**Question:** Contour Follower 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 final pixel. Once the user selects the final pixel, the program returns the optimal contour between the initial and final pixel in red color overlaid over the black and white image.
Estimating good values for the parameter $\gamma$ is part of the homework. Notice that the values of the first term are between 0 and 255 while the values of the second term are between 0 and $\pi$. We do want the second term to have less weight, but how much less? This is the role of $\gamma$. We don’t expect $\gamma > 40$, since we don’t want to penalize too much for $\pi/4$ angle changes along the contour. The cost would be $40 \times \pi/4 \sim 30$) and we don’t expect $\gamma < 10$ since we don’t want to allow for the 180 degree turns to be easy to appear, i.e., cost of $10 \times \pi \sim 30$). Good candidates to try are $\gamma = 15, 20$ and $30$.

Details: In dynamic program we will have 8 starting nodes corresponding to the user selected initial pixel and its 8 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=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 “final” 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 produce the best contour from the starting point to any selected final point.

NOTE: Make sure that the dynamic programming runs before the user selects the final point. By the time the user selects the final 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

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 $“p_v”$, select the optimal cost corresponding to the best node $v=(p_v,\theta^*)$ and its optimal length $\tau^*$ and optimal angle $\theta^*$. Finally backtrack the optimal path starting at this optimal node $v^*_\tau$ back to $u^*_1$, again using $v^*_{\tau-1}=back_\tau(v^*_\tau)$ recursively.

The modification of the program from class can be made introducing a variable Optimal($p$) 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 */
    $p$=ExtractPixel($v$);  /* function that extracts pixel p from node $v=(p,\theta)$ */
    Optimal ($p$)=$\infty$;  /* function carrying the optimal cost for pixel p of node $v=(p,\theta)$ */
end loop

loop for $\theta$  
    /* 8 angle values */
    $\Phi_{\tau-1}(p^*,\theta)=0$  /* initializing at all nodes $u^*$: $\Phi^*_{\tau=1, u^*}=0$, where $u^*=(p^*,\theta)$ */
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∗(u) + γ min (|θu − θv|, 2π − |θu − θv|);
    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
        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 final pixel p, find the corresponding optimal node vτ∗=(p,θ∗(p)) at length τ=τ∗(p) and angle θ=θ∗(p); and then backtrack using the recurrent formula vτ−1∗ = backτ(vτ∗).

Try γ = 15, 20, 30.