Consider the images

![Test Images](image1.png)

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

\[ E(v_1,\ldots,v_C) = \sum_{r=1}^{C} E_r(v_{r-1},v_r) = \frac{1}{C} \sum_{r=1}^{C} f_r(v_r) + G_r(v_{r-1},v_r) \]

where \( g(|\theta_r - \theta_{r-1}|) \) is linear and \( f_r(p_r,\theta_r) = T(s)[|D_\rho \hat{I}(p_r,\theta_r,s)|+1] \). We will consider T(s)=1 and investigate the parameter \( \gamma \).

Note that we have eight different angles \( \theta_r = -\frac{\pi}{2}, -\frac{3\pi}{4}, 0, \frac{\pi}{4}, \frac{3\pi}{4}, \frac{\pi}{2}, \pi, \frac{5\pi}{4} \) but we have for each pixel \( p_r \) only four derivatives since the magnitude of the derivatives are the same for \( \theta_r \) and for \( \theta_r + \pi \). Thus, when writing the program, make sure to consider \(|D_\rho \hat{I}(p_r,\theta_r+s)|\) to make all fit positive values as we don’t want to display them. We will use their values as they come which can vary from -255 to 255, but with the absolute values they will be from 0 to 255. Also the function \( g(|\theta_r - \theta_{r-1}|) \) to be used is linear, or \( g(|\theta_r - \theta_{r-1}|) = \min(|\theta_r - \theta_{r-1}|, 2\pi-|\theta_r - \theta_{r-1}|) \) to account for the circular nature of angles. We may call \( |\theta_r - \theta_{r-1}|_{\text{circular}} = \min(|\theta_r - \theta_{r-1}|, 2\pi-|\theta_r - \theta_{r-1}|) \)

The parameter \( \gamma \) will have to be estimated. The range of values of \( f_3(p_r,\theta_r) \) is from the highest contrast 0.004=1/256 to the lowest contrast 1. The range of the second term is \( 0 \leq \gamma |\theta_r - \theta_{r-1}|_{\text{circular}} \leq \gamma \pi \).
and it should be just a bias to differentiate similar contours, i.e., it should not dominate the data term \( f_\gamma(p, \theta_t) \). So, choosing \( 0 \leq \gamma \leq 0.5 \) with a 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 \( p_A \) and the end pixel \( p_B \), the program returns the optimal contour between \( p_A \) and \( p_B \) in red color overlaid over the black and white image. For the larger images, to reduce memory requirements, one may create a box around \( p_A \) and \( p_B \) (large enough to include the desired solution in the box).

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/vertices 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). Then the program will output the best contour from \( p_A \) to \( p_B \).

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

Initialization of the selected pixel \( p_A \): For all angles, the nodes \( u_A = (p_A, \theta) \) should be assigned \( \Phi_{\tau=1}(u_A) = f_{s\theta\phi}(p_A, \theta) \). Keep track/store for each node \( v \) the minimum average cost value \( \Phi_{\tau=1}/\tau \) (stored in the variable \( \text{Optimal}(p) \)) as well as its corresponding length \( \tau \). Once the program ends at \( \tau = C \), and for each pixel “\( p_B \)”, select the optimal cost corresponding to the best node \( v^* = (p_B, 0) \) and its optimal length \( \tau^* \). Finally one needs to backtrack the optimal path starting at this optimal node \( v^* \) back to one of the \( u_A \). Here we can again use \( v_{\tau^*} = \text{back}, (v_{\tau^*}) \) recursively. More precisely,

```
Contour-Detection DP( Image, C, p*, \gamma)

/* u_A are the set of eight nodes corresponding to initial pixel p_A */
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 p=1,...,N
    loop for \( \theta = -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{3\pi}{4}, \frac{\pi}{2}, \frac{5\pi}{4} \) /* 8 angle values */
        If (p==p_A) \( \Phi_{\tau=1}(p_A, \theta) = f_{s\theta\phi}(p_A, \theta) \); /* initializing at all nodes \( u_A \) */
        else \( \Phi_{\tau=1}(v) = \infty; \) /* first column place very large values, e.g., 1,000,000 */
    end loop
end loop

Optimal ( p_B ) = \infty; /* function carrying the optimal cost for end pixel p_B */
```
/* precomputation of the transition costs for storing in memory, can be stored in an array of size (8, 8*N) for a 8 neighbor structure */

loop for v in V /* nodes at τ */
    loop for u such that e(u,v) ≠ ∞ /*u represents the 8 neighbors at τ−1 that can reach v */
        E(v,u) = f_{s,j}(v) + γ |θ_v − θ_u |_{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 (Φ_{τ−1}(u) + E(v,u) < F);
                F = Φ_{τ−1}(u) + E(v,u);
                back_{τ}(v) = u;
        end loop
        Φ_τ(v) = F;
    end loop
loop for θ=-π/4, −π/4, 0, π/4, 3π/4, π/4, 5π/4 /* 8 angle values */
    if ( 1/τ Φ_τ(p_B,θ) < Optimal(p_B) ) {
        Optimal(p_B)= 1/τ Φ_τ(p_B,θ); /* getting the best average cost path */
        τ*'(p_B) = τ;
        θ*'(p_B) = θ;
    }
end loop
end loop

In order to obtain the optimal contour to p_B, find the corresponding optimal node v*_{τ⋆}= (p_B,θ*) at length τ=τ*(p_B) and angle θ=θ*(p_B); and then backtrack using the recurrent formula v*_{τ−1}= back_{τ} (v*_{τ}).