INTRODUCTION TO CONTOURS

What are image contours?

**Example:** Binary B & W images - set of locations of level crossings, from 0 to 1, which means closed curves defining different regions.

*Black & White:* straight lines suffer from discretization effects that create zigzag lines according to the angle of inclination (0 and $\pi/2$ angles suffer no effect).

What about these Kaniza images below? The contours include coordinates where no contrast exists.

- a. Kanizsa Triangle
- b. Curvy shapes are also perceived.

These images suggest that contours are objects that are not only defined by the contrast. One may hypothesize that contour perception must be restricted to closed curves.
Gray level images: the gray values are related to the scene radiance, and are affected by blurring of the lenses, noise, and pixel discretization. How one defines the contours?

Should contours be closed in images, so that they represent a well-defined region?

Is there any open contour in images?

Should we consider contours that self-intersect?

Should we consider contours that bifurcate?

Good models of contours in scenes are needed to extract the “most likely” boundary present in an image.
1. The space and graph where contours are embedded.

Given (i) pixels \( p=(x,y) \) (ii) measurements \( \hat{I}(x,y,\theta,s) \) for the 16 directions \( \theta_i = 0, \pi/6, \ldots, 11\pi/6 \) (or \( \{\theta_i; k=1,2,\ldots,16\} \) ) and different scales, \( s=3,5,7 \).

Create a directed graph \( G(V,E) \), with node/vertex \( v=(p,\theta) \) and edge \( e(v,u) \).

![Graph representation](image)

**Figure 2.** (a) A Graph \( G(V,E) \) represented by 10 x 6 x 16 voxels/vertices. (b) a pixel \( p \) and its sixteen near neighbors.

**Graph for curves → Edges \( e(v,u) \):** if a curve passes by pixel \( p \) with direction \( \theta \), then \( q \) is the first pixel reached after \( p \) (in the direction \( \theta \)), with \( q(p,\theta) = p + (\Delta_{qx} \cos \theta, \Delta_{qy} \sin \theta) \). However, the direction at \( q \) will still be unspecified. Thus, the neighbors of \( (p,\theta) \) are all 16 nodes \( \{u_l = (q,\phi_l); l=1,\ldots,16\} \), represented as a column at pixel \( q \). Another way to say is that the only non zero edges, leaving the node \( v=(p,\theta) \) are the nodes \( \{u_l = (q,\phi_l); l=1,\ldots,16\} \).

![Graph representation](image)

**Edges leaving \( v \), i.e., \( e(v,.) \):** if pixel \( q \) is the first pixel reached by \( p \) in the direction \( \theta \), i.e., \( q(p,\theta) = p + (\Delta_{qx} \cos \theta, \Delta_{qy} \sin \theta) \), then all 16 nodes \( \{u_l = (q,\phi_l); l=1,\ldots,16\} \),
represented as a column at pixel \( q \), will be reached by \( v \) (figure 2a). Note that given a pixel \( p \), all 16 neighbor pixels to \( p \) will be reached by one of the 16 directions \( \theta \).

**Edges reaching \( v \), i.e., \( e(\cdot, v) \):** are represented by 16 different nodes, each one corresponding to a distinct pixel coordinate and a distinct orientation, i.e., \( e(u_l, v) \neq \infty \) for 16 nodes \( \{ u_l = (q_l, \phi_l); l=1,...,16 \} \) such that

\[
 p = q_l + (\Delta_{x,y} \cos \phi_l, \Delta_{x,y} \sin \phi_l) \quad \text{(see figure 2b).}
\]

It is worth to note that given one node, say \( v \), there is only one node among the neighbor column \( \{ u_l = (q, \phi_l); l=1,...,16 \} \) that has an edge coming back at \( v \).

### 2. Open and Closed Contours

![Figure 3](image)

**Figure 3.** Contours to be detected: (a) An open contour (b) a closed contour (c) self-intersecting contours: open and closed ones (d) a contour that bifurcates at \( v_c \) into two contours.

Say a contour \( \gamma \) of size \( C \) is represented by a sequence of pixels in an image, i.e., \( \gamma_C = \{ p_t; \tau = 1,...,C \} \). Then, \( \gamma_C \) is also represented by a path \( \Gamma_C = \{ v_\tau = (p_\tau, \theta_\tau); \tau = 1,...,C \} \) in the graph \( G(V,E) \), such that each vertex/node \( v_\tau \), at step \( \tau \), is described by the pixel \( p_\tau \) and the tangent angle \( \theta_\tau \) of the contour at pixel \( p_\tau \).

An open contour is given by an open path in the graph \( G(V,E) \) (see figure 3a) or a sequence of neighbor nodes. A closed contour in the image of size \( C \) is given by a closed path in the graph \( G(V,E) \), i.e., we require that \( p_1 = p_C \). Moreover, we may require smoothness and then \( v_1 = v_C \), i.e., \( (p_1, \theta_1) = (p_C, \theta_C) \) (see figure 3b). A self-intersecting contour in the image contains two intermediate nodes \( v_{\tau_1} \) and \( v_{\tau_2} \) such that \( p_{\tau_1} = p_{\tau_2} \) and in general \( \theta_{\tau_1} \neq \theta_{\tau_2} \) (see figure 3c), though two circles touching by their tangents will exhibit \( \theta_{\tau_1} = \theta_{\tau_2} \). A contour that bifurcates is more difficult to be described by an ordered sequence of points as is better described by a tree structure (with as many branches as the image supports). For the parameterization, \( \tau = 1,...,C \), one of the branches is visited before the other, so there is a jump of coordinates once the first branch visit is finished (once each branch visit is finished, in case of many branches). Moreover, the bifurcation pixel is allowed to precede two (or more) distinct points in the “curve” (see figure 3d).

When an image \( I \) is given, what is the criteria to choose a contour \( \Gamma \) of size \( C \)?

### 3. The Bayesian Framework

Say we consider all possible contours \( \Gamma \) of size \( C \) pixels. Are they equally likely to occur in images? If we were to collect a large sample size of images and trace contours of size \( C \) pixels, we could have an estimate of what contours of size \( C \) pixels are more likely to occur.

One may expect that the higher is the pixel contrast the more likely to be part of contours. One may also expect that the more smooth contours are more likely to be present in real images.
So let us treat each contour $\Gamma$ as an instance of a random variable or an instance of a set of random variables defining the pixel locations of a contour. So $\Gamma$ is a random variable. Say the size of a contour is also random, i.e., $C$ is a random variable. Say any given Image is an instance of a random variable $I$, i.e., $I_N$ represent all possible different images of a size $N \times N$.

We can then define $P(I|I_N, C)$ as the probability of a any contour $\Gamma$ of a given length $C$ being present in a given image $I_N$ of size $N$.

We follow the approach that a contour $\Gamma^*_C$ is selected to maximize a probability of occurrence of contours of size $C$ given an image $I_N$, i.e.,

$$\Gamma^*_C = \arg \max_{\Gamma_C} P(\Gamma | I_N, C)$$

$$= \arg \max_{\Gamma_C} P(I_N | \Gamma, C) P(\Gamma | C) / P(I_N | C) \quad \text{Bayes rule}$$

$$= \arg \max_{\Gamma_C} P(I_N | \Gamma, C) P(\Gamma | C) \quad P(I_N / C \text{ is constant}) \Gamma_C$$

Where $x^* = \arg \max_x f(x)$ represents the value of $x$ that maximizes $f(x)$. We have used Bayes rule, i.e., that $P(A, B | C) = P(A | B, C) P(B | C)$ and $P(A, B | C) = P(B | A, C) P(A | C)$ and combining both equations we have $P(A | B, C) = P(B | A, C) P(A | C) / P(B | C)$

Many times it is preferable to work with the $-\log \left[ P(I|\Gamma, C) P(\Gamma | I_N) \right]$ function, and so

$$\Gamma^*_C = \arg \max_{\Gamma_C} P(I | \Gamma, C) P(\Gamma | C) = \arg \min_{\Gamma_C} -\log P(I | \Gamma, C) P(\Gamma | C)$$ \hspace{1cm} (1)

Moreover, we can require such a contour $\Gamma^*_C$ to be a closed one, or allow to be an open one, or to be any contour, including contours that self-intersect. We don’t really know these probabilities, so we build (ad-hoc) models for them.

4. Modeling Contours in Images, $P(I|\Gamma, C)$

The model for the image formation, $P(I|\Gamma, C)$, favor images such that when a contour is present in a pixel $p_\tau$ at orientation $\theta_\tau$ a probability dependent on $|D_{\rho} \hat{I}(p_\tau, \theta_\tau, s)|$ with $\hat{\rho} = (-\sin \theta_\tau, \cos \theta_\tau)$ should result. Let us say $T(s)$ is an edge threshold such that if $|D_{\rho} \hat{I}(p_\tau, \theta_\tau, s)| > T(s)$ it indicates that very likely the image pixel is of an edge present at orientation $\theta$. Of course the higher $|D_{\rho} \hat{I}(p_\tau, \theta_\tau, s)|$ is the more likely an edge is present at orientation $\theta_\tau$. We are implicitly assuming that we can assign an independent probability for each pixel in the contour $P_c(I | p_\tau, \theta_\tau)$ and also on the rest of the image. With this assumption, the probability for the entire image is written as

$$P(I | \Gamma, C) = \prod_{\tau=1}^{C} P_c(I | p_\tau, \theta_\tau) \prod_{z=1}^{N-C} P_{N-C}(I | p_z, \theta_z)$$

where $N$ is the number of pixels in the image and the probability $P_{N-C}(I | p_z, \theta_z)$ refers to each pixel that is not where the contour is present. We propose a model for $P_c(I | p_\tau, \theta_\tau)$

$$P_c(I | p_\tau, \theta_\tau) = \frac{1}{Z_l} \exp\left( -t(s) / \langle |D_{\rho} \hat{I}(p_\tau, \theta_\tau, s)| \rangle + 1 \right)$$
where the benefit for $|D_p \hat{I}(p, \theta, s)|/T$ being large is expressed. The extra “1” value is added the smallest constant to avoid a singularity and also to express that even where no contrast occur, a contour may be present (think of the illusory contours shown on the first pages). To obtain a value for $Z$, we must normalize across all different image values. Let us skip for now the calculation of $Z$.

$$P_c(I | p, \theta) = \frac{1}{Z} \exp \left( -T(s) \right) \left( |D_p \hat{I}(p, \theta, s)| + 1 \right)$$

Figure 4. Graph of the distribution of pixel contrast. One could learn it by constructing histograms from human drawings of contours over images.

Note that a more precise model for this probability could be learned. One could provide several image samples and request a user to draw contours. Then, produce the histogram for each $|D_p \hat{I}(p, \theta, s)|/T$ of the set of pixels where contours were present. A normalized histogram would represent the probability, the more pixels a given value $|D_p \hat{I}(p, \theta, s)|/T$ represents the larger the probability.

We sometimes can think the probability in terms of costs (the lower the cost the higher the probability) and then we can rewrite

$$P_c(I | p, \theta) = \frac{e^{-f_s(p, \theta)}}{Z}$$

where

$$f_s(p, \theta) = T \left( |D_p \hat{I}(p, \theta, s)| + 1 \right)$$

Note that for perfect homogeneous areas, where no edges are present, the cost becomes $T$. For high contrast areas, regardless of the value for the numerator (as long as it is not very high), the cost function will have small values.

We can then assign the independent probability for each pixel in the contour, where $Z$ is the normalization constant. This implies that for the entire contour, with the independence assumption, we obtain

$$\prod_{r=1}^C P_c(I | p, \theta) = \prod_{r=1}^C \frac{e^{-f_s(p, \theta)}}{Z} = \prod_{r=1}^C \frac{1}{Z} \exp \left( -\sum_{r=1}^C f_s(p, \theta) \right)$$

We now must model $\prod_{\tau=C+1}^{N-C} P_{N-C}(I | p, \theta)$ for pixels outside the contour. Since we don’t know anything about the image outside where the contour is, any gray value of I is as likely as any other. We can express this by the uniform distribution on the gray values at each pixel, or
\( p = 1/\# \), where \# are the number of gray values. For an 8 bit image \( \# = 256 \). In an image with \( N \) pixels, there are \( N - C \) pixels that are not the given contour pixels and so the probability of each set of image values for these \( N - C \) pixels is 
\[
\prod_{i=1}^{N-C} P_{N-C}(I | p, \theta) = \rho^{N-C}.
\]
One could “improve” this model by arguing that neighbor pixels to the contour pixels are likely to have some contrast (there is usually some blur near image boundaries), but we limit ourselves here not considering this effect. The (ad-hoc) model for the image formation given a contour of length \( C \) is then written as
\[
P(I | \Gamma, C) = \frac{1}{Z_1} \exp \left( -\sum_{i=1}^{C} f_i(p, \theta) \right) \rho^{N-C} \prod_{i=1}^{C} e^{-f_i(p, \theta) - \lambda}
\]
where \( Z_1 = (Z_2)^C / \rho^N \) is a normalization constant and \( \lambda = -\log \rho \) accounts for all pixels in the image (8 bit images \( \rho = 1/256, \lambda = 8 \)). For the formula above, we considered the following calculations
\[
\rho^{N-C} = \rho^N \rho^{-C} = \rho^N e^{\log \rho^{-C}} = \rho^N e^{-C \log \rho} = \rho^N e^{-\sum_{i=1}^{C} \log \rho} = \rho^N e^{-\sum_{i=1}^{C} \lambda} \Rightarrow
\]
\[
\frac{1}{Z_1} e^{-\sum_{i=1}^{C} f_i(p, \theta)} \rho^{N-C} = \frac{1}{Z_1} \frac{\rho^N}{\rho} e^{-\sum_{i=1}^{C} f_i(p, \theta)} e^{-\sum_{i=1}^{C} \lambda} = \frac{1}{Z_1} \prod_{i=1}^{C} e^{-f_i(p, \theta) - \lambda}
\]

**Gray level consistency:**

We could add a term that prefers images such that the gray value along the sides of the contour do not vary much, at least on one side of the contour. This is not always the case in real images, for example checkerboard images where contrast alternates sides along an edge (near corners). However, gray level consistency will occur for most images with contours and we can add a bias for these images. One can write the cost/probability for this bias as

\[
C(v, v_{-1}) = \min ( |I(p, \theta + \pi/2, s) - \hat{I}(p_{-1}, \theta_{-1} + \pi/2, s)| , |I(p, \theta + 3\pi/2, s) - \hat{I}(p_{-1}, \theta_{-1} + 3\pi/2, s)| )
\]
so that the bias occur since the cost should be small and as boundary condition we can define \( C(v_1, v_0) = 0 \). Thus, the probability becomes
\[
P(I | \Gamma, C) = \frac{1}{Z_1} \exp \left( -\sum_{i=1}^{C} f_i(p, \theta) + \varepsilon C_s(p, \theta, p_{-1}, \theta_{-1}) - \lambda \right) = \frac{1}{Z_1} \prod_{i=1}^{C} e^{-\hat{f}_i(p, \theta, p_{-1}, \theta_{-1})}
\]
where \( \hat{f}_i(p, \theta, p_{-1}, \theta_{-1}) = f_i(p, \theta) + \varepsilon C_s(p, \theta, p_{-1}, \theta_{-1}) - \lambda \) and \( \varepsilon \) controls the weight of the gray level consistency. We will not follow up on this extra formula, but it is solvable by the same methods we develop here.

**5. Prior Models of Contours, \( P(\Gamma | C) \)**

A natural prior model of contours of size \( C \) pixels, \( P(\Gamma | C) \), favor smooth and short contours, i.e., contours such that the tangent angles don’t change too much and the ones exhibiting small euclidean distance (given \( C \) pixels). Smoothness will ensure that straight lines are preferred over curvy lines (of the same \( C \) number of pixels) and curvy lines are preferred over
“zig-zag” lines. The justification for smoothness is that most contours found in images are smooth (to some fine enough scale). Worth to note that when we, humans, draw objects such as contours, freely, i.e., randomly according to the unconscious bias to contours, we tend to produce smooth contours. It is a suggestion of a prior model of contours.

One simple (ad hoc) model (that do not account for bifurcations of the contour) is

$$P(\Gamma | C) = \frac{1}{Z_2} \prod_{i=1}^{C} e^{-\gamma g(\theta_i, \theta_{i-1})} = \frac{1}{Z_2} \exp \left( -\gamma \sum_{i=1}^{C} g(|\theta_i - \theta_{i-1}|) \right)$$

where $Z_2$ is a normalization constant, $g(x)$ is a monotonic increasing function, and we apply the initial conditions that $\theta_1 = \theta_0$, $p_1 = p_0$ (i.e., the first term vanishes).

$$P(\Gamma | C) = \frac{1}{Z_2} \exp \left( -\gamma \sum_{i=1}^{C} g(|\theta_i - \theta_{i-1}|) \right)$$

Figure 5. A graph of how the probability of the change in angle varies for contours. One could learn it by constructing histograms from human drawings of contours over images.

The specific function $g(x)$ affects the bias one imposes on smoothness. Say our variables $\theta_i$ are all scaled and discretized so that $x = |\theta_i - \theta_{i-1}| > 1$ or $x = |\theta_i - \theta_{i-1}| = 0$. If $g(x)$ is a convex function (such as $g(x) = x^2$), then corners or sharp changes of angle will be more unlikely to occur as $|\theta_i - \theta_{i-1}|$ becomes large. If $g(x)$ is a concave function (such as $g(x) = \sqrt{x}$), then corners will be more likely to occur. Once again, a learning procedure could be applied. A histogram of a set of sampled contour images for different contour angle changes, $x = |\theta_i - \theta_{i-1}|$, could be plotted and normalized to represent the probability for each value of $x$. We adopt the linear model (absolute differences) as it is simple and reasonable to use.

Given the same number of pixels, horizontal and vertical lines exhibit smaller Euclidean distance. Thus, by introducing a bias towards smaller euclidean distance one will effectively bias towards horizontal and vertical lines. Is this desired? Again, the answer could be found by taking many image samples and drawing contours with a given euclidean distance and verify if there are bias towards horizontal and vertical lines. Of course, this may vary according to the set of samples obtained. If the sample images come from man made objects, it is likely to be the case. One can introduce the bias for horizontal and vertical lines by adding a term that measures the euclidean distance, and a new control parameter $\mu$ to weight this bias is introduced as follows

$$\left\{ \gamma \sum_{i=1}^{C} g(|\theta_i - \theta_{i-1}|) \right\} \rightarrow \left\{ \gamma \sum_{i=1}^{C} g(|\theta_i - \theta_{i-1}|) + \mu |p_i - p_{i-1}| \right\}.$$
6. Posterior Models of Contours given the Images, \( P(I|\Gamma,C) \)

The final product \( P(I|\Gamma,C) P(\Gamma|C) \) is then written as

\[
P(I|\Gamma,C) P(\Gamma|C) = \frac{1}{Z} \prod_{\tau=1}^{C} e^{-f_{\tau}(v_{\tau}) - G_{\tau}(v_{\tau},v_{\tau-1})} = \frac{1}{Z} e^{\sum_{\tau=1}^{C} f_{\tau}(v_{\tau}) + G_{\tau}(v_{\tau},v_{\tau-1})} = \frac{1}{Z} e^{-E(v_1,...,v_C)} \tag{2}
\]

where \( G_{\tau}(v_{\tau},v_{\tau-1}) = \gamma g(|\theta_{\tau} - \theta_{\tau-1}|) \) is the value of the edge between nodes, \( Z \) is a new constant, and

\[
E(v_1,...,v_C) = \sum_{\tau=1}^{C} E_{\tau}(v_{\tau},v_{\tau-1}) = \sum_{\tau=1}^{C} f_{\tau}(v_{\tau}) + G_{\tau}(v_{\tau},v_{\tau-1}) \tag{3}
\]

If we choose \( Z \) to be a normalization constant (partition function) so that the sum over all contours of \( \frac{1}{Z} e^{-E(v_1,...,v_C)} \) is 1, then we will have created the posterior distribution \( P(\Gamma|I_C,C) \). It is not however necessary to estimate \( Z \) in order to obtain the optimal contour since we can apply (2) to (1) to obtain

\[
\Gamma^*_C = \arg \min_{\Gamma} -\log P(I|\Gamma,C) P(\Gamma|C) = \arg \min_{\Gamma} E(v_1,...,v_C) = \arg \min_{\Gamma} \sum_{\tau=1}^{C} E_{\tau}(v_{\tau},v_{\tau-1}) \tag{4}
\]

7. Extracting Contours of size \( C \) from Images (Open and Closed)

Equation (4) is amenable to the dynamic programming method. More precisely, by defining

\[
F_{C-1}(v_{C-1}) = \min_{v_C} E_C(v_C,v_{C-1}) \quad \text{and} \quad F_{\tau-1}(v_{\tau-1}) = \min_{v_\tau} E_\tau(v_\tau,v_{\tau-1}) + F_\tau(v_{\tau}) \quad \text{equation (4) has the following recurrent property}
\]
\[
\min_{\tau} E(\Gamma) = \min_{\gamma_1, \ldots, \gamma_C} \sum_{\tau=1}^{C} E_\gamma (v_\tau, v_{\tau-1}) \\
= \min_{\gamma_1, \ldots, \gamma_C} \left( \sum_{\tau=1}^{C-1} E_\gamma (v_\tau, v_{\tau+1}) + \min_{v_c} E_c (v_C, v_{C-1}) \right) \\
= \min_{\gamma_1, \ldots, \gamma_C} \left( \sum_{\tau=1}^{C-1} E_\gamma (v_\tau, v_{\tau+1}) + F_{C-1} (v_{C-1}) \right) \\
= \min_{\gamma_1, \ldots, \gamma_{C-1}} \left( \sum_{\tau=1}^{C-2} E_\gamma (v_\tau, v_{\tau+1}) + \min_{v_{C-1}} E_{C-1} (v_{C-1}, v_{C-2}) + F_{C-1} (v_{C-1}) \right) \\
= \min_{\gamma_1, \ldots, \gamma_{C-2}} \left( \sum_{\tau=1}^{C-2} E_\gamma (v_\tau, v_{\tau+1}) + F_{C-2} (v_{C-2}) \right) \\
= \min_{\gamma_1, \gamma_2} \left( \sum_{\tau=1}^{2} E_\gamma (v_\tau, v_{\tau+1}) + F_2 (v_2) \right) \\
= \min_{\gamma_1} E_1 (v_1, v_0) + \min_{\gamma_2} E_2 (v_2, v_1) + F_2 (v_2) \\
= \min_{\gamma_1} E_1 (v_1, v_0) + F_1 (v_1) \\
= \min_{\gamma_1} \Phi_\gamma (v_1) + F_1 (v_1) \\
\tag{5}
\]

and so the dynamic programming algorithm can be applied as follows: we create at each step \( \tau \) a column matrix of size 16xN (number of nodes in the graph \( G(V,E) \), i.e., the size of \( V \)) . Every entry/node \( v \in V \) contains the cost to reach node \( v \) at step \( \tau \), and it is stored in the matrix \( \Phi_\tau^*(v) = \Phi_\tau^*[\tau,v] \). From this matrix and the costs \( E_\tau(v_\tau, v_{\tau+1}) \) we obtain \( \Phi_{\tau+1}^*(v) = \Phi_{\tau+1}^*[\tau+1,v] \). We proceed with the same process forward on \( \tau \) until \( \tau = C \) and thus, we fill all the entries of the matrix \( \Phi_\tau^*[\tau,v] \).

We also should initialize the first column matrix as \( \Phi_1^*[\tau=1,v] = \Phi_1^*[\tau=1,v] = 0 \). Sometimes the user may have a choice (strong bias) of a starting point. Say the user wants the contour to start at node \( u \). In this case we can set \( \Phi_1^*[\tau=1,u] = 0 \) and \( \Phi_1^*[\tau=1,v] = \infty \) for all other nodes \( v \neq u \). Nodes with large costs assure that they will never be used to minimize the cost. We also keep a matrix \( \Psi_\tau^*(v) \) to remember the (parent) previous node to \( v \) at time \( \tau \) so that a path can be retrieved. Dynamic programming works on a graph \( \Phi(T, V, E) \) which are effectively \( T = C \) copies of the graph \( G(V,E) \). Figure 4 below illustrates the graph and dynamic programming. Note that if the user selects starting point and end point, say \( u_1 \) and \( u_2 \) respectively, one can find the best contour of size \( C \) pixels between these two nodes by initializing the matrix \( \Phi_1^*[\tau=1, u_1] = 0 \) and \( \Phi_1^*[\tau=1, v] = \infty \) for all other nodes \( v \neq u \), and we backtrack the solution from the matrix element \( \Phi_1^*[\tau=C, u_2] \).
Figure 4. The graph $\Phi(T=C, V, E)$ which are effectively $C$ copies of the graph $G(V,E)$, i.e., represented as $C$ copies of a column of $V$ vertices and $E$ edges between columns. Dynamic programming works from left to right, by solving for each node in a column (known as “subproblems” at time $\tau$).
The pseudocode for the dynamic programming (DP) algorithm becomes

Contour-Detection DP( Image, C, u*)

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)

\[
\text{loop for } v \text{ in } V \\
\Phi_{\tau-1}(v)=\infty; \quad /* \text{first column */} \\
\text{end loop} \\
\Phi_{\tau-1}(u^*)=0 \quad /* \text{initializing at } u^* : \Phi^*[\tau=1, u^* ]=0 */ \\
\text{loop for } v \text{ in } V \\
\text{loop for } u \text{ such that } e(u,v) \neq \infty /* u \text{ represents the } 8 \text{ neighbors that can reach } v */ \\
E(v,u) = f^*_\tau(v) + G^*_\tau(v,u) /* \text{precomputation of the transition costs can be stored in an array of size } (8, 8* N) \text{ for a } 8 \text{ neighbor structure */} \\
\text{end loop} \\
\text{end loop} \\
\text{Main loop} \\
\text{loop for } \tau=2, 3, ..., C \\
\text{loop for } v \text{ in } V \\
F=\infty; \\
\text{loop for } u \text{ such that } e(u,v) \neq \infty /* \text{consider only neighbors } u \text{ that can reach } v */ \\
\quad \text{if } (\Phi_{\tau-1}(u) + E(v,u) < F); \\
\quad F=\Phi_{\tau-1}(u) + E(v,u); \\
\quad \text{back}_\tau(v)=u; \\
\text{end loop} \\
\Phi^*_\tau(v)=F; \\
\text{end loop} \\
\text{end loop} \\
\text{end}
\]

The complexity of the algorithm is \( O(C E) \), which is \( O(256 \ C \ N) \). To extract the optimal contour \( C^* = \{ v^*_0, v^*_1, v^*_2, ..., v^*_C \} \), we backtrack from the optimal final state \( v^*_C \) using the recurrent formula \( v^*_{\tau-1} = \text{back}_\tau(v^*_\tau) \).

\[\text{a. Detecting open contours of size } C:\]
Find \( v^*_C \) that minimizes \( \Phi_C^*(v) \) at time step \( \tau=C \), i.e., \( v^*_C = \text{argmin}_{v} \Phi_C^*(v) \). Then backtrk using \( v^*_{\tau-1} = \text{back}_\tau(v^*_\tau) \) recursively.

\[\text{b. Detecting open contours from } u^* \text{ to } u^{**} \text{ of size less than } C:\]
Run dynamic progrmizing with initialization \( u^*, \Phi_{\tau=1}(u^*)=0 \). Keep track of all nodes \( u^{**} \), i.e., store the minimum cost value \( \Phi^*_\tau(u^{**}) \) node as well as its corresponding length \( \tau^* \). Once the program ends at \( \tau=C \), select the optimal length \( \tau^* \) that has the lowest cost.
\( \Phi^*_\tau(u^{**}) \) for node \( u^{**} \). Finally, backtrack the optimal path starting at node \( u^{**} \) at length \( \tau^* \) back to \( u \) at length \( \tau=1 \), again using \( v^*_\tau=\text{back}(v^*_\tau) \) recursively.

The modification of the program above can be made introducing a variable Optimal and \( \tau^* \).

\textbf{loop for} \( \tau=2, 3, \ldots, C \) 
\hspace{1em} \text{Optimal} = \infty; 
\hspace{1em} \text{loop for} \ v \text{ in V} 
\hspace{2em} F = \infty; 
\hspace{2em} \text{loop for} \ u \text{ such that } e(u,v) \neq \infty /* \text{consider only neighbors } u \text{ that can reach } v */ 
\hspace{3em} \text{if } (\Phi_{\tau-1}^*(u) + E(v,u) < F); 
\hspace{4em} F = \Phi_{\tau-1}^*(u) + E(v,u); 
\hspace{4em} \text{back}_\tau(v) = u; 
\hspace{3em} \text{end loop} 
\hspace{2em} \Phi^*_\tau(v) = F; 
\hspace{2em} \text{end loop} 
\hspace{1em} \text{if } (\Phi^*_\tau(u) < \text{Optimal}) \{ 
\hspace{2em} \text{Optimal} = \Phi^*_\tau(u); 
\hspace{2em} \tau^* = \tau; 
\hspace{2em} \text{end loop} 
\hspace{1em} v^*_\tau=\text{back}(v^*_\tau) /* \text{where the first node is } v^*_\tau= u^{**} \tau^* */ 

\textbf{c. Detecting open contours from pixel } p^* \text{ to all pixels of size less than } C: \text{ Run dynamic programing with initialization at all nodes } u^*=(p^*,0), \text{ i.e., } \Phi_{\tau=1}^*(u^*)=0. \text{ Keep track of all nodes, i.e., store the minimum cost value } \Phi^*_\tau(v), \text{ the node } v \text{ as well as its corresponding } \tau^*. \text{ Once the program ends at } \tau=C, \text{ and for each pixel "p_v" select the optimal cost corresponding to all angles, i.e., best node } v^*=(p_v,0)^* \text{ corresponding to each pixel } p_v \text{ and its optimal length } \tau^*. \text{ Finally backtrack the optimal path starting at this optimal node } v^*_\tau \text{ back to } u^*_{\tau=1}, \text{ again using } v^*_\tau=\text{back}(v^*_\tau) \text{ recursively.} 

The modification of the program above can be made introducing a variable Optimal(v) for each \( v=(p_v,0) \) and its corresponding optimal length \( \tau^*(v) \). More precisely

\textbf{loop for} \( \tau=2, 3, \ldots, C \) 
\hspace{1em} \text{loop for} \ v \text{ in V} 
\hspace{2em} F = \infty; 
\hspace{2em} \text{Optimal}(v) = \infty; 
\hspace{2em} \text{loop for} \ u \text{ such that } e(u,v) \neq \infty /* \text{consider only neighbors } u \text{ that can reach } v */ 
\hspace{3em} \text{if } (\Phi_{\tau-1}^*(u) + E(v,u) < F); 
\hspace{4em} F = \Phi_{\tau-1}^*(u) + E(v,u); 
\hspace{4em} \text{back}_\tau(v) = u; 
\hspace{3em} \text{end loop} 
\hspace{2em} \Phi^*_\tau(v) = F; 
\hspace{2em} \text{if } (\Phi^*_\tau(v) < \text{Optimal}(v)) 
\hspace{3em} \text{Optimal}(v) = \Phi^*_\tau(v); 
\hspace{3em} \tau^*(v) = \tau; 
\hspace{2em} \text{end loop} 
\hspace{1em} \text{end loop}
to obtain the optimal contour for any selected pixel pᵣ, find among all angles the best v=(pᵣ,θ) value and its corresponding \( \tau^*(v) = \tau \); and then backtrack.

More precisely, find \( \theta^* = \text{argmin}_θ \text{Optimal}(pᵣ,θ) \) and its corresponding \( \tau^*(v=(pᵣ,\theta^*)) \). Then, use the recurrent formula to backtrack \( v_{τ-1} = \text{back}(v_τ) \).

d. Detecting closed contours (not optimal) of size C:

Here we only initialize at one vertex \( v_1 \), i.e., \( \Phi_{τ=1}^*(v_1) = 0 \) and set \( \Phi_{τ=1}^*(u) = \infty \) for \( u \neq v_1 \). After C iterations we choose the node \( [τ,v_1] \), we save the total cost \( \Phi_{C}^*(v_1) \), and we backtrack from the node \( [τ,v_1] \) to obtain the optimal closed contour passing through \( v_1 \). Why this may not be the optimal contour through \( v_1 \)? Because it is possible that some other (closed) contour with lower cost was discarded in favor of a better contour that will however not go through \( v_1 \). To obtain the closed contour through any node we must run the algorithm \( 8N \) times, starting each time at a different vertex/node and obtaining the optimal closed contour through this vertex/node. Finally one must compare the total cost of each closed contour obtained to select the optimal closed contour. A true optimal closed contour algorithm was developed by Jermyn and Ishikawa here at NYU, based on the mean cycle algorithms due to Karp (see I. Jermyn and H. Ishikawa (1999). Globally Optimal Regions and Boundaries. Proc. of 7th IEEE Intl. Conf. on Comp. Vision (ICCV’99), Kerkyra, Greece.

e. Detecting self-intersecting contours

By the virtue of the algorithm to search for all possible contours (open or closed), it will automatically extract contours that self-intersect as self intersections point almost always have the property that \( v_{τ1} \neq v_{τ2} \). since \( p_{τ1} = p_{τ2} \) but almost always \( \theta_{τ1} \neq \theta_{τ2} \).

8. Extracting Contours from Images (Open and Closed ones)

We have developed a method to extract contours of size C from images. We now want to extract optimal contours from images regardless of their size. The first step is to define “optimal contours of any size”. We may want to extract the optimal contour \( \Gamma^* \) and the optimal size \( C^* \), i.e., to find

\[
\Gamma^*_C = \text{arg max}_{\Gamma,C} P(\Gamma,C \mid I) = \text{arg max}_{\Gamma,C} P(I, \Gamma \mid C)P(C) / P(I) = \text{arg max}_{\Gamma,C} P(I, \Gamma \mid C)
\]

Assuming \( P(C) \) is uniform. Our formulation of \( P(I,\Gamma \mid C) \) have not considered the fact that longer contours tend to have higher costs. Thus, it is natural to normalize the cost (3) by the length of the contour, i.e., to compute the average cost of contours to un bias the preference for smaller contours. We then rewrite (3) as

\[
E(v_1, ..., v_C) = \frac{1}{C} \sum_{τ=1}^{C} E_τ(v_τ, v_{τ-1}) = \frac{1}{C} \sum_{τ=1}^{C} f_τ(v_τ) + G_τ(v_τ, v_{τ-1}) \quad (6)
\]

We can then apply dynamic programming to (3) for the maximum length C of a contour in an image of size N pixels (note that C \( \leq N \), since there cannot exist more than N contour pixels.) Once the computations is finished, we divide the optimal cost of each node by its corresponding length in pixels, and therefore we obtain the optimal cost (6). The optimal contour, according to (6), is then readily extracted as the minimum cost node in the entire graph (\( V \times C_{\text{max}} \)), where \#V=8 N and \#C_{\text{max}}=N. If we require the contour to be closed, we focus these selection
of the optimal node along an entire row in the graph, corresponding to the starting node. The complexity of the algorithm is $O(64 N^2)$.

After $C$ iterations we search in the graph $\Phi[T, V, E]$ along the row corresponding to the node $v_i$ for the “optimal” (closed) contour cost, i.e., we search among all nodes $[\tau, v_i]$ for the lowest value of the matrix $\Phi^*[\tau, v_i]$, i.e., among all possible $\tau$ (not including the starting point).