1. Graded Exercise: Edge Detection and Delineation

Instructions

- You are allowed 1 hour 45 minutes for this graded exercise and you have to submit your files by 10:00 AM.
- You may use your notes, books and code from previous exercise sessions, but access to Internet resources is forbidden.
- Once you are done, make sure that your files are successfully uploaded to the system.

1.1 Introduction

Recently Google Earth provided an option to visualize 3D reconstructions of the mountain chains. An important feature to identify a mountain chain is its ridge (or edge) in the image. Human vision is very good at detecting such structures but doing it automatically using computer vision algorithms is not trivial. Here we will try to delineate the ridge of a mountain as given in Fig. 1.1 using Dijkstra's algorithm.

We provide a package with a base code in the Moodle webpage of the course. You should write your code either in the file called `main.m` or in a separate function specified in the respective exercise. You can provide qualitative answers by writing them as comments in `main.m` file. Use the image inside folder `figures` as input.

(a) (b)

Figure 1.1: **(a) Input image.** Start and end positions of the ridge are displayed on the image. Pixel values for these positions are provided in the code. **(b) Detected ridge.** Ridge is detected between the start and end points using Dijkstra's algorithm and overlayed on the image.

1.2 Computing gradients (30 pts)

We will first detect the edges in the image by simply computing the gradient image. This will then be used in the next exercise session for ridge delineation.

Exercise 1.1 Computing the gradient image.

1. In `main.m`, convert the image to grayscale. Smooth the images using a Gaussian filter of size 7×7. Choose suitable σ. What is the trade-off of using bigger σ?
2. In `main.m` compute the gradients in x- and y-directions using Sobel mask.
3. Compute a gradient-magnitude image and compare it to the one obtained by

Matlab's `gradient` function.

4. Threshold gradient image to find the most pronounced edges and visualize it using `imagesc`. The output should look similar to 1.2.

■

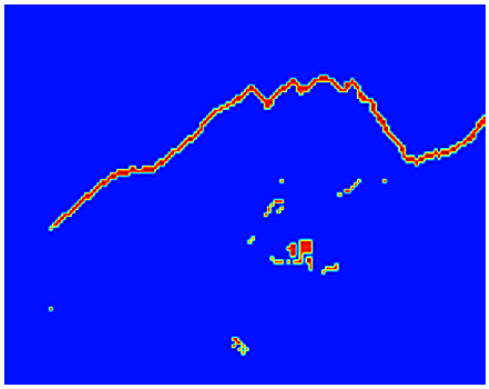

Figure 1.2: **Thresholded gradient image.** It depicts the most pronounced edges in the image.

1.3 Delineating ridges (70 pts = 30 pts + 30 pts + 10 pts)

Given the thresholded gradient image computed in the previous exercise, we will now delineate the ridge of a mountain using Dijkstra's shortest path algorithm. Roughly speaking, starting from an initial node, this algorithm looks around the neighboring pixels and chooses the pixel with the shortest distance to the current pixel as the next element of the path. Particularly, in this example, we will find the shortest paths from a starting pixel to all the other pixels in the image.

The distance of a pixel to a neighboring one can be defined by a cost value for the edge connecting these two pixels. For this particular problem, we will define the cost of the edge to pixel `(i, j)` from any of its 4-neighbors as `C − thresholded_grad(i, j)`, where `C` is a constant and `thresholded_grad(i,j)` is the value of the thresholded gradient image at pixel `(i, j)`. With such definition of cost, the algorithm will assign low cost values to high gradient regions, which are typically the regions in the image that belong to the ridges of the mountain, and therefore will choose pixels belonging to mountain ridges as the next element in the shortest path. In the end, this would effectively delineate the ridge of the mountain.

In particular, the steps of the algorithm can be summarized as follows:

1. Assign to every pixel a distance value: zero for the initial pixel and infinity for all other pixels.
2. Set the initial pixel as *current* and mark it *visited*. All the other pixels are initially unvisited. Create a set of all the visited pixels called the *visited set* (e.g., a binary matrix of 1's for visited pixels and 0's for unvisited pixels).
3. For the current pixel, consider all of its unvisited neighbors and calculate their distances. Use the edge cost defined above to measure the distance between a pixel and its neighboring one. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current pixel is marked with a distance of 15, and the edge connecting it with a neighbor

has value 2, then the distance between them will be 15 + 2 = 17. If the neighbor was previously marked with a distance greater than 17 then change it to 17 and update the previous pixel position. Otherwise, keep the current value.

4. When we are done considering all of the neighbors of the current pixel, mark the current node as visited. A visited pixel will never be checked again.
5. Select the unvisited pixel that is marked with the smallest distance, set it as the new "current pixel", and go back to step 3.

The algorithm will terminate when all the pixels in the image are visited.

Exercise 1.2 Computing the shortest paths. (30 pts) Fill in the function `dijkstra` to implement the above algorithm. The input to the function should be respectively, the thresholded gradient image, the constant `C`, and starting position of the ridge. The algorithm should return a distance matrix that encodes the shortest path from the starting point to each pixel in the image and a matrix that stores for each pixel the position of the previous pixel that lies on the shortest path to the starting point. Visualize the distance matrix using `imagesc`. The output should look like Fig. 1.3 ■

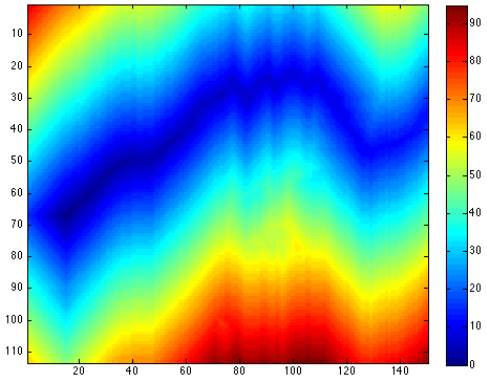

Figure 1.3: **Distance matrix.** Each entry in the matrix corresponds to the shortest path from the corresponding pixel to the start pixel.

Exercise 1.3 Ridge delineation. (30 pts) Now use the matrix that stores previous pixel positions in the shortest paths to the starting point to reconstruct the path from the end point back to the starting point. Write your code in `main.m`. Visualize the recovered path by overlaying it on the image as shown in Fig. 1.1. ■

Exercise 1.4 Evaluating the edge cost. (10 pts) In `main.m`, try various values of C in Dijkstra algorithm. How does the distance matrix and the final detected ridge change? Do you understand why? Visualize the distance matrix and the detected ridge for two more C values (one smaller and the other larger than the predefined C). ■