Recognizing Actions

Programming Assignment #5
CS253 Spring 2014
Due 6:00pm Wednesday, March 9th, 2016

Motivation

PA1 through PA4 were designed to get you to this point: recognizing actions in streams of pose data. Your PA5 program will read in a short video and a long video and answer one simple question: how well does the action shown in the short video match *some part* of the longer video?

There are two factors that make this task complex. The first is that you don't know where (if anywhere) in the long video the shorter action starts. It might start at frame 1 or frame 2 or … . The second is that the action might be slower or faster in the longer video. For example, some people raise their arms quickly, while others do it more slowly. As a result, the action might take more or fewer frames in the longer video, compared to the shorter one. The task of your program is to find a mapping from frames in the short video to frames in the long video that minimizes the total distance between the mapped frames. The smaller the total distance, the better the match.

Task

Your program will take two filenames as arguments. Both are the names of Kinect video files. Whichever video file has fewer poses is designated as the *action video,* while the file with more poses is the *target video.* Your program, called PA5, will write a single output value to std::cout. This value is the *smallest possible* total distance between the action video and the target video according to *any* legal mapping between the two.

What is a legal mapping between an action and target video? We denote frames in the action video as Action[i], and frames in the target video as Target[i]. A *mapping* assigns every frame Action[i] to some frame Target[j]. Conceptually, you can represent a mapping as a vector M, where the length of M is the length of the action video, and every value M[i] is an integer from 0 to T-1, where T is the length of the target variable. The idea is that if M[i] = j, then frame Action[i] is mapped to frame Target[j]. A *legal mapping* is one in which the values in M form a non-decreasing series. In other words, $M[i] \leq M[i+1]$ for all 'i'.

Given a mapping M, the distance between the action and target videos is the sum of the pairwise Euclidean distances between Action[i] and Target[M[i]], all action frames 'i'. The trick is finding the legal mapping M that has the smallest distance among all legal mappings.

Fortunately, there is an algorithm for this, called *dynamic time warping* (DTW). As input, DTW needs the distance between every pose in the action video and every pose in the target video. In other words, the input to DTW is the output of PA4. For the purposes of describing DTW, we will assume that the action video has N frames and the target video has M frames, with N ≤ M. The output of PA4 can then be written as a 2D distance array D, where D[i,j] is the distance between Action[i] and Target[M[i]].

DTW works by creating another array W of the same size as D, but with different semantics. W[i,j] is be the total distance of the best mapping among all mappings that (1) begin at Action[0] and (2) end at Action[i] with M[i] = j. DTW builds this array one row at a time. It starts with the first row, 'i'=0. This row represents all mappings that start at Action[0] such that M[0] = j. In other words, all mappings that only map the first action frame to a target. The total distance of any such mapping is of course the distance between Action[i] and Target[j], so the first row of W is the same as the first row of D.

But what about the next row of W? An entry W[1,j] represents the total distance for the best mapping that starts with Action[0] and ends by mapping Action[1] to Target[j]. Clearly, this distance includes the distance between Action[1] and Target[j], so it includes D[1,j]. But is also includes the cost of whatever Action[0] was mapped to. So of course, we pick the best match for Action[0], with one constraint: legal mappings are non-decreasing sequences, so Action[0] can only be mapped to frames in the range Target[0] through Target[j]. In other words:

$$W[1,j] = D[1,j] + Min(W[0,k]), k \leq j$$

More generally,

$$W[i,j] = D[i,j] + Min(W[i-1,k]), k \leq j$$

Note that this is a recurrence relation. The Nth row of W depends on the Nth-1 row, so you have to compute the rows in order. The base case is that the 0^{th} row of W is the same as the 0^{th} row of D.

Once you have computed all of W, the smallest value in the last row of W is the total distance of the best possible mapping between the action and target videos. So print this value (and *nothing else*) to std::cout.

Submitting your program

You will submit your program through Canvas. You should submit a single tar file, and this file should contain all the source files of your program *and a makefile*, but no object or executable files. The makefile should create an executable program called PA5. To grade your programs, the GTAs will write a script that untars your file in a clean directory, runs 'make', and then tests PA5 on novel inputs. If your program does not compile, whether because of an error in your makefile or an error in your code, you will receive no credit on any of the test cases. Note that as always, we will test your code on the department Linux machines.

Grading

As always, most of your grade will be determined by how well your program performs on (novel) test cases. However, a small amount of points are awarded for other factors, such as whether your code compiles. For PA5, for the first time, we will use valgrind to check for memory errors. Programs that leak memory, read uninitialized memory, access out-of-bounds data, or have similar memory problems will loose points.

Hints

1. Do not write any output other than your final answer! *Debugging statements are wrong answers!* You can have debugging statements while you develop your code, but remove them all before submitting!
2. Before submitting, unmake your tar file in an empty directory and test it. I know I have said this before, but some people aren't doing it and they are getting burned. For example, they forget to include a file in their tar file…
3. If you compare a video to itself, the value should be (approximately) zero.
4. If you compare a video to any fragment of itself, the value should be zero.
5. If you compare a video to a fragment of itself from which you delete some frames and replicate others, the result should still be zero.
6. If you implement this algorithm correctly, the complexity it $O(NM^2)$