Redefining Distance

Programming Assignment #6
CS253 Spring 2014
Due 6:00pm Wednesday, March 30th, 2016

Motivation

As the focus of the course changes from memory management to object oriented programming, it makes sense that the programming assignments will change, too. Through the first 5 assignments, you kept building a larger and larger project. You will continue to add to your project, but now you will also start to reach down into it and change parts you have already written. If your code is modular, this won't be too hard. If it isn't…

Also, this assignment is designed to use not just the encapsulation component of objects, but polymorphism as well. This assignment is most easily written using inheritance, virtual dispatch, and functors (i.e. objects for which the parentheses are overloaded). Of course, you can write it however you'd like ☺.

Task

As in PA5, your program will take in two filenames as arguments, and write the distance between the two videos to std::cout. The distance is the time warped distance from the shorter video to the longer video. This time, however, there is also a third command line argument. This argument tells the program what distance measure to use when computing the frame-to-frame distances in the D array. (The W array is then created from the D array.)

In PA3, we defined the distance between two frames as the sum of the Euclidean distances between the corresponding points. (We continued to use this definition in PA4 and PA5.) To be precise, the distance between two frames was:

$$D(P1, P2) = \sum_{i=1}^{25} D(P1[i], P2[i])$$

This is similar to defining the frame-to-frame distance to be the *average* distance between corresponding points, since the difference between the sum and average is a constant multiple. The average distance is defined as:

$$D_{avg}(P1, P2) = {}^{1}/_{25} \sum_{i=1}^{25} D(P1[i], P2[i])$$

There are, of course, other ways to define the frame-to-frame distance. The *median* frame-to-frame distance is median of the point-wise distances:

$$D_{med}(P1, P2) = Median_{1 \le i \le 25}\big(D(P1[i], P2[i])\big)$$

The median, of course, is defined as the middle value after you sort the entries in the set. In this case, it will be the 13th largest value of all the D(P1[i],P2[i]) values.

Another popular distance is the *Linf* norm, which is just the largest of all the point to point distances:

$$D_{Linf}(P1, P2) = Max_{1 \le i \le 25}\big(D(P1[i], P2[i])\big)$$

Finally, you can take the Euclidean (or *L2*) distance of all the point-wise Euclidean distances, which gives you to following distance measure:

$$D_{L2}(P1, P2) = \sqrt{\sum_{i=1}^{25}(D(P1[i], P2[i])^2)}$$

In PA6, your program will take a total of 3 command line arguments. The first two are filenames of videos, as in PA5. The third is one of *avg, med, Linf,* or *L2*. (Note that these are literal strings, and that capitalization matters.) You program, as in PA5, will write the time warped distance to std::cout (and nothing else). This PA6, however, the time warped distance is computed using the frame to frame distance measure signaled by the 3rd command line argument. (If the command line argument is not one of the 4 literals above, throw an error.)

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 PA6. 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 PA6, like PA5, 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. If you create a vector of distance values, you can sort it using the std::sort algorithm. Using this algorithm requires that you do

`#include<algorithm>`. To call std::sort on a vector called 'vec', the command looks like: `std::sort(vec.begin(), vec.end());`

2. Note that we are not changing how videos are normalized.
3. If the 3rd argument is L2, your answer should be 1/25 of your answer in PA5.
4. I strongly suggest writing unit tests. It is getting harder and harder to verify your code using only end-to-end testing.