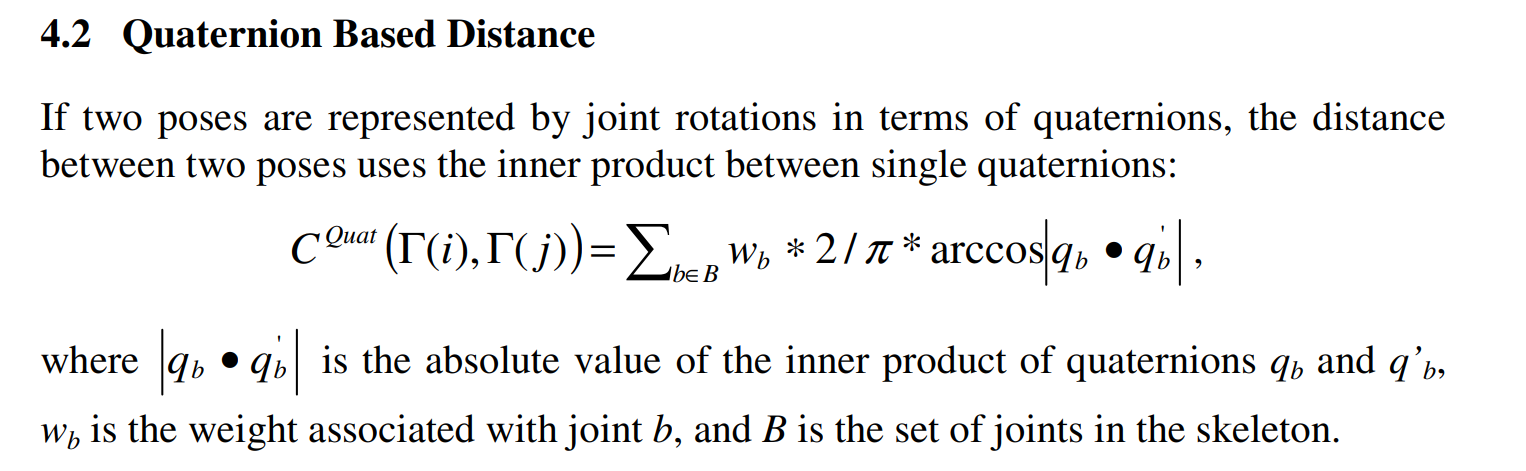
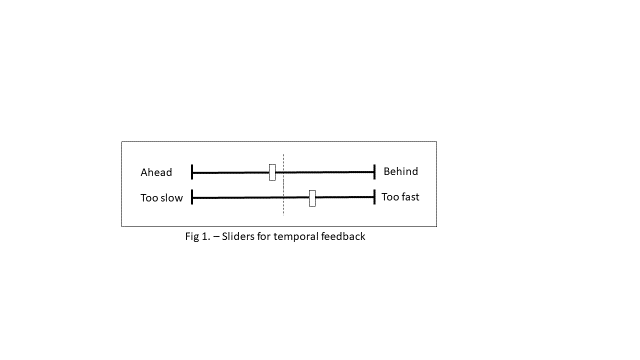
This repository has the following:

1. *Details of my OPW (Online Predictive Warping) approach.*  
   This has been implemented using C++ as a DLL for MotionBuilder so sorry for the all horrible MoBu SDK specific stuff. Also this researcher’s code so it is a bit hacked out.
2. *My python code for measuring the distance between two quaternions.*   
   This is a standard approach to comparing joints, although there are other approaches which I have also implemented in Python. It is based on the approach below as described by (Guerra-Filho and Bhatia, 2011).  
     
   I suggest that we want to measure the difference between the live input motion and reference motion using quaternions and not joint positions in 3D space as using in my current OPW implementation.  
     
   

The OPW approach warps a reference motion in real time to match that of a live input motion. What we need to do in the dance training system is simpler, we just want to measure the temporal distance between the live input and a reference motion (i.e. we don’t need to warp the reference motion). Once the temporal difference is established spatial differences can be found.

There are a couple of different approaches which could be used for this:

1. Simple approach:  
     
   On each update do:
   1. get users current pose
   2. search the current match frame and next few frames of reference motion to find best match, using quaternion based distance.
   3. Save position of reference frame with best match to be used as match frame for next call of update function.
   4. Measure the difference between time position of the reference frame and the time since the start of live dance.
   5. Update UI to show temporal deviation.
      1. Suggest a two slider system one showing the positional error and on showing the velocity error based on the positional differential.  
         The slider puck could change colour, e.g. go red the further away it is from the centre line.  
         
   6. Measure the spatial different between the live pose just captured and match reference frame.
   7. Update UI to show special difference
      1. This could be an overly of the user position over where they should be. This could also highlight the limbs which are the furthest out to help focus the user or just colour code the limbs based on the distance from reference pose.
2. Warping approach
   1. The approach above wouldn’t give you sub frame accuracy in the temporal alignment.
   2. We could continually warp the reference motion using a continually updated spline curve as function of time.
      1. Spline curve is shrunk in y if input motion is slow to map x to a frame behind the current playback position
      2. Spline curve is expanded in y if input motion is fast to map x to a frame ahead of the current playback position
   3. To get temporal alignment, you measure the warped position reference frame at the same time position as the current.
3. DTW window
   1. Measure the best match frame using DTW on a small window, comparing the frames ahead of the current matched frame in the reference motion with predicted frames in the input motion. This would pick a best match based on matching a window of frames not a single frame. I have a DTW algorithm in Python that could be converted to C++, based on the algorithm below from Wikipedia:
4. int DTWDistance(s: array [1..n], t: array [1..m]) {
5. DTW := array [0..n, 0..m]
7. for i := 1 to n
8. for j := 1 to m
9. DTW[i, j] := infinity
10. DTW[0, 0] := 0
12. for i := 1 to n
13. for j := 1 to m
14. cost := d(s[i], t[j])
15. DTW[i, j] := cost + minimum(DTW[i-1, j ], // insertion
16. DTW[i , j-1], // deletion
17. DTW[i-1, j-1]) // match
19. return DTW[n, m]
20. }

**References:**

Guerra-Filho, G. and Bhatia, H. 2011. A Comparison and Evaluation of Motion Indexing Techniques, Motion in Games. Springer.