Salient Pose: Quick Notes

Nicolas Nghiem

May 2019

1 Salient Pose: overview

1.1 Motivation

The goal here is to perform keyframe reduction on Mocap (but not necessarily Mocap, the good point of this algorithm is that it can be used more widely) datas. Two key objectives here:

- Manipulation. It can be tedious to edit Mocap data since all the frames are keyed: manipulating one frame does not affect at all its neighbouring frames so the artist has to work on each frames separately. In practice, there still are some tools that let have multi-frames manipulations via working in the graph editor but these are really basic and not particularly efficient (but artists can still do the work with it) SIGGRAPH Asia 2018 Technical Brief "A Magic Wand for Motion Capture Editing and Edit Propagation" by Christopher Dean and J.P. Lewis proposed a technique for better manipulation without working in the graph editor.

- Compression. Keyframing seems to be a good solution to compress the movement. First, reducing drastically the number of keyframes can help having a lighter impact on the memory costs since it stores way less informations (it is possible to divide by 5 to 10 the number of frames without significant losses in the animation). But, an important aspect of the keyframes is that they correspond to poses that are meaningful to understand the movement. Being able to compute those keyframes will be helpful on a technical point of view as it means getting the most important features of the movement (if it is the key poses for humans to understand the movement it should be the same for computers) leading to extract information, classify movements. On an artistic point of view, it can also be helpful, as it can become a tool for beginners to analyze a movement and understand the animation principles and the way the movement sets up.

TL;DR: Keyframe reduction is good for easier manipulation and most importantly compression of the movement (which will may be used later for classification or information extracting tasks)

1.2 Notations

We represent the animation of a joint or a set of joint starting at frame 1 and ending at frame N_f, $\mathcal{A}_{(1,N_f)}$ as an element of $\mathcal{K}^{N_f}$ where the data of a frame is an element of $\mathcal{K}$ (for example, if we represent a joint by its position in World Space and that we only consider one joint, $\mathcal{K} = \mathbb{R}^3$). Note that we can always translate the problem so that we can start at frame 1 without loss of generality. We call $\mathcal{S}$ the set of all possible keyframe choices which is basically the set of all subsets of $\{1, ..., N_f\}$. $\mathcal{S}_k$ is the set of the selections of exactly k keyframes so it is the subset of the elements of cardinal k of $\mathcal{S}$.
We assume (we will discuss this function later on) that we have a value function $\mathcal{V} : \mathcal{S} \times \mathcal{K}^{N_f} \to \mathbb{R}$ that calculates the value -error or reward depending on the design- of a selection of keyframes for a given animation.
We call $\mathcal{S}_{k,n,\mathcal{A},\mathcal{V}}$ the optimal selection of k keyframes of the input animation cropped at frame n, with regards to our value function. To simplify the notations, we omit the $\mathcal{A}$ and $\mathcal{V}$ from now on. Our goal is to calculate $\mathcal{S}_{k,N_f}$ for all $k \in \{2, ..., n\}$. Note that we start at 2 since we always have to choose the first and last keyframes.

TL;DR:

- $\mathcal{A}_{(1,N_f)}$: animation starting at frame 1 and finishing at frame N_f
- $\mathcal{V}(S, \mathcal{A})$ is the value (error or reward) of the selection of keyframes S for the animation A
- $\mathcal{S}_{k,n}$ the optimal selection of k keyframes of the input animation cropped at frame n, with regards to our value function.
- INPUTS: $\mathcal{A}$ and $\mathcal{V}$
- OUTPUT: $\mathcal{S}_{k,N_f}$ (but also all possible $\mathcal{S}_{k,n}$)

1.3 Understanding the algorithm

Basically, what we do is seeing the animation as an oriented graph where each frame corresponds to one node. See Figure 1. Playing the animation normally boils down to take the path that goes through all nodes. A k-keyframe reduction consists of choosing a path that only goes through k nodes. The algorithm consists of two steps: first giving values to all the edges of the graph, then finding the best k-path with a dynamic programming approach. Let's explain briefly the idea behind the DP algorithm.
We solve the problem by a dynamic programming approach. The key aspect of a dynamic programming approach is to decompose the algorithm into subproblems which are easier to solve. In our case we will calculate all $\mathcal{S}_{k,n}$ by iterating on n. The main idea is that if, for a given endframe e we know all $\mathcal{S}_{k,e-1}$ for $k \in \{2, ..., e-1\}$ then we claim that we know how to calculate $\mathcal{S}_{k',e}$ for $k' \in \{2, ..., e\}$.
Initialization is easy as $S_{2,2} = \{1, 2\}$. Let's now assume that we know all $\mathcal{S}_{k,n}$ for $n \in \{2, ..., e-1\}$. Let k' be in $\{2, ..., e\}$. We want to calculate $\mathcal{S}_{k',e}$ which looks like: $\{f_1, f_2, ..., f_{k'-1}, f_{k'}\}$ where $f_1 = 1$ and $f_{k'} = e$. If we fix $f_{k'-1}$, we know exactly how to choose $f_2, ..., f_{k'-2}$ to

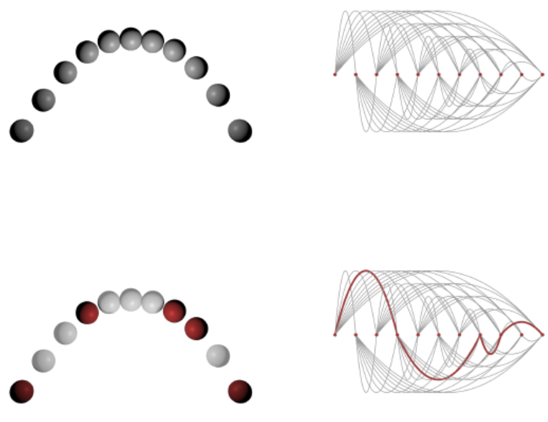

Figure 1: The balls colored in red corresponds to the keyframes chosen for reduction

optimize the value function as it is exactly the definition of $\mathcal{S}_{k'-1,f_{k'-1}}$ -that we know how to calculate according to our assumption. Then we only need to iterate on all possible $f_{k'-1}$ to see which one optimizes the value $\mathcal{V}(\mathcal{S}_{k'-1,f_{k'-1}} \cup \{e\}, \mathcal{A})$. Let $f^*_{k'-1}$ be this value, then $\mathcal{S}_{k',e} = \mathcal{S}_{k'-1,f^*_{k'-1}} \cup \{e\}$ which validates our claim.

TL;DR: Imagine that $S_{3,k} = \{1, 2, k\}$ for all k between 3 and 5 (the optimal keyframe that can be selected for the animation that starts at 1 and finish at k is 2). Then, if we want to find $S_{4,6}$ we do not have to actually try all combinations of 4 keyframes possible (1 and 6 being always chosen), because of previously, we know that 1,2 and 6 are always chosen. We can prove easily that if we can calculate all $\mathcal{S}_{k,n}$ for $n \leq e - 1$ then we can compute $\mathcal{S}_{k,e}$ easily and efficiently.

1.4 Pseudo-code for the algorithm

The pseudocode for applying the algorithm is really quick, everything is condensed in a few lines. We first initialize $\mathcal{S}_{k,e}$ for $k = 2$ and $e = 2$. Then for all e we deduce $\mathcal{S}_{k,e}$ using the previous $\mathcal{S}_{k,n}$ for $n < e$

Algorithm 1 Salient Pose: Pseudo-code

for e in range(2,N_f) **do**
  if $e == 2$ **then**
    $\mathcal{S}_{2,2} = \{1, 2\}$
  else if $e > 2$ **then**
    $\mathcal{S}_{2,e} = \{1, e\}$
    $\mathcal{S}_{e,e} = \{1, ..., e\}$
    for k in range(2,e) **do**
      $j = \text{argmin}_{j'}(\mathcal{V}(\mathcal{S}_{k-1,j'} \cup \{e\}, \mathcal{A}))$
      $\mathcal{S}_{k,e} = \mathcal{S}_{k-1,j} \cup \{e\}$
return $\mathcal{S}_{k,N_f}$

1.5 Design of the value function

We did not cover how to compute the value function which is a critical step of the algorithm. The main assumption of the paper is that "the keyframes are the frames that allow to interpolate well the movement". Following this idea, from a given set of keyframes, we create an animation by interpolating linearly between these keyframes. We then compute the error by computing the maximum Euclidean distance between the corresponding (in terms of time) points of the two animations. If we call A_t, I_t the values at time t of the animation and the interpolation respectively, we can write:

$$\mathcal{V}(\mathcal{S}, \mathcal{A}) = max_t \, \|I_t - A_t\|^2$$

Question can be made about this choice, why choosing a linear interpolation rather than a spline or an as-close-as-possible interpolation? First for computation reason. Secondly, the author claims in the paper that the linear interpolation is the best at capturing extremes which are most susceptible to be good keyframe choices.

1.6 Implementation of the algorithm

The author provides an implementation in Maya using Python, C++ and OpenCL for parallelizing and accelerating the computations. But since I wanted to custom the algorithm and study its output, I provided my own implementation of it using only Python so that I could control totally the data structure used to fit my need. In order to still get decent performances, I chose to use numpy and represented everything I could with numpy arrays (using numpy operations for performance). Furthermore, to decrease the computation cost, I decided to do a pre-computation to store the value of the error. Indeed, computing the error of the selection $\mathcal{V}(\{f_1, f_2\})$ requires calculating a norm $f_2 - f_1$ times but storing this value can be done and is actually helpful even for selections of more than 2 keyframes using the fact that:

$$\mathcal{V}(\{f_1, f_2, f_3\}) = max(\mathcal{V}(\{f_1, f_2\}), \mathcal{V}(\{f_2, f_3\}))$$