Conversation Animation for Virtual Humans

IdleBehaviour with AsapRealizer

An implementation with Motion Graphs

Vanessa da Silva, Yannick Bröker

17. August 2015

Contents

1 How to run

1.1 gesturebinding

In the gesturebindung, the RestPose needs to be specified:

```
<RestPoseSpec>
    <constraints>
        <constraint name="stance" value="STANDING"/>
        <constraint name="BODY" value="IDLE"/>
    </constraints>
    <RestPose type="class" class="asap.realizerdemo.idle.IdleMovement"/>
</RestPoseSpec>
```

1.2 BML

After loading the gesturebinding in the animation-engine-configuration, the Idle-Movement can be started with the following BML:

```
<bml id="bml1" xmlns="http://www.bml-initiative.org/bml/bml-1.0">
    <postureShift id="pose1" start="0">
        <stance type="STANDING"/>
        <pose part="BODY" lexeme="IDLE"/>
    </postureShift>
</bml>
```

2 Description of used techniques

2.1 MotionGraphs

We initialize the MotionGraph with an List of captured motions, given as SkeletonInterpolators, so at start the Graph looks like in figure 1.

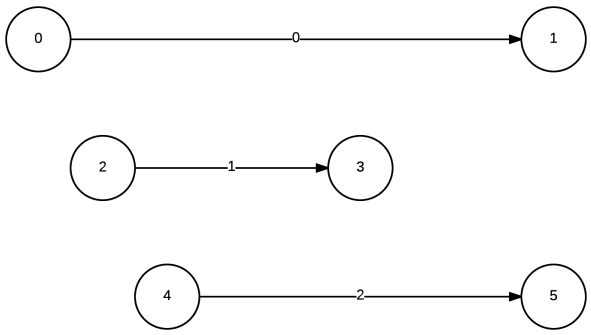

Figure 1: Graph at start

We tried to also add mirrored motions, but we discovered an error in SkeletonInterpolator.mirror, so we made it optional. It can be triggered with MotionGraph.MIRRORED, also, if you want only mirrored motions, you can disable the normal motions with Motiongraph.NORMAL.
After loading the graph we split motions to gain more nodes in the graph. Currently, we just split motions in equally long pieces. A better way could be to detect moments in the motion without any movement and cut them there.
figure 2 shows the graph after splitting.

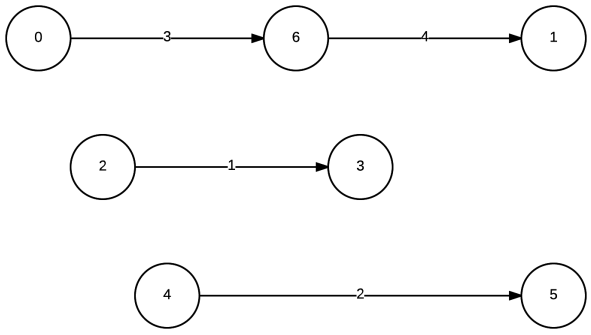

Figure 2: Graph after splitting

After splitting, we use a Distance Metric as described at section 2.2 on the following page to find motions, which are equally enough for Blending.
To blend to motions, we cut both in half, where the end-piece of the first and the start-piece of the second motion are the parts used for blending and so of the same length. The blended motion which is created like described in section 2.3 on page 6 then connects the start-piece of first and the end-piece of the second motion.

Currently, we compare and blend 100 frames of the motions. We tried some different values, but they didn't led to better results for every motion, so we remain at 100 frames.
Maybe it's better to change the number of blended frames based on the distance, but for our project a fixed value leads to good-enough results.

In figure 3 on the following page, motion 3 should be blended with motion 1, so both are cut in half, 3 in 5 and 6, 1 in 7 and 8. Then, motion 6 and motion 7 are blended to create motion 9, wich now connects motion 5 and 8.
The connected motions 5,9 and 8 are now a smooth transition from node 0 to 3.
figure 4 on the next page shows the graph, after all possible blendings are created.
It contains already infinite-length motions, which are concatenations of single motions, but also Dead-Ends, nodes which

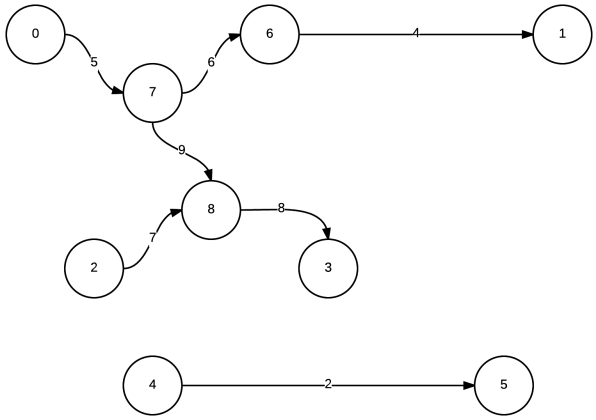

Figure 3: Graph after first blending

doesn't have any successor. If one of these are reached in a random-walk, the random-walk stops, so each of them needs to be removed from the graph.

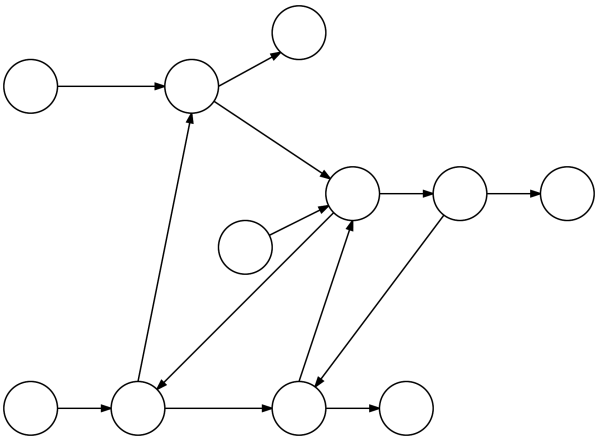

Figure 4: Graph after all blendings

After pruning, the Graph looks like in figure 5 on the following page.
Since all Dead-Ends are removed, there's always a next-motion in a random-walk.

2.2 Distance Metrics

As default Distance-Metric we use a Joint-Angles-Metric based on the one described in [vanbasten2009, 4.1 Joint angles]. Because velocity isn't very relevant in idle-motions, we just compared the root-position and joint-angles.

$$d(a,b) = (p_a - p_b)^2 + \sum_{k \in J} w_k (\log(q_{a,k}) - \log(q_{b,k}))^2 \tag{1}$$

Also, the weights we use are different than described in the paper.
The Legs are often stand at the same position in one motion, but in different positions in other motions. If we blend two different leg-postitons, the feet slide from one position to the other, which isn't very natural.
Others, like hands and arms can be blended from every position, and it's looking natural.
Based on this, we set the weights for hip, knee and ankle very high, and all other weights low.

If the Distance is lower than 20, (see MotionGraph.DEFAULT_THRESHOLD), the Motions are 'near' enough to be blended. We tried some thresholds, but they didn't led to better results, so we remain with this.
Also, the threshold is strongly connected with the weights, so if you change them for different results, you only need to change thresholds or weights.

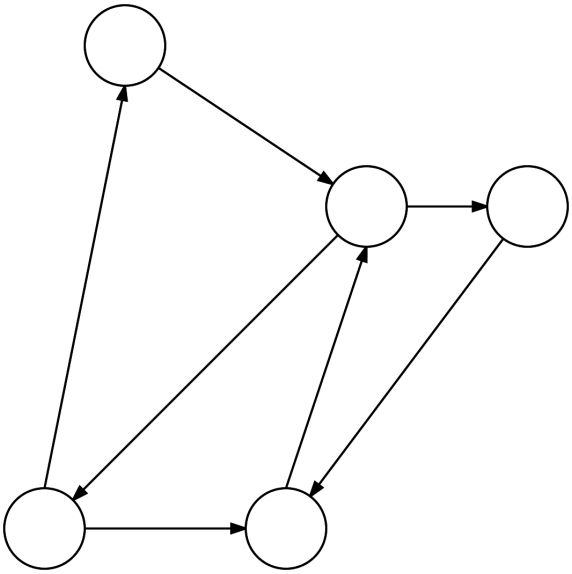

Figure 5: Graph after pruning

In later versions, it's maybe better for understanding and other metrics to set the threshold to 1 and set the weights accordingly.
See also section 4.1.6 on page 8.

2.3 Blending

We use the simple blending as described in [kovar2012, 3.3 Creating Transitions] instead of the one described in [kovar2003]. The second one may lead to better results, but is much more complicated and the first one already produces good-enough results. Blending with registration curves as in [kovar2003] is much more useful with Motion Graphs that contain much movement, like walking, jumping etc, which is not the case in idle-movement. Maybe, it can be implemented in a later version to see differences between the two blending-technique.

We calculate the resulting frame as follows:
The root-position is just a linear transition between the two original positions:

$$p_i = \alpha * p_{a_i} + 1 - \alpha * p_{b_i} \tag{2}$$

Where $p_a$and p_b are the root-positions of the blended frames, i ist the index of the blended frame, and p is the resulting position.

The rotation if each joint is calculated with the slerp-algorithm:

$$r_{j_i} = \text{slerp}(r_{a_{j_i}}, r_{b_{j_i}}, 1 - \alpha) \tag{3}$$

Where $r_{a_{j_i}}$ and $r_{a_{j_i}}$ are the rotations of joint j at frame i, and r_{j_i} ist the resulting rotation.
(We need to use $1 - \alpha$ in slerp, some other implementations (e.g. the papers) uses α.)

α ist calculated with the following formula, which results in better motions than just a linear transition from 0 to 1.

$$\alpha = 2 * \left(\frac{i+1}{k}\right)^3 - 3 * \left(\frac{i+1}{k}\right)^2 + 1 \tag{4}$$

Where i is the frames index as in (2) and (3), and k ist the number of frames that are to be blended.

2.4 Alignment

Before calculating the distance between motions and blending them, they need to be aligned.
Not all motions have the same root-position and rotation, so unaligned blending would result in unnatural sliding, for the

blended root would slide between the two original ones.
To avoid this, we set the root-position of the first frame of both blended pieces of the original motions to the same position. Also, we rotate the model around the Y-axis, so both frames 'look' at the same direction. Rotation around X and Z doesn't need to be aligned, because these are part of the motion.

3 Extensibility

3.1 Load mocap files

Currently, other motion-capture-files could only be defined in IdleMovement.java. Sadly, after this, the whole package need to be recompiled, so we hope theres a better way in a later Version of AsapRealizer.
In the version of AsapRealizer we use, the only way to set different files without recompiling are the parameters in the BML, but those parameters are set while running the RestPose, so the MotionGraph would be calculated when the RestPose is started, which leds to a delay of a few seconds.

3.2 Other Classes for Blending etc.

The MotionGraph is constructed to be used with other classes for Metrics, like PointCloudMetric as described in [vanbasten2009] rather than JointAnglesMetric we use, Alignment etc. We use a Builder (MotionGraph.Builder) to set those classes and create an instance of the graph.

Setting different classes is only possible in IdleMovement.java, which leads to the same problems as loading other mocap files.

4 System overview

4.1 Class structure

4.1.1 IdleMovement

The RestPose, which could be started with BML.
It uses the MotionGraph to generate real-looking, infinite Motions for idle-behavior.

If it's started, it gets the motion which should be played with MotionGraph.next(), and align position, rotation and time

Currently, if the idle-behaviour ist running, it overrides every other behaviour which ist started. This could be fixed in a later version, maybe a Bcs-thesis.

4.1.2 IMotionGraph

Interface for MotionGraphs. Its only Method ist next(), which returns the next Motion (as SkeletonInterpolator) in the MotionGraph.

4.1.3 MotionGraph

Our MotionGraph-Implementation.
It's described in section 2.1 on page 4.

4.1.4 MotionGraphBuilder

Our Builder for the MotionGraph. It's implemented to easily construct a MotionGraph with different implementations for each aspect.
It's not necessary, but was usefull for changing different Parts of the MotionGraph.

4.1.5 IDistance

Interface for DistanceMetrics.

4.1.6 JointAngles

Our implementation of an DistanceMetric. It's based on the Joint-Angle-Method as described in [vanbasten2009, 4.1 Joint angles] with few changed. We only compare root-position and joint angles, and ignore the velocities. Thats mainly because joint velocities are very low in idle motions and if they are also low-weighted, the tend to be near 0.

The weights we use are stored in WeightMap, which is an implementation of a Map. It holds weights for each joint, and ignores the L_- and R_-prefixes of joint-names.

Right now, the weights for arms are very low, and the weights for legs are high.
The legs are high, because if they are in different Positions while blend, the slide above the floor. Same doesn't count for arms, since blending of different Positions only leads to new Motions.
See also section 2.2 on page 5

4.1.7 IBlend

Interface for Blendings

4.1.8 Blend

Our Implementation of a blending, as described in [kovar2012, 3.3 Creating Transitions], see also section 2.3 on page 6 for an explanation.

4.1.9 IAlignment

Interface for Alignment.

4.1.10 Alignment

Our implementation of an Alignment, see section 2.4 on page 6 for an explanation.

4.1.11 NopAlignment

It's an implementation which doesn't do anything, but was usefull for testing.

4.1.12 ISplit

Interface for Splitting.

We think, a good implementation could split long, captured motions, which contains different, unrelated motions, in pieces, which contains only one motion.

4.1.13 DefaultSplit

Our Implementation currently splits Motions in pieces with lenght around 2.5 seconds.

5 References

[kovar2003] Lucas Kovar and Michael Gleicher. Flexible Automatic Motion Blending with Registration Curves, 2003.

[kovar2012] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion Graphs, 2012.

[vanbasten2009] B.J.H. van Basten and A. Egges. Evaluating distance metrics for animation blending, 2009.