IdleBehaviour with AsapRealizer

An implementation with Motion Graphs

Conversation Animation for Virtual Humans

Vanessa da Silva, Yannick Bröker

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 Used Techniques

2.1 MotionGraphs

We initialize the MotionGraph with a List of captured motions, given as SkeletonInterpolators, where nodes represent the start and end of a motion while the edges represent the motion itself.(cf. figure 1.)

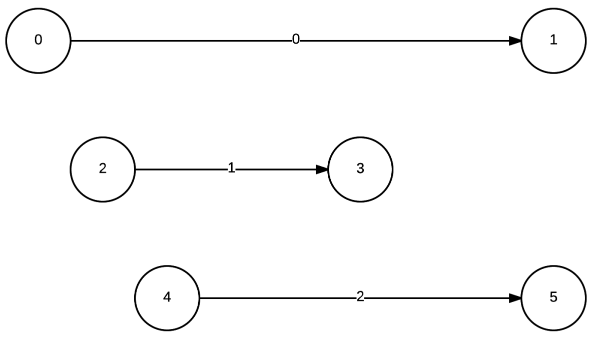


Figure 1: initial Graph

When trying to add mirrored motions, we discovered an error in `SkeletonInterpolator.mirror()`, so we made it optional. It can be triggered with `MotionGraph.MIRRORED`. If you only want the mirrored motions, you can disable the normal motions with `MotionGraph.NORMAL`.

After loading the graph we split the existing motions to gain more nodes in the graph. Currently, we just split motions in equally long pieces. A better way could be to detect frames without any movement and cut them there.
figure 2 displays the graph after splitting.

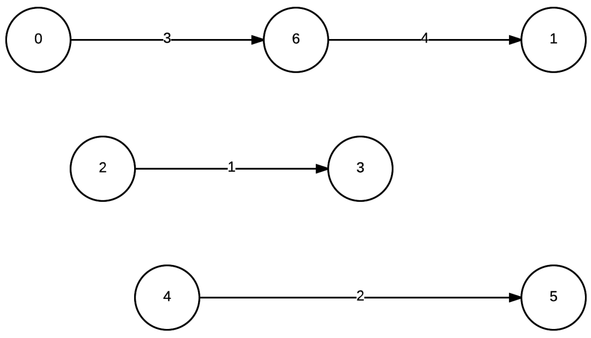
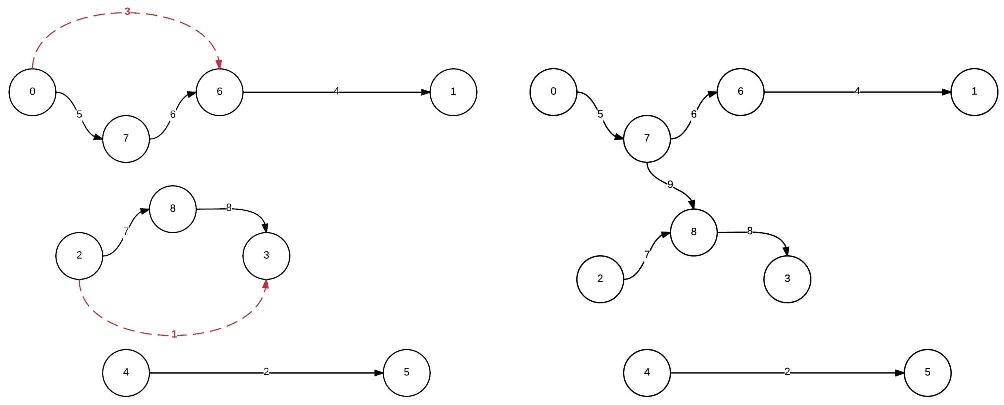


Figure 2: Graph after splitting

After splitting, we use a Distance Metric (as described in section 2.2 on the following page) to find motions, that are equal enough for blending.
To blend two motions, we cut both of them into two pieces, where the end-piece of the first and the start-piece of the second motion are the parts used for blending. In this process two new nodes are created which mark the cuts.The blended motion - created like described in section 2.3 on page 6 - then connects those new nodes.

Currently, we compare and blend 100 frames per motion. We tried multiple values, but they didn't lead to better results in general, so we remain at 100 frames.
It may be better to change the number of blended frames based on the distance, but that would have exceeded this projects scope.
It may become an option when dealing with our Bachelor Thesis.

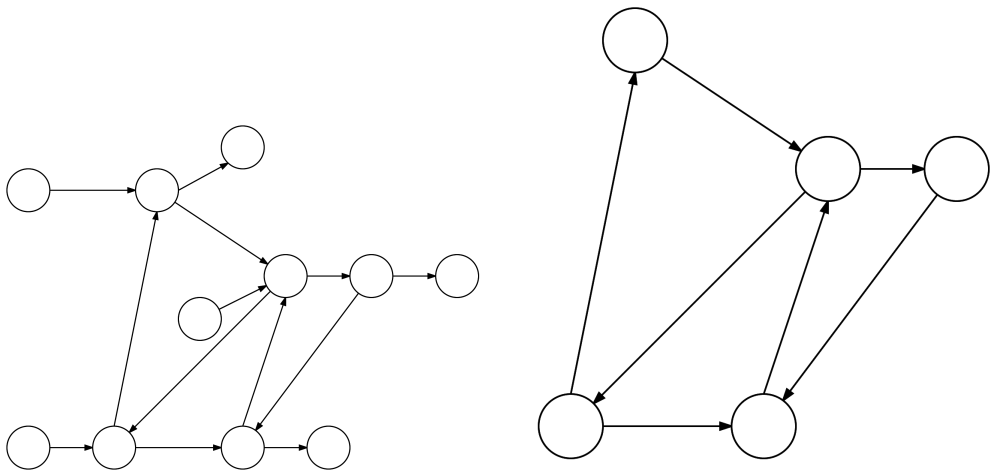
In figure 3, motion 3 shall be blended with motion 1. Therefore both are cut into pieces to create new nodes to mark the beginning and the end of the blended motion. (cf. figure 3a). Then, motion 6 and motion 7 are blended to create motion 9, which now connects motion 5 and 8.
The connected motions 5,9 and 8 are now a smooth transition from node 0 to 3. (cf. figure 3b)



(a) Graph after cutting for blends (b) Graph after blending

Figure 3: Process of blending

figure 4a shows an exemplary Motion Graph, after all possible blendings have been created.
It already contains endless motions, which are concatenations of single motions, but also Dead-Ends - nodes that don't have any successors. Reaching one of these during a random-walk would bring the motion to an end. Therefore such dead-ends have to be removed from the graph.



(a) Graph after blending (b) Graph after pruning

Figure 4: Pruning

After pruning, there is at least one following motion for every node in the graph. This guarantees an endless random-walk. (cf. figure 4b)

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 only compare root-positions 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.

Since legs seldom move very much during idle movement, but their absolute position may vary between different motions, blending can result in the feet sliding over the floor. This is unnatural-looking behavior.
Other limbs, like hands and arms can be blended from every position, without looking unnatural.
Based on this, we chose the weights for hips, knees and ankles very high, and all other weights low.

If the Distance is lower than 20, (see `MotionGraph.DEFAULT_THRESHOLD`), the Motions are 'close' enough to be blended. We experimented with different thresholds, but decided to stick to this one.
Because the threshold is strongly connected to the weights, only one of those values has to be changed, whenever different results shall be achieved.
In later versions, it may be better to change the threshold to 1 and set the weights accordingly, for better understanding or using other metrics.
See section 4.6 on the next page for more information on the current metric.

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 while the first one already serves it's purpose. Blending with registration curves as in [kovar2003] is 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-techniques.

The blending is calculated 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 of 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) use 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 bring the root-positions of the first frame of both motions in line with each other. We then rotate the model around the Y-axis, so both frames 'look' into 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 Loading mocap files

Currently, other motion-capture-files could only be defined in IdleMovement.java. Sadly, after this, the whole package needs 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 leads to a delay of a few seconds.

3.2 Other Classes for Blending etc.

The Motion Graph is constructed to be used with other classes for metrics, e.g. PointCloudMetric as described in [vanbasten2009], different 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 results to the same problems as loading other mocap files.

4 System overview

4.1 IdleMovement

The RestPose, which can 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 aligns position, rotation and time

Currently, if the idle-behaviour ist running, it overrides every other behaviour that may be started afterwards. This could be fixed in a later version, maybe a Bachelor Thesis.

4.2 IMotionGraph

Interface for Motion Graphs. Its only Method is `next()`, which returns a following Motion (as SkeletonInterpolator) in the MotionGraph.

4.3 MotionGraph

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

4.4 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.

4.5 IDistance

Interface for DistanceMetrics.

4.6 JointAngles

Our implementation of a DistanceMetric. It's based on the Joint-Angle-Method as described in [vanbasten2009, 4.1 Joint angles] with few changes. We only compare root-positions 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, they tend to be close to 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.
See section 2.2 on page 5 for more information.

4.7 IBlend

Interface for Blendings

4.8 Blend

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

4.9 IAlignment

Interface for Alignment.

4.10 Alignment

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

4.11 NopAlignment

It's an implementation that doesn't align anything at all, but was usefull for testing.

4.12 ISplit

Interface for Splitting.

We think, a good implementation could split long, captured motions, that contain different unrelated motions, into pieces, which contain only one motion.

4.13 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.