Motion Data and Model Management for Applied Statistical Motion Synthesis

E. Herrmann[1,2] (ORCID), H. Du[1,2], A. Antakli[1,2], D. Rubinstein[2], R. Schubotz[2],
J. Sprenger[1,2], S. Hosseini[1,2], N. Cheema[1,2,3], I. Zinnikus[2], M. Manns[4], K. Fischer[2] and P. Slusallek[1,2]

[1]Saarland University, Germany
[2] German Research Center for Artificial Intelligence (DFKI), Germany
[3] Max Planck Institute For Informatics, Germany
[4] University of Siegen, Germany

Abstract

Machine learning based motion modelling methods such as statistical modelling require a large amount of input data. In practice, the management of the data can become a problem in itself for artists who want to control the quality of the motion models. As a solution to this problem, we present a motion data and model management system and integrate it with a statistical motion modelling pipeline. The system is based on a data storage server with a REST interface that enables the efficient storage of different versions of motion data and models. The database system is combined with a motion preprocessing tool that provides functions for batch editing, retargeting and annotation of the data. For the application of the motion models in a game engine, the framework provides a stateful motion synthesis server that can load the models directly from the data storage server. Additionally, the framework makes use of a Kubernetes compute cluster to execute time consuming processes such as the preprocessing and modelling of the data. The system is evaluated in a use case for the simulation of manual assembly workers.

CCS Concepts

• Computing methodologies → Motion capture; Motion processing; • Human-centered computing → Visualization toolkits;

1. Introduction

Machine learning has become a popular method to solve difficult problems based on example data. Specifically, for character animation, machine learning models provide a method to create realistic results based on examples from motion capture data.

However, in practice managing training data for such models can become a tedious and error-prone task. Furthermore, motion data that comes from diverse capture sources has to be merged into a consistent dataset by retargeting to a single skeleton. Therefore, the training of motion models needs a framework for the editing and retargeting of a large amount of data. Existing exchange formats such as BVH often lack important information about the coordinate system of the character which makes it difficult to retarget the motion of the trained model. Additionally, existing motion editing tools are built for editing single motions and editing of multiple files requires the users to write custom scripts.

In this paper we integrate a framework for the management of motion clips with a statistical motion modelling pipeline by Min et al. [MC12]. Compared to existing tools, the framework was developed to provide a streamlined workflow to iteratively improve motion models based on new data from diverse sources. Additionally, it enables the retargeting of entire training data sets to different characters.

The data management system is based on an SQL database at its core that stores data in JSON data structures. The schema-less data format enables easy extension of data with experiment specific entries. The central repository provides an overview of the data and can store statistical models in different versions which simplifies the management of the experiment data and enables easy sharing of the result. Our framework also includes a model deployment server to display motions synthesized from the trained models in a 3D scene for testing with user defined constraints.

We integrate a preprocessing tool with the database that offers functions for manual annotation and retargeting of motions. Furthermore, the tool integrates motion editing functionality to apply small fixes using inverse kinematics. This motion editing functionality also enables enhancement of training data sets using manually edited variations of captured motions. For execution of time expensive processes of statistical motion modelling pipelines such as dynamic time warping or the retargeting of a large collection of files, we have integrated a generic job system based on a Kubernetes cluster system.

We first give a brief overview of related work in Section 2. We provide the background on the statistical motion synthesis method in Section 3. Our data management framework, the preprocessing tool and the model deployment server are described in Section 4.

Implementation details on the compute cluster integration are given in Section 5 and a use case for our framework for the simulation of workers is described in Section 6. Finally, we provide a conclusion and an outlook on future work in Section 7.

2. Related Work

The framework covers different aspects of the workflow of the motion modelling, from data management to deployment, with the goal to simplify the use of motion models in practical applications.

To facilitate practical application of motion models different solutions have been presented in the literature. Shapiro et al. [Sha11] present the SmartBody system which integrates motion synthesis methods based on motion graphs and scattered data interpolation. The ICT Virtual Human Toolkit [HTM*13] provides a complete system for the creation of autonomous human agents and the visualization of the behavior in a 3D engine. Shoulson et. al integrate different motion synthesis methods into a framework for the evaluation of agents [SMKB13]. Gaisbauer et al. [GLA*19] present a system that integrates different motion synthesis approaches into unified controller framework.

Research on motion data storage has been focused mainly on efficient storage of large datasets and the retrieval of similar motion data from unordered data sets. Kapadia et al. [KCT*13] present a motion retrieval approach that allows the definition of arbitrary key features by experts which are then used to construct a novel trie-tree data structure. Similarly, Bernard et al. [BWK*13] present a visual motion data base exploration tool that makes use of k-Means clustering of poses represented as feature vectors containing 3D positions of important joints to generate interactive visualizations at different levels of detail.

The management of training data has only recently been started to be investigated in the context of motion data. Riaz [RKW16] present a relational database system for the management of motion recordings. Mahmood et al. [MGT*19] started an effort to collect motion capture data in a central database and retarget it to a parameterized character mesh model.

Commercial solutions such as Motion Builder [Aut19] provide state of the art tools for motion editing and retargeting and interoperability with different tools. Blade by Vicon [Vic19] is a popular motion data storage solution that integrates functions for retargeting and motion editing. Both solutions, however, do not include a data management solution intended for machine learning. Mixamo [Ado19] is a commercial solution that offers motion editing and retargeting tools integrated with a database of characters and motions. However, they provide only single example clips for each motion type and the tools are intended for traditional animation pipelines. There are also commercial solutions for markerless motion capturing such as iClone [Rea19] that integrate functions to retarget and edit data for use on a game character.

In this paper, we combine a motion database with a motion preprocessing tool and a deployment server into a framework to manage and evaluate human motion data for statistical modelling. In our experience we found it useful for the iterative improvement of statistical motion models based on a dataset that contains multiple examples of each motion type.

3. Statistical Modelling and Synthesis Pipeline

Given a set of motion clips for different actions, we construct a graph of low dimensional statistical motion models, each representing the variation of a motion primitive such as left step or right step for a walk action. New motions can then be generated by evaluating samples from these motion primitive models and concatenating the results. The statistical modelling pipeline we apply was originally presented by Min et al. [MC12]. Our adaption of the method was already presented in an earlier work [AHZ*18] but is summarized here for completeness. Figure 1 gives an overview of the statistical modelling pipeline.

In our framework, motion clips are defined as a sequence of skeleton poses. Each pose consists of one root translation and a set of quaternions each representing the relative orientation of one joint.

In order to project example clips of motion primitives into low dimensional space, we first align them spatially and temporally to a reference clip using dynamic time warping. To be able to constrain specific keyframes during the sampling, we use a manually defined semantic annotation of the time line and perform the dynamic time warping process separately for the segments between keyframes.

We reduce the dimensionality using functional Principal Component Analysis. For this purpose we create a functional representation by fitting cubic B-splines to the aligned skeleton pose sequences and time warping functions. The motion splines can be evaluated by the time spline, which maps from sample time to canonical time.

We then apply Principal Component Analysis on the coefficients of the aligned motion splines and time splines separately to project the motions into a low dimensional space. We apply the Expectation-Maximization algorithm to fit a Gaussian Mixture Model (GMM) to the low dimensional samples. Using this statistical model, new variations of a motion primitive can be sampled and back projected to cubic B-spline parameters using the inverse PCA transform. For more details on the modelling, we refer the readers to previous work by Du et al. [DHM*16].

In order to accelerate the constrained search for a motion, we also construct a space partitioning tree structure on the latent motion space according to Herrmann et al. [HMD*17]. The tree is constructed by applying the k-Means algorithm recursively on random samples of the motion model. Each node in the resulting tree stores the mean latent parameters of its children.

At runtime we perform a search in the k-Means tree to find the optimal parameters of the motion primitive using Equation (1) as objective function.

$$O(s,C) = \sum_{i=0}^{N} \sqrt{(f(M(s)) - C_i)^2} \qquad (1)$$

Here s is the latent motion parameter, C is a set of constraints, N is the number of constraints, f is the forward kinematics function and $M(s)$ is the projection from latent space to motion space. The tree search algorithm evaluates the mean of each child node using the objective function to decide which branch to traverse. To avoid ending up in a local minimum, parallel candidates traverse the tree
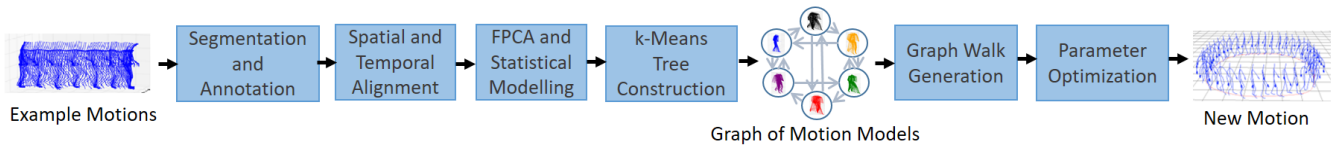
Figure 1: *Overview of the statistical motion synthesis method based on Min et al. [MC12] that is used in our framework. We use functional PCA (FPCA) for dimensionality reduction of examples of motion primitives and model the distributions of low dimensional examples as Gaussian Mixture Models (GMM). To accelerate the constrained motion synthesis we also construct k-Means trees as space partitioning data structures on the low dimensional samples of each motion primitive [HMD*17].*

using different branches. The parameters found by the search can be used as initial guess for further numerical optimization [Mar63]. The parameters of multiple motion primitives can be stacked into one vector for this purpose. Equation 2 shows the objective function for numerical optimization.

$$\arg\min_{s_1,\ldots,s_T} \sum_{i=1}^{T} O(s_i, C_i) - ln(pr_i(s_i)) \qquad (2)$$

Here s_i is the latent motion parameter for step i and pr gives the likelihood from the GMM of the motion primitive. C_i are the constraints for step i and O is Equation 1.

The resulting parameters will be back projected into a motion which can be further modified using inverse kinematics to reach constraints outside of the range of the models. To generate smooth transitions between motion primitives, transition constraints on hand and feet are applied and a standard smoothing method is used.

4. Motion Data and Model Management Framework

Our statistical modelling framework consists of three parts. Firstly, a motion preprocessing tool provides functions to edit and annotate motions. The second part is a data storage server with a REST interface, that manages the input and output data of the statistical modelling pipeline. Lastly, a deployment server can deploy models directly from the database. Additionally, a WebGL client can be used to browse and visualize the content of the database. The motion preprocessing tool is integrated with the database to directly edit the data in the database. An overview of the framework is shown in Figure 2.

4.1. Motion Preprocessing Tool

Our framework includes a motion preprocessing tool in order to annotate data, fix errors in the motion data and retarget motions to different character models. The tool provides a simple 3D visualization of motions using either a skeleton or a loaded character mesh. Individual clips can be imported from the standard formats BVH[†] and ASF/AMC[‡] or loaded from the data storage server using a database browser. Loaded motions can be edited and uploaded to the database. A screenshot of the motion database browser of the tool is shown in Figure 3.

[†] https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

[‡] https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html

4.1.1. Motion Editing

We have integrated a motion editing tool into our framework based on the approach presented by Lee and Shin [LS99]. A screenshot of our motion editor is shown in Figure 4.

The user can modify specific frames by applying translation and rotation constraints on specific joints. The pose to reach those constraints is found using inverse kinematics (IK). Instead of a hybrid IK-approach presented by Lee and Shin, we use a simple IK solver based on cyclic coordinate descent [Wri15]. To create a smooth transition, the change is applied on a frame range via a cubic B-spline representation. In addition to the IK-based editing, the tool integrates functions for mirroring, smoothing and trimming of motions. All commands used during the editing of one motion clip can be saved into a JSON format to later be applied on other clips in the database.

4.1.2. Retargeting

In order to retarget motions, we make use of an algorithm similar to the method of Monzani et al. [MBBT00]. We modified the original approach to work without a reference T-pose by first estimating a coordinate system for each joint. For each pose of the source motion, the algorithm estimates the global orientations of the joints of the destination skeleton by aligning the coordinate systems of corresponding joints. For this purpose, the global coordinate system of each joint of the source skeleton is calculated via forward kinematics (FK). The global orientation of a corresponding joint of the destination skeleton is then calculated by first finding the rotation to align the twist vectors and then finding the rotation to align the swing vectors. The global orientations can be brought back into the local coordinate systems of the destination skeleton, by making sure that parent joints are processed before their child joints.

The retargeting method is summarized for a single pose in Algorithm 1. Note that the method can be optimized for performance by precalculating an alignment rotation using two corresponding poses and applying it as a correction to the source pose.

The method takes a source and a destination skeleton as inputs. In addition, it requires a root translation scale factor and a joint list that has to be ordered to ensure that parent joints are retargeted before their child joints. Our internal skeleton definition stores the coordinate system of each joint which can be automatically derived using a heuristic from the zero pose of the skeleton. The twist axis corresponds to the vector to the child joint and the swing axis can be set to the axis defined by vector from the left hip to the right hip. The guess of the swing axis has to be manually corrected for
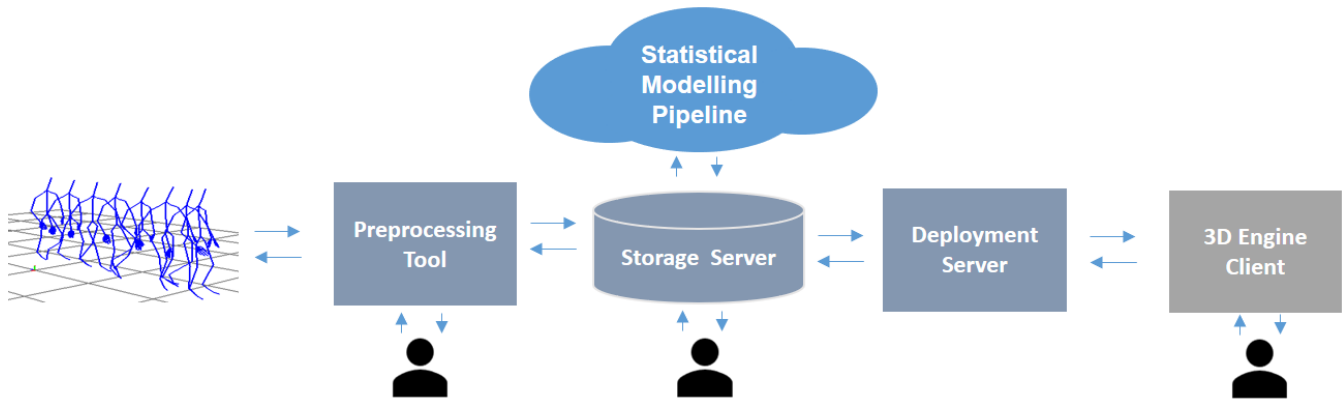
Figure 2: *Overview of the motion modelling framework. The motion data and the models are stored in a central data storage server with a REST interface. The data can be edited using a preprocessing tool. The motion model can be directly evaluated with a character mesh in a WebGL client based on Unity. A collection of models can deployed using a motion synthesis server to animate characters in a 3D application. Time expensive processes of the motion modelling pipeline are executed as jobs on a Kubernetes compute cluster.*
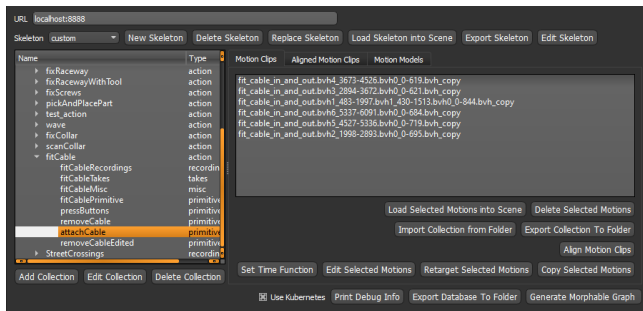

Figure 3: *Screenshot of the database browser integrated in the motion preprocessing tool*

some joints. The refinement process has to be done only once per skeleton and can be performed by loading the skeleton in our tool and editing the skeleton meta data. Screenshots of the skeleton meta data editor are shown in Figure 5. Once the coordinate systems for a skeleton are defined, entire motion collections in the database can be retargeted via the Motion Database Browser shown in Figure 3.

4.1.3. Semantic Time Annotation

The tool also provides a method for manual semantic annotation of the timeline. The semantic annotation is necessary as meta information for the temporal alignment of different motion examples on specific keyframes, such as the time of contact during a picking motion. Additionally, the annotation can be used to extract motion primitives from motion capture takes by splitting the motion clip into separate smaller clips based on the annotation segments.

A screenshot of the semantic timeline annotation tool is shown in Figure 6. The annotation is created by setting the start and end of a segment in the timeline. Additionally, users can move segment boundaries or cut and merge segments to fix mistakes. All functions are mapped to keyboard shortcuts. The trajectory of specific joints can also be plotted to simplify identification of keyframes.
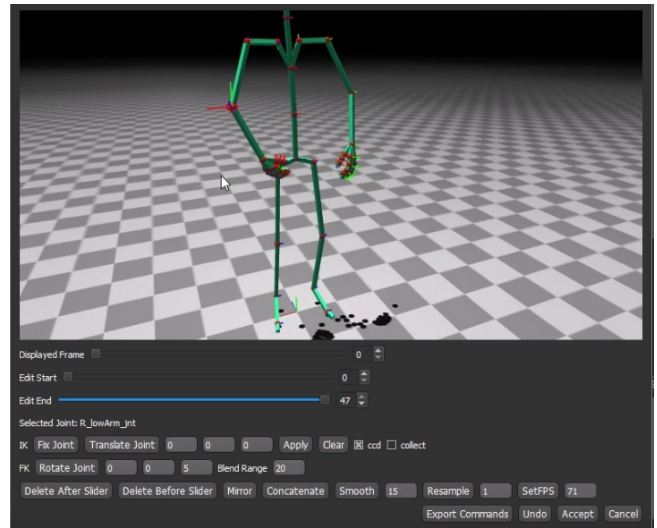
Figure 4: *Screenshot of motion editor tool. The motion can be edited by specifying inverse kinematics constraints or by directly applying rotations on joints. Joints can be selected by clicking on the visualization.*

4.1.4. Copy and Paste

In addition to standard motion editing, the tool also provides the option to fix specific joints in a motion by a copy and paste function. This way broken parameters in a motion can be overwritten with correct parameters from another motion, for example the parameters of a finger during a grasp motion. The frame range that should be copied can be specified. A transition between the modified and the original frames is automatically generated using SLERP [Sho85]. The source motion parameters are automatically stretched if the source frame range is different than the target frame range. A screenshot of the copy and paste tool is shown in Figure 7.
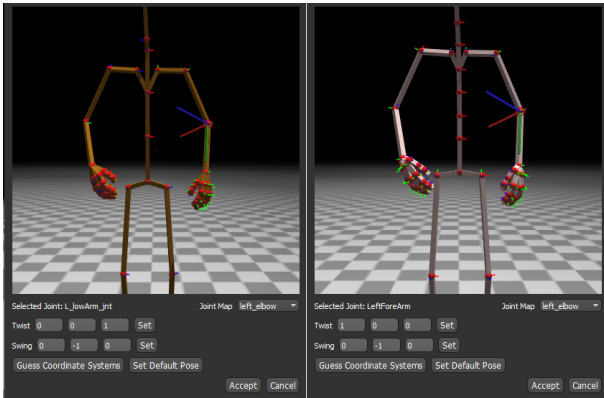

Algorithm 1: Retargeting algorithm

Input : $src\_skel, dst\_skel, src\_pose, scale, joint\_list$

Output: dst_pose pose parameters

1   $dst\_pose \leftarrow zero\_pose()$

2   $dst\_pose.translation \leftarrow src\_pose.translation \times scale$

3   **for** $joint \in joint\_list$ **do**

4     $src\_twist, src\_swing \leftarrow getAxes(src\_skel, joint)$

5     $dst\_twist, dst\_swing \leftarrow getAxes(dst\_skel, joint)$

6     $global\_transform \leftarrow FK(src\_skel, src\_pose, joint)$

7     $global\_twist \leftarrow global\_transform \times src\_twist$

8     $global\_swing \leftarrow global\_transform \times src\_swing$

9     $q_y \leftarrow alignAxis(dst\_twist, global\_twist)$

10    $dst\_swing \leftarrow rotate(dst\_swing, q_y)$

11    $q_x \leftarrow alignAxis(dst\_swing, global\_swing)$

12    $q \leftarrow q_y \times q_x$

13    $p\_joint \leftarrow get\_parent(dst\_skel, joint)$

14   **if** $p\_joint \; != null$ **then**

15      $p\_transform \leftarrow FK(dst\_skel, dst\_pose, p\_joint)$

16      $q \leftarrow inverse(p\_transform) \times q$

17   **end**

18    $dst\_pose \leftarrow replace(dst\_pose, q, joint)$

19 **end**

20 **return** dst_pose

21

Figure 6: *Screenshot of the time line annotation tool. The annotation is edited by defining a start and end point of a segment or by cutting or merging segments at the frame slider. Mistakes can be fixed by moving the boundary of neighboring segments to the frame slider.*
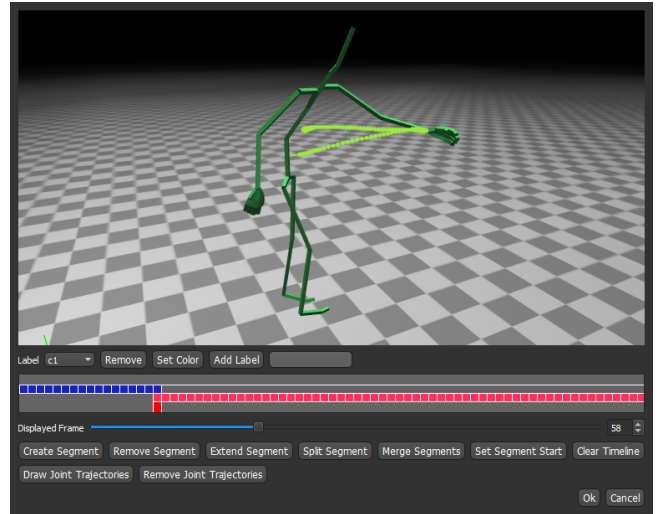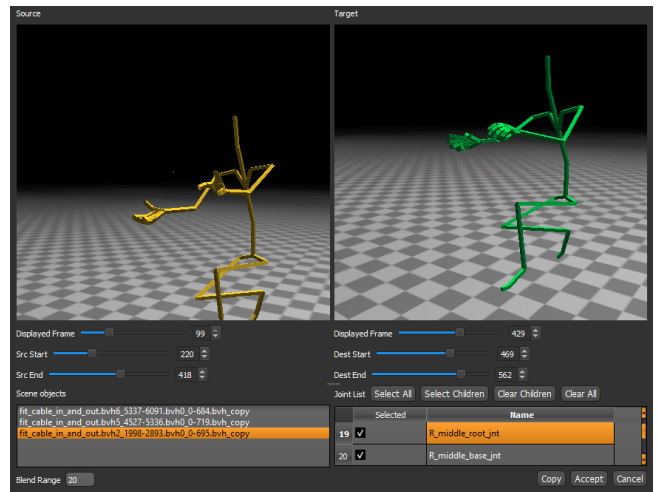

Figure 5: *Two screenshots of the skeleton meta data editor showing the coordinate system of the left elbow for two different skeletons transformed into the global coordinate system based on corresponding poses. The green axis represents the twist axis and the red axis represents the swing axis. The user can edit the local coordinate system of joints by defining unit vectors for the axes and map joints to a standard skeleton. Note that the thin red, green and blue lines at each joint represent the standard X, Y and Z axes rotated by the joint orientation in the global coordinate system.*

4.1.5. Graph Editor

For the deployment of multiple statistical models, we need to define a graph structure with transitions between the motion models. For this purpose, we have added a graph editor shown in Figure 8. The user can define a graph by selecting models from our database as nodes and add transitions between them.


Figure 7: *Screenshot of the joint parameter copy and paste tool. The left view shows the source motion and the right view the target motion. The affected joints can be selected from a list. The source and target frame range can be defined using sliders.*

4.2. Data Storage Server

The data storage server keeps motion and model data in an SQL database and provides a REST interface to edit entries in the database. Motion capture formats such as BVH often have the problem of being difficult to retarget to mesh characters due to missing information on the coordinate system. Additionally, some formats store redundant information on the skeleton in each file which makes it less efficient for storage and retrieval. We took this problem into account when designing the internal data representation to reduce redundancy and add information necessary for retargeting.
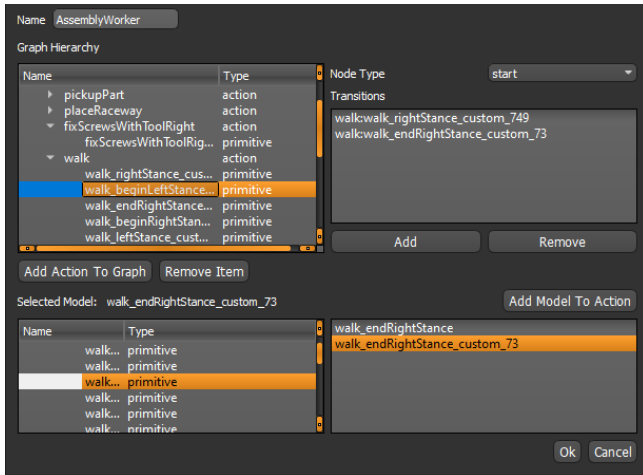
Figure 8: *Screenshot of the graph editor. The graph is constructed by adding models from the database into a hierarchy of nodes and adding transitions between them.*
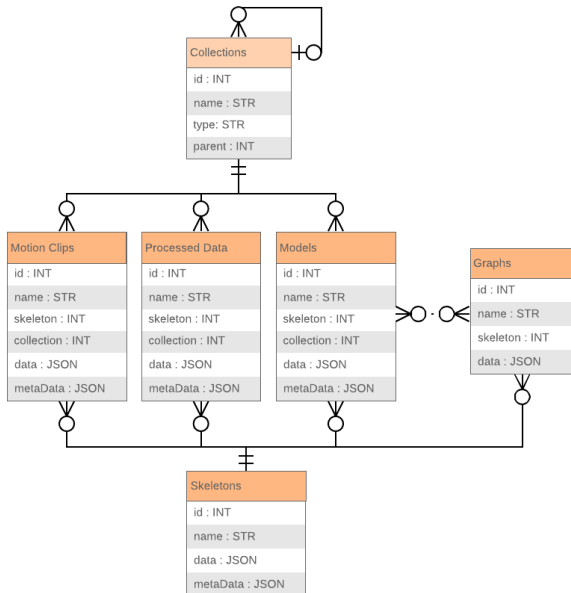
Figure 9: *Schema of the motion and model database. Data is organized based on a hierarchy of collections. All objects are stored in the JSON format in the records of the tables.*

4.2.1. Data Representation

The database contains a motion clip table for the input and a model table for the output of the modelling pipeline. Intermediate results can be stored in a processed data table. The data in those tables is organized using references to a hierarchy of collections. Additionally, an extra skeleton table is used to store information on the skeleton hierarchy data. Separating the skeleton from the motion clips and the model allows reduction of storage size by avoiding redundant information. The definition of graphs of motion models
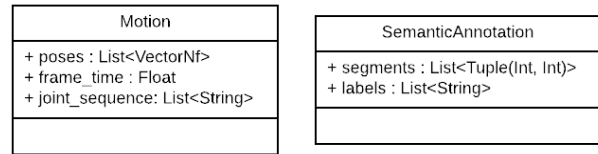
Figure 10: *Internal motion format. Each vector in the pose list contains a root translation and the relative orientation of joints as quaternions ordered according to the joint sequence.*

for deployment are stored in an additional graphs table. The layout of the database is shown in Figure 9. Objects are generally stored in the JSON format[§]. To reduce the required space a binary encoding of JSON can be used [¶].

The motion clips table stores individual clips in a JSON format as shown in Figure 10 that describes a sequence of parameters for a skeleton. A pose consists of a root translation and a sequence of quaternions for the joint rotations, which we concatenate into a single vector. This corresponds also to the input to the preprocessing pipeline.

Optionally, each motion clip entry can store semantic annotations of the time line. The semantic annotations consist of a list of segments, each defined by a name, a start frame and an end frame. The semantic annotation is stored in a meta data column. This allows both columns to be updated separately. To store the intermediate result of the preprocessing pipeline we have added a processed data table with corresponding entries to the motion clips table. In this table we store the result of the dynamic time warping. For each motion clip we store the time function to restore the original motion as meta information. The content of the processed data table is used as input for the statistical modelling step.

The motion model table stores both the statistical motion model and the acceleration data structure. The model structure is shown in Figure 11 and defined by spatial and temporal mean and Eigen vectors, which result from the dimension reduction, and the parameters of the Gaussian mixture distribution, resulting from the statistical modelling. The acceleration data structure is a hierarchy of nodes each storing its children and a mean parameter value. The model is stored in the data column and the acceleration data structure is stored in the meta data column so that they can be updated separately.

Figure 13 shows the custom format for the skeleton that defines a hierarchy of joints stored in the skeleton table. Each joint stores its name, an offset to its parent and a list of its children. To enable retargeting each skeleton table also stores a mapping of joint names to a reference skeleton. To convert the joint rotations between the different coordinate systems, each joint also stores a twist and a swing axis. The information for retargeting is stored in a meta data column.

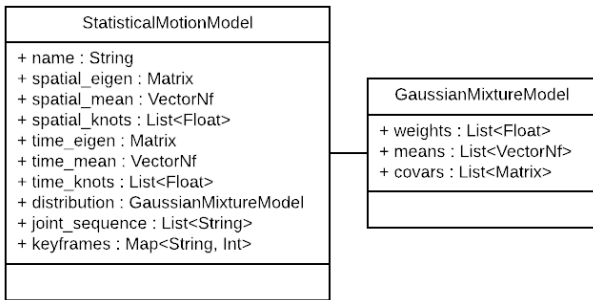The graphs table is used to store graph definitions as shown in

---

[§] https://www.json.org/

[¶] http://bsonspec.org/

Figure 11: *Model format for storing the result of the dimensionality reduction and statistical modelling.*
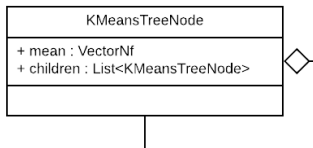

Figure 12: *Format of the acceleration data structure. Each node stores the mean low dimensional parameter of its children.*

Figure 14. Each node in the graph refers to a motion model in the database by its id and stores a list of transitions to other nodes in the graph definition. Additionally, each action stores a list of nodes for the possible start and end states. We group nodes into actions which is useful for actions that are separated into multiple motion models such as walking.

4.2.2. Database Interface

For the editing of the database, the storage server provides a REST interface. The interface offers functions for the insertion, modification and removal of entries in the tables. The interface takes messages in the JSON format that can be easily processed into strings for storage in the database.

The database server also provides a web client that enables users to browse the database and visualize selected motion clips and samples from motion models using different meshes. This way users can immediately inspect how their captured data and the trained motion models appear in a game engine using a character mesh. A screenshot of the database web interface is shown in Figure 15.

To display a motion, the web viewer queries the REST interface of the storage server for the skeleton and motion data described in Section 4.2.1 which is transferred directly using the JSON format.
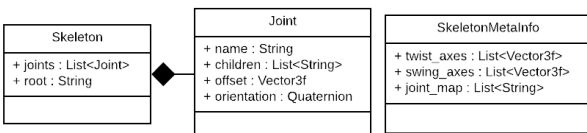

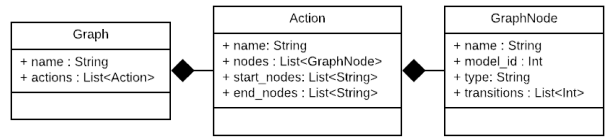**Figure 13:** *Internal skeleton format. The joint map assigns important joints a default joint name.*

Figure 14: *Graph format.*

Figure 15: *Screenshot of the Web-based database browser.*

The web client application converts the retrieved data into transformation matrices that are applied on the characters in the 3D visualization. When the user requests samples from a statistical model, the server converts the low dimensional samples into motion clips before they are send.

New motions can also be uploaded directly using the web interface. Corrections to existing motions can be made using the motion editing tool described in Section 4.1.

4.3. Motion Model Deployment Server

Our framework also includes a stateful motion synthesis server to deploy motion models and enable the interactive control of characters in a 3D scene given constraints, such as a walk trajectory or a pick target. For the visualization of the motion, the server is integrated with a 3D engine such as Unity. The server was already discussed in previous work [AHZ*18], however we extended it to load a graph of motion models directly from the data storage server. Figure 16 shows the interaction between the motion synthesis server and a 3D application.

The server sends individual poses to the client as messages in a JSON format using either a TCP or a WebSocket connection. In each scene update, the current pose containing the root translation and the local orientation of each joint is applied on the skeleton hierarchy of the target character. The pose update message can also include a list of events for scene manipulation. This way, scene objects can be attached to and detached from the character when a pick and place action is performed.

To control the motion, the server expects a message in a JSON format that describes a single action with an optional walk trajectory and an optional list of constraints on joints of the skeleton at specific labeled keyframes. To keep the server interface responsive independent of the synthesis speed and duration of a task, the motion synthesis is executed in a separate thread. This thread generates
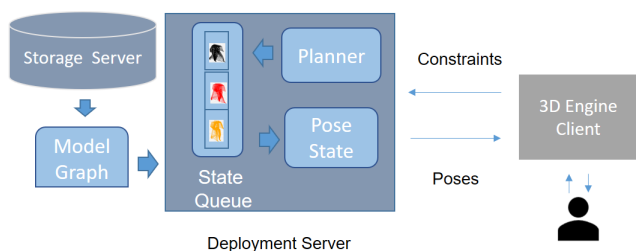
Figure 16: *Interaction between the deployment server and a 3D engine client. In each update the server sends the current pose of the state machine. The state is updated by replaying a queue of motion states. Each motion state represents a sample from a motion primitive model. The queue of motion states is filled by a planner on demand based on constraints from the client. If the queue is empty, the server sends an idle motion.*

motion states that are added to a motion state queue ahead of time. If new input is provided before an action is complete, the buffer is cleared and the synthesis thread is restarted based on the current state and the new constraints. This way, it is possible to react to changes in the scene using a continuous motion, for example in order to avoid a collision with another worker.

When Unity is used as engine, the application can also be compiled to run in the browser using WebGL. The motion synthesis server can then be integrated with the database client during testing of the motion models. For the purpose of testing, the server also provides a REST interface to reload updated motion models from the database.

5. Implementation Details and Compute Cluster Integration

Each component of the system, including the preprocessing tool, data storage server and the deployment server was implemented using Python. The preprocessing tool uses Qt for Python and OpenGL. This makes the framework executable on most desktop and server operating systems. The database itself stores data internally using an SQLite database[||]. This format has the benefit of being easy to backup and copy between operating systems. Additionally, it can be accessed by a standard Python library. The REST interface was implemented using the Tornado library[**]. We use Unity[††] for the display of the motion in the database web client. The data storage server was deployed on a central server running CentOS. We also integrated an instance of the model deployment server with the central data storage server. However, for a practical application it is run on a dedicated server.

Our framework uses a compute cluster based on Kubernetes 1.13.2[‡‡] for the execution of time consuming processes involved in the preprocessing and modelling of motions. For this purpose, the

preprocessing tool integrates the option to run specific functions as jobs on the cluster using the Python API of Kubernetes. We expose the functions from the modelling pipeline that we want to run on the cluster in form of Python scripts with command line interfaces that are stored in a Git repository[§§].

When a job is executed, a Docker image with a Python interpreter is instantiated as a Kubernetes Pod[¶¶]. After the initialization of the Pod, the Git repository containing the code is cloned and a specific Python script is executed with command line arguments provided to the REST API of Kubernetes. This generic job interface is used for the retargeting, dynamic time warping and motion modelling steps of the statistical modelling pipeline. The scripts use the REST interface of the storage server to download input data and store the result back into the database. If necessary, arbitrary Python scripts can be run on the cluster using this generic interface. To simplify the access management, the storage server exposes a wrapper for the job system as REST service and checks for permissions based on an access token so users do not need direct access to the cluster.

The Kubernetes cluster runs on three physical machines each with 2 Intel Xeon CPUs and 128 GB RAM. Access to GPUs is also available via the cluster but this is not relevant in our case because the statistical modelling pipeline only makes use of the CPU.

6. Application in Worker Simulation Use Case

We have applied our framework in a use case of worker simulations in a Unity application. In addition to walking, the workers needed to perform basic actions for the interaction with their environment. These actions include picking up objects, fixing screws and using a powerdrill. To model the walk action, we used an existing data set of 29 minutes that was captured using a marker-based OptiTrack system. However, the stationary actions such as picking up objects were captured using a marker-less optical motion capture system by The Captury[||||] and an IMU-based system by Noitom[***]. Each of the stationary actions was modelled using a single motion primitive. However, we created separate motion primitives for interactions with objects above and below the head of the character. The details for the newly recorded data for each motion primitive of the scenario are shown in Table 1. Figure 17 shows a comparison of the training data and samples from the statistical motion model of the pick action.

The recorded data was retargeted to a character with 63 joints using the method described in Section 4.1.2. For this purpose the source skeletons had to be first manually mapped to 21 common body joints and 30 common hand joints. Joints that were not mapped were given a default rotation. The retargeted motions were then manually segmented. However, an extra script with a heuristic was used for the segmentation of the existing walk data set into primitives. To be able to constrain the motion for the interaction

---

[||] https://sqlite.org
[**] https://www.tornadoweb.org
[††] https://unity.com
[‡‡] https://kubernetes.io/

[§§] https://git-scm.com/
[¶¶] https://kubernetes.io/docs/concepts/workloads/pods/pod/
[||||] http://thecaptury.com/
[***] https://neuronmocap.com/

Table 1: *The newly recorded motion primitives for the use case. The size is given after retargeting to a common skeleton and using the binary encoded JSON format. Source a is the optical system by The Captury and source b is the IMU-based system.*

Motion Primitive	Examples	Poses	Size (MB)	Source
Pick object	89	17779	67.884	a
Place object	35	6240	23.832	a
Pick up screws	9	3544	13.517	a
Place screws	5	2413	9.201	a
Fix screws	24	7995	30.500	a
Use powerdrill	11	4395	29.547	a
Place object high	5	3970	15.133	b
Fix screws high	5	5011	19.100	b
Use powerdrill high	5	6808	25.948	b
Use scanner	23	5835	22.269	b
Press buttons	12	1570	6.001	b
Attach cable	6	4007	15.276	b
Remove cable	6	2473	9.431	b
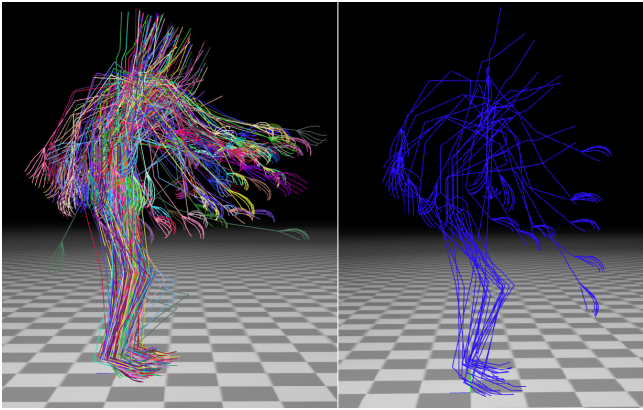
Figure 17: *Visualization of the pick training data (left) and 10 samples from the resulting statistical model (right).*

with the scene, the timeline of each clip of the stationary actions was manually annotated. Finally, each collection was given as input to the statistical modelling pipeline and organized into a graph structure connected by an idle motion.

After the time alignment the size of the training data for all motion primitives, including the walk data and mirrored motions, is 758 MB. The size of the resulting 13 motion models is 60,6 MB. The acceleration data structures, each with 10000 low dimensional samples, have a size of 173 MB. The aligned motion data, the motion models and the acceleration data structures are stored in the binary encoded JSON format described in Section 4.2.1.

To control the behavior of the workers in the scene, the motion synthesis server was integrated with the AJAN agent system as described by Antakli et al. [AHZ*18]. Each worker automatically derives constraints from annotated scene objects. Figure 18 shows the test scene with two workers driven by the motion synthesis server. The target walk trajectories for the workers are generated using the path planner of Unity. Our motion synthesis server also supports the interaction with tools as shown in Figure 19. However, to ac-

Figure 18: *Screenshot of the worker simulation in Unity with two agents. The result of the Unity path planner is shown in purple.*
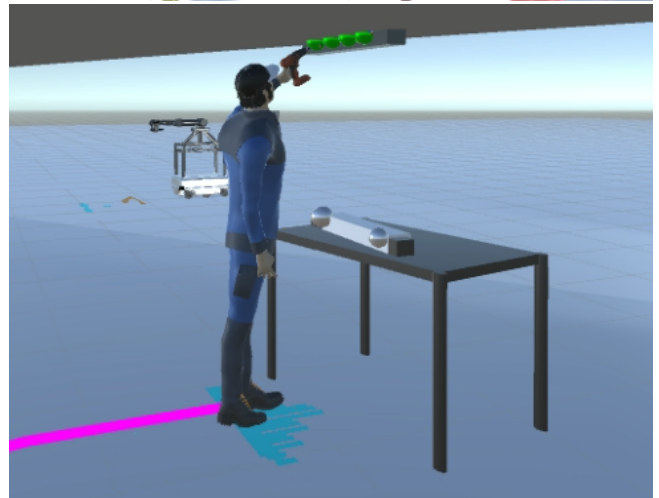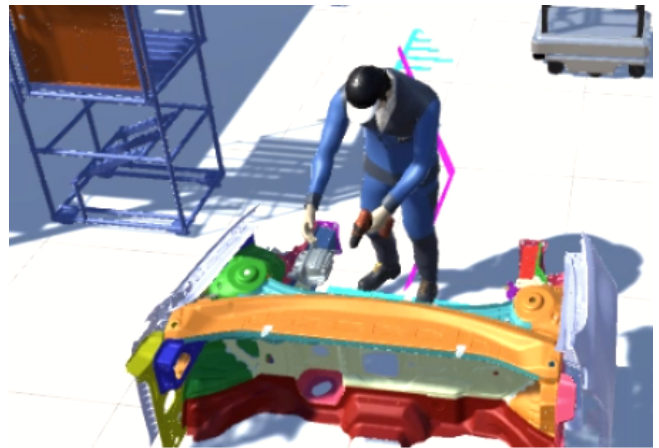
Figure 19: *Screenshots of the worker simulation in Unity using a tool to interact with objects.*

curately reach constraints using tools, inverse kinematics has to be applied.

7. Conclusion and Future Work

We have presented a data management framework for the application of statistical motion synthesis. The framework combines a motion preprocessing tool with a motion database and a model de-

ployment server. This simplifies the management of the training data and the resulting models and reduces the time needed to get feedback on the quality of the motion models. This is important, because we found in practice users have to improve machine learning models multiple times by editing the motion and adding new training data, which can come from different sources.

The integration of a Kubernetes cluster system to run jobs enabled the acceleration of the modelling process by allowing users to run multiple time expensive experiments in parallel. Furthermore, the database allowed easy tracking of multiple versions of the motion models.

The motion database can be browsed on our project website***. A subset of the motions is also available for download. We plan to make the code of the motion preprocessing tool and the backend including the data storage server, modelling pipeline and the motion synthesis server available on this website as well.

For future work we plan to integrate automatic motion annotation [ChS*19] and to move the existing motion editing functionality into the web client. We also plan to integrate other machine learning methods into our framework such as Phase-Functioned Neural Networks by Holden et al. [HKS17]. The database itself is agnostic to the type of motion model due to the storage of models in a JSON format. However, for the storage of larger models a migration to a different database system might be necessary. The combination of the database with the Kubernetes job system could be useful to run experiments for hyperparameter optimization of neural networks. By integrating different model types, our data management framework can be complementary to unified controller systems such as proposed by Gaisbauer et al. [GLA*19].

ACKNOWLEDGEMENTS

References

[Ado19] ADOBE: Mixamo, 2019. URL: https://www.mixamo.com. 2

[AHZ*18] ANTAKLI A., HERMANN E., ZINNIKUS I., DU H., FISCHER K.: Intelligent distributed human motion simulation in human-robot collaboration environments. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents* (2018), ACM, pp. 319–324. 2, 7, 9

[Aut19] AUTODESK INC.: Motion builder, 2019. URL: https://www.autodesk.com/products/motionbuilder. 2

[BWK*13] BERNARD J., WILHELM N., KRÜGER B., MAY T., SCHRECK T., KOHLHAMMER J.: Motionexplorer: Exploratory search in human motion capture data based on hierarchical aggregation. *IEEE Transactions on Visualization and Computer Graphics (Proc. VAST)* (Dec. 2013). 2

[ChS*19] CHEEMA N., HOSSEINI S., SPRENGER J., HERRMANN E., DU H., FISCHER K., SLUSALLEK P.: Fine-Grained Semantic Segmentation of Motion Capture Data using Dilated Temporal Fully-Convolutional Networks. In *Eurographics 2019 - Short Papers* (2019), Cignoni P., Miguel E., (Eds.), The Eurographics Association. doi:10.2312/egs.20191017. 10

[DHM*16] DU H., HOSSEINI S., MANNS M., HERRMANN E., FISCHER K.: Scaled functional principal component analysis for human motion synthesis. In *Proceedings of the 9th International Conference on Motion in Games* (2016), ACM, pp. 139–144. 2

[GLA*19] GAISBAUER F., LEHWALD J., AGETHEN P., SUES J., RUKZIO E.: Proposing a co-simulation model for coupling heterogeneous character animation systems. In *14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (GRAPP)* (2019). 2, 10

[HKS17] HOLDEN D., KOMURA T., SAITO J.: Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 42. 10

[HMD*17] HERRMANN E., MANNS M., DU H., HOSSEINI S., FISCHER K.: Accelerating statistical human motion synthesis using space partitioning data structures. *Computer Animation and Virtual Worlds 28*, 3-4 (2017). 2, 3

[HTM*13] HARTHOLT A., TRAUM D., MARSELLA S. C., SHAPIRO A., STRATOU G., LEUSKI A., MORENCY L.-P., GRATCH J.: All together now: Introducing the virtual human toolkit. In *13th International Conference on Intelligent Virtual Agents* (Edinburgh, UK, 2013). 2

[KCT*13] KAPADIA M., CHIANG I.-K., THOMAS T., BADLER N. I., KIDER JR J. T., ET AL.: Efficient motion retrieval in large motion databases. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2013), ACM, pp. 19–28. 2

[LS99] LEE J., SHIN S. Y.: A hierarchical approach to interactive motion editing for human-like figures. In *Siggraph* (1999), vol. 99, pp. 39–48. 3

[Mar63] MARQUARDT D. W.: An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics 11*, 2 (1963), 431–441. 3

[MBBT00] MONZANI J.-S., BAERLOCHER P., BOULIC R., THALMANN D.: Using an intermediate skeleton and inverse kinematics for motion retargeting. In *Computer Graphics Forum* (2000), vol. 19, Wiley Online Library, pp. 11–19. 3

[MC12] MIN J., CHAI J.: Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 153. 1, 2, 3

[MGT*19] MAHMOOD N., GHORBANI N., TROJE N. F., PONS-MOLL G., BLACK M. J.: AMASS: Archive of motion capture as surface shapes. *arXiv:1904.03278* (2019). 2

[Rea19] REALLUSION: iclone, 2019. URL: https://www.reallusion.com/de/iclone. 2

[RKW16] RIAZ Q., KRÜGER B., WEBER A.: Relational databases for motion data. *International Journal of Innovative Computing and Applications 7*, 3 (July 2016), 119–134. doi:http://dx.doi.org/10.1504/IJICA.2016.078723. 2

[Sha11] SHAPIRO A.: Building a character animation system. In *International conference on motion in games* (2011), Springer, pp. 98–109. 2

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics* (1985), vol. 19, ACM, pp. 245–254. 4

[SMKB13] SHOULSON A., MARSHAK N., KAPADIA M., BADLER N. I.: Adapt: the agent developmentand prototyping testbed. *IEEE Transactions on Visualization and Computer Graphics 20*, 7 (2013), 1035–1047. 2

[Vic19] VICON: Blade, 2019. URL: https://www.vicon.com/. 2

[Wri15] WRIGHT S. J.: Coordinate descent algorithms. *Mathematical Programming 151*, 1 (2015), 3–34. 3

*** http://motion.dfki.de