# Generating Motion Capture Animation for Arbitrary Rigs

## Summary

We propose a system that generates motion capture-based animation for an arbitrary input rig. Our system applies a k-means clustering algorithm to classify the joints, then uses linear regression on the clusters to output an appropriate animation sequence for each joint in the input test rig.

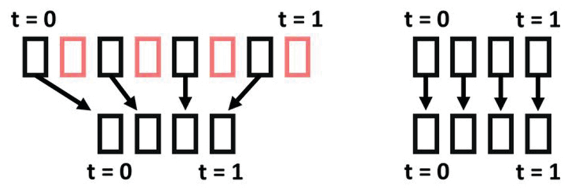
## Dataset

We have limited the scope of this project to train and test on walk cycle animations.

*BVH file format*: the BVH file format holds a skeleton, which is defined by a joint hierarchy.
*Walk cycle*: each joint has an associated set of time-varying transformations, which define the walk cycle.

## Data Preprocessing

*Normalise animation frames*



We normalise all data to 30 fps by omitting extraneous frames of animation from higher-rate samples.

*Rig splitting*
Our system is designed to works on varying rigs, so the number of joints and their placement will vary between different samples. To handle this, we “explode” each rig into individual joints and train over joints rather than rigs..

## Features

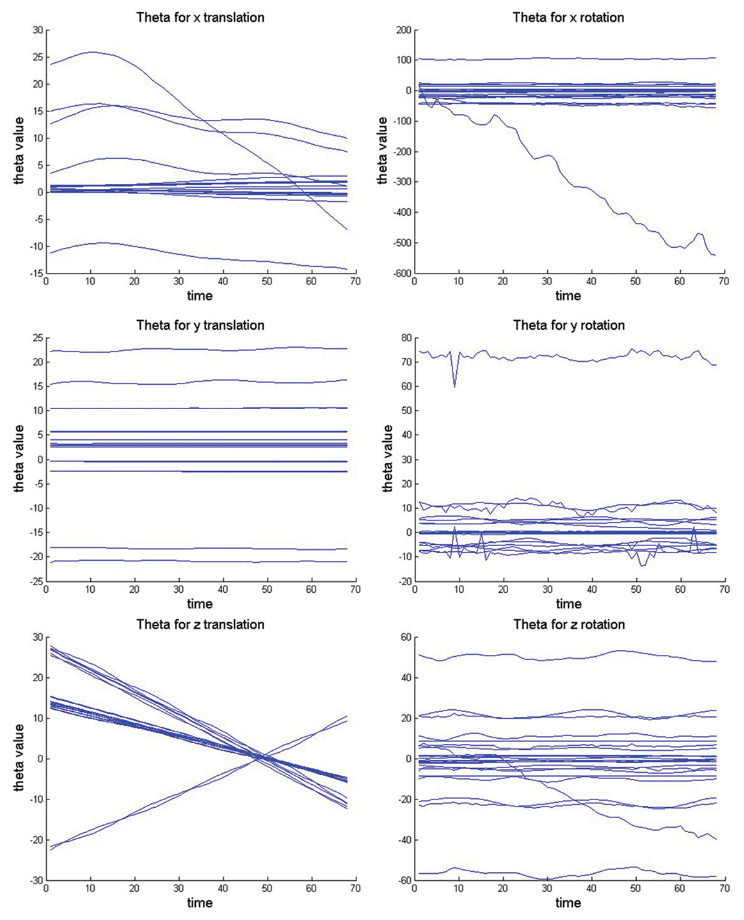
*Input Features*
Absolute position: (x, y, z) coordinates in world space.
Relative distance to parent: (x, y, z) coordinates defining the offset to the parent.
Hierarchy depth: the joint’s depth from the root.
Number of children: the number of child nodes for a given joint.

*Target Features*
We generate a sequence defining the movement of a joint over time. This output is a series of translations and rotations, each associated with a particular *t*.

Translation: a 3 x T matrix, where T is the total number of animation frames in our training data.
Rotation: a 3 x T matrix, with [x, y, z] values corresponding to the Euler rotation about each axis. We take this rotation data and rotate the position of the training joint to generate a 3 x T matrix of rotated positions.

## Final Algorithm

*Cluster-based Linear Regression*



The most effective method was a combination of the previous approaches. We used linear regression on each cluster to generate a set of thetas for each class of joint. The linear regression uses the input position of the joint in order to output the set of time-varying translations and rotations.

These graphs show the thetas of each cluster. The left column consists of the x, y, and z translations of each joint while the right column consists of the x, y, and z euler rotations.

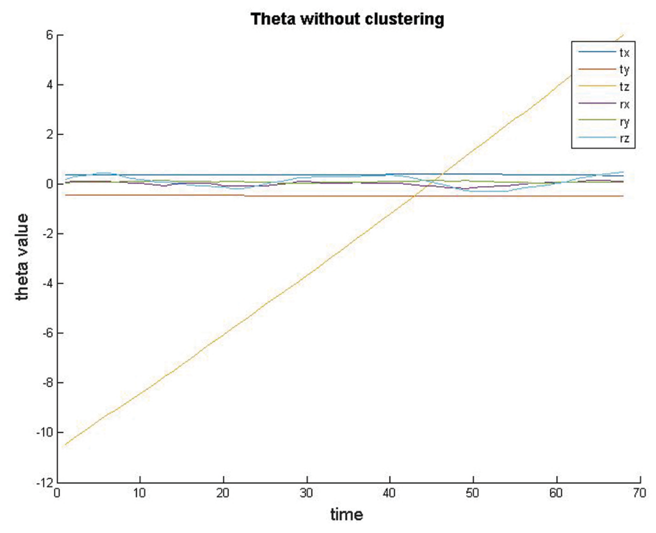
Regressing over all joints generated essentially flat thetas, as the movements of each class of joint were different enough to cancel each other out. However, regressing over each cluster, as demonstrated in the graphs above, generates much more complex and useful thetas, accurately representing the motion of that joint.

## Initial Methods

*Evaluation*
We evaluated our methods using cross-validation, training on 80% of our dataset and testing on the remaining 20%. Distance between curves was calculated using the following metric:

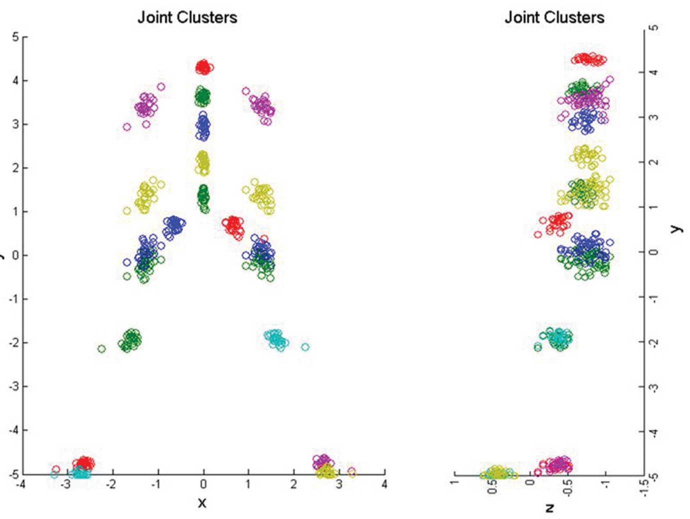
$$\frac{1}{n}\sum_{i=0}^{n}[y^{(i)} - h(x^{(i)})]^{2}$$

*Linear Regression*



Our first attempt was a linear regression over the joints. This was not very effective, as the sample data consists of a wide variety of very different joints, with very different animation sequences.

*K-means Clustering*



The sample data holds many joints of different types, which lends itself naturally to a clustering algorithm. Joints are clustered based on the set of input features described above. A test joint is then assigned to a cluster, and the output animation is the average of the animation curves of all training points in the cluster.