The goal of my project is to implement a data-driven IK solver that utilizes a large motion database to synthesize natural poses that satisfy a set of user-specified positional constraints. The solver must be able to achieve tasks such as having the character's hand touch a particular location in a reaching motion, but also infer the most natural pose given sparse motion data, such as only trajectories of end-effectors.

**Approach**

My current approach is largely based on the paper by Wu et al. [1] There are three stages to this approach:

1. **Reduced frameset selection.** In this stage, the large motion database consisting of millions of frames is reduced to a much smaller set of about 2000 most representative frames.
2. **Data modeling.** A prior model of data is learned from the reduced frameset using an approach called Gaussian process latent variable model (GPLVM).
3. **Pose optimization.** This is the runtime stage, in which numerical optimization is used to infer a pose that best satisfies the positional constraints, while being as close as possible to the previous pose, and motion priors (expressed using GPLVM).

Reduced frameset selection is achieved using an adaptive k-means clustering algorithm described in the paper by Wu et al. [1] It is necessary because Gaussian process models do not scale beyond several thousand training examples – a lot fewer than the millions of poses found in the CMU motion capture database. Reduced frameset selection is a slow process (takes about a day for the whole CMU database), but it only needs to be done once, and after that training a Gaussian process model can be done within minutes or hours, depending on the method used.

For data modeling GPLVM offers two key advantages over most other methods for modeling large datasets:

1. The probabilistic formulation allows us to use Bayesian methods to deal with data, like log-likelihood optimization. The latter is particularly advantageous, since model learning becomes a linear problem. Other minor advantages include implicit dealing with missing data, like not having particular joint information in some examples.
2. GPLVM performs dimensionality reduction, which is especially important for motion data. Because motion data is very high-dimensional (~115 dimensions), our reduced frameset of only ~2000 frames effectively becomes sparse. Learning a simple Gaussian mixture model directly from that dataset would yield a very poor approximation of the underlying
data. GPLVMs reduce the data to a much lower number of dimensions (Wu et al. use 6 latent variables) before fitting a mixture of Gaussian distributions to it.

The objective function for the pose optimization stage in the non-linear function proposed by Wu et al. [1] It has terms for proximity of the pose to priors, the previous pose (smoothness term), and prediction error (distribution variance given current input parameters). I intend to extend this formulation with a term for an input pose (obtained from a noisy performance capture device like the Kinect), with weights assigned to different degrees of freedom to express lower confidence in data that is more likely to contain noise and occlusions (e.g. shoulder and hip torsion).

**Implementation**

I am implementing my system as an extension of the ZombieHorse animation system used in my previous project. The system is written in C++ and uses a combination of standard C++ libraries and boost. I have been using third-party libraries and algorithm implementations where possible:

- Reduced frameset selection is implemented in my own code, except for the k-means clustering algorithm, for which I use the source code of the highly optimized kd-tree implementation proposed by Kanungo et al. [2]
- For GPLVMs I am using the GPLVM software by Neil Lawrence [3]. The software is a command-line tool written in C++ that reads in training data in SVMlight format.
- The pose optimization stage is not yet implemented, but since the optimization problem can be converted into unconstrained non-linear optimization problem, I will likely end up using some variant of the BFGS method to solve it.

I also intend to implement a graphical interface for specifying and editing pose constraints. This will likely be implemented as part of my animation testbed built for the previous project, which uses wxWidgets and OGRE.

**Results**

So far I have implemented the algorithm for reduced frameset selection. I have conducted limited testing of the algorithm on a small dataset (several thousand frames). Processing this small dataset takes less than a second, and the resulting poses appear to be correct (see Figure 1).
Figure 1: Representative poses chosen from a dataset containing actions such as walking, high-fiving, carrying, and picking up objects.

These poses come from a dataset containing actions such as walking, high-fiving, carrying, and picking up objects.

I have attempted to test the algorithm on a large dataset (several hundreds of thousands of frames from the CMU database), but frameset selection failed due to memory corruption after ~30 minutes. The cause is likely to be a bug on my side, but it may also be one of the documented heap corruption bugs in the Microsoft implementation of STL. I may opt to use boost containers as a less bug-prone alternative. Based on this preliminary testing, I do expect performance to be in line with what Wu et al. report in their paper (~20 hrs for a million frames).

I am currently in the phase of experimenting with GPLVM training. An example of training data output by my system into the GPLVM training tool is shown in Figure 2. Authors of the GPLVM tool recommend employing the IVM (Informative Vector Machine) approximation for training. Wu et al. propose using an FITC approximation instead, which supposedly has much lower training times, but it is unclear at present whether FITC is supported by the GPLVM tool.
Three data points are shown. Features are labeled 1 to 129. The first 12 features are inputs – 3D positions of the four end-effectors relative to root position. Overall, about a third of the intended system features are implemented at present. The biggest (and most complex) missing piece is the pose optimization. Once that is functional, implementing the remaining features like the posing GUI should be fairly straightforward.

References

