

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Learning Inverse Rig Mappings by Nonlinear Regression

Citation for published version:
Holden, D, Saito, J & Komura, T 2017, 'Learning Inverse Rig Mappings by Nonlinear Regression', *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 3, pp. 1167 - 1178. https://doi.org/10.1109/TVCG.2016.2628036

Digital Object Identifier (DOI):
10.1109/TVCG.2016.2628036

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE Transactions on Visualization and Computer Graphics

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.



Download date: 29. Mar. 2024

Learning Inverse Rig Mappings by Nonlinear Regression

Daniel Holden
University of Edinburgh

Jun Saito
Marza Animation Planet

Taku Komura
University of Edinburgh

Abstract—We present a framework to design inverse rig-functions - functions that map low level representations of a character's pose such as joint positions or surface geometry to the representation used by animators called the animation rig. Animators design scenes using an animation rig, a framework widely adopted in animation production which allows animators to design character poses and geometry via intuitive parameters and interfaces. Yet most state-of-the-art computer animation techniques control characters through raw, low level representations such as joint angles, joint positions, or vertex coordinates. This difference often stops the adoption of state-of-the-art techniques in animation production. Our framework solves this issue by learning a mapping between the low level representations of the pose and the animation rig. We use nonlinear regression techniques, learning from example animation sequences designed by the animators. When new motions are provided in the skeleton space, the learned mapping is used to estimate the rig controls that reproduce such a motion. We introduce two nonlinear functions for producing such a mapping: Gaussian process regression and feedforward neural networks. The appropriate solution depends on the nature of the rig and the amount of data available for training. We show our framework applied to various examples including articulated biped characters, quadruped characters, facial animation rigs, and deformable characters. With our system, animators have the freedom to apply any motion synthesis algorithm to arbitrary rigging and animation pipelines for immediate editing. This greatly improves the productivity of 3D animation, while retaining the flexibility and creativity of artistic input.

Index Terms—animation rig, character animation, regression

1 Introduction

Professional animators design character movements through an *animation rig*. This is a system in the 3D tool that drives the mechanics of the character, e.g. joints, constraints, and deformers. In the production pipeline, animation rigs are designed by specialists called *riggers* who are responsible for building a rig that is as productive and expressive as possible so that it intuitively covers all the poses and expressions the animators may want to create. For a complex rig there may be hundreds of rig control parameters. For example, our quadruped rig in the examples has six hundred degrees of freedom.

Yet, most character animation research and technology uses raw, low-level structures such as articulated skeletons or 3D polygon meshes as the representation. This makes them difficult to adopt in the pipeline of animated film production. After data such as motion data or deformable surfaces are captured, synthesized or edited in the raw representation, the motion has to be mapped to the animation rig for the animators to edit the results. However, there are often no clear correspondences between the rig controls and the skeletal representation. Previously, complex rig-specific scripts have been created individually for each character and rig. However, these are not general, and require revisions every time new characters and/or rigs are introduced.

We propose several frameworks to map the state of the character's kinematics to the state of some character rig. Given a set of animator-constructed examples, the joint positions as well as the corresponding rig parameters can be extracted. Our system then learns the mapping from the 3D motion data to the rig parameters in an offline stage, employing nonlinear regression techniques.

We examine and compare two types of nonlinear regression techniques: In addition to the Gaussian process regression [1] (GPR), proposed in the earlier version of this paper [2], we also present results using feedforward neural networks. While Gaussian process regression is suitable when there is not much training data, it can suffer from memory and computation issues when the volume of the training data gets larger. The feedforward neural network has excellent runtime performance, can handle larger volumes of training data, and additionally, can achieve higher accuracy than Gaussian process regression even for a small training data set by providing extra training data using super-sampling.

The rest of this paper is structured as follows. After describing about the related work, we discuss in detail about the nature of animation rigs, and show how the problem of retargeting some joint positions or joint angles can be equivalent to the inversion of some *rig function*. Next we demonstrate this rig function, its behaviours, and present the techniques we use for approximating the inverse of it. Finally, we evaluate our method, presenting a number of applications including full body motion editing and synthesis, facial animation, and 3D shape deformation. Finally we explain our results.

email:s0822954@staffmail.ed.ac.uk
email:saito@marza.com
email:tkomura@ed.ac.uk

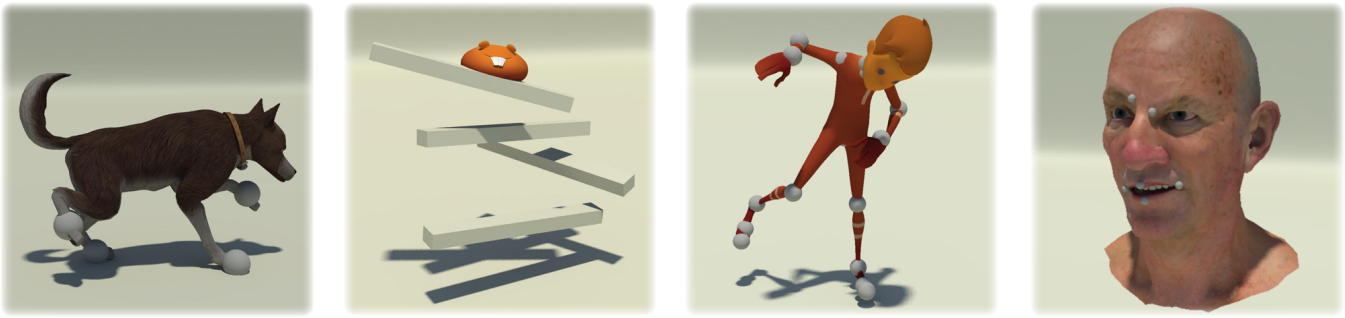


Fig. 1. Results of our method: animation is generated in the rig space for several different character rigs including a quadruped character, a deformable mesh character, a biped character, and a facial rig. This animation is generated via some external process yet because it is mapped to the rig space remains editable by animators.

The **contribution** of this paper can be summarized as follows:

- a method to invert any character *rig function* to generate accurate rig attributes from joint positions in real-time,
- a framework to use Gaussian process regression to map the joint positions to rig attributes,
- a framework to use feedforward neural networks to map the joint positions to rig attributes, and
- improving the accuracy of the inverse rig-mapping with the neural network by super-sampling the data.

2 Related Work

In this section we first briefly review research related to data-driven animation where mesh surfaces are produced by controlling the blending weights of some example data. We then review techniques that learn the mapping between parameters in the task space (i.e. joint positions, landmark positions) and the control parameters. Finally, we review the work related to animation rigs.

Animation by Blending Example Data: Data-driven approaches are known to be effective for controlling the fine details of characters, which are difficult to produce by simple analytical approaches. Facial animation is one of the main areas that makes use of data-driven approaches as the degrees of freedom of the system are too high to be entirely modelled by the animators [3], [4]. Traditionally, the desired expressions are produced by blending the geometry of different expressions which are either captured by optical cameras or are manually designed by animators. In this case the blending weights are the control parameters. Such data-driven approaches are also applied for other purposes such as skinning; Pose-space deformation [5] maps the joint angles to the vertex positions using radial basis functions. Sloan et al. [6] extends this approach to arbitrary types of mesh deformation. These methods are used to produce a forward mapping from control parameters to surfaces. We attempt the inverse of this mapping.

Inverse Mapping of Control Parameters: As directly providing the control parameters can be inconvenient in many situations there is a continuing interest in the inverse mapping. Here the control parameters are estimated from some output parameters, such as the joint positions or the vertex positions of the mesh. One example is inverse kinematics. Required are the control parameters (joint angles) that realizes the task, such as moving the hand to the target location. Classic methods include techniques such as task priority methods [7], singularity robust inverse [8], and damped least squares [9], originating in robotics research [10], [11], [12].

Researchers in computer graphics propose to directly map the joint positions to the joint angles using radial basis functions [13], [14], Gaussian processes [15] and GPLVM [16]. Similarly in facial animation, researchers compute the blending weights of different expressions from a number of landmark positions. This allows animators to control the face as if it were controlled by inverse kinematics [4], [17], [18], [19]. Xian et al. [20] proposed an optimisation based method for the inverse mapping specific to Example Based Skinning. Previous studies assume certain articulation or deformation models such as articulated joint skeletons or blend shapes while our method is agnostic to the underlying rig mechanism.

Animation Rig: *Character rigging* is the process in a professional animation pipeline where the static geometry of a character is manipulated via various animation mechanisms, such as skeletal structure, constraints, and deformers. These are then wrapped in intuitive controls for animators. Controls exposed to animators often drive underlying mechanics with custom expressions and chains of graph-structured computation nodes. This makes the rig's behaviour non-linear and difficult to formulate in general. In this paper, we refer to this general mapping of the user-exposed control parameters to the result of the underlying animation mechanics (more specifically, joint positions) as the *rig function*, and the space defined by it as *rig space*. The rig functions includes all the parameters involved in the control of the character, including but not limited to those of forward kinematics, inverse kinematics, blend shape weights, etc.

Only a few papers treat the production animation rig as a system with complex controls and layers of arbitrary underlying driving mechanisms. Hahn et al. [21], [22] introduced

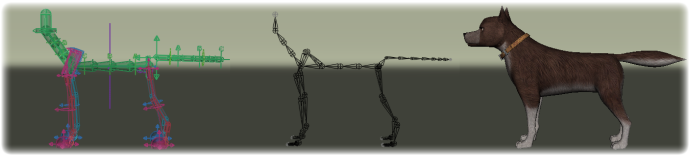


Fig. 2. Typical setup of rigged character, showing animation rig, underlying skeletal structure, and mesh.

the idea of the *rig function*, which is a black-box mapping from user-defined controls to mesh vertex positions. In this case *black-box* means that only the forward mapping in provided by the system, and there is no analytical inverse mapping available for computing the rig parameters. The major bottleneck in performing this inverse mapping of such black-box rig function via conventional techniques, as discussed in [21], [22], is computing the Jacobian by finite difference, which involves thousands of calls to evaluate a complex rig customized on a 3D software package. For arbitrary and complex rigs this becomes intractable. Seol et al. [19] is one of the few papers other papers treating the face rig as a black-box. Their objective, however, is on re-targeting plausible human expressions to virtual characters, not inversely satisfying positional constraints. Our work is motivated by speeding up such computations such that the inverse mapping can be obtained at interactive rates.

In summary, we propose an approach to produce an inverse mapping from the output of the animation pipeline to the rig parameters. Although there are methods to produce such inverse mapping for rigs consisting of simple skeletons or blendshapes, there has not been a framework that handles arbitrary types of black-boxe rig functions that can compute the inverse at interactive rates.

3 Rig Function

In this section we first explain about how the rig is used to determine the posture of a character and then describe about the requirements of the inverse of the rig function.

3.1 Rig Description

Although our approach does not rely on a specific rig, or 3D tool, to give more specific details we describe our experimental set up with an example character - a dog character as set-up in *Maya*.

Fig. 2 shows the rig of the character, the underlying skeletal structure, and the mesh. This character's rig consists of *manipulators*. These are the colourful controls, which animators can translate, rotate, or scale in 3D space. These *manipulators* move the skeletal structure, which in turn deforms the mesh. The skeleton itself cannot be moved manually by the animators, nor can the mesh.

Whenever a rig attribute is changed, *Maya* propagates the values to connected components in the scene. This causes *Maya* to recalculate a new configuration for the character skeleton. After this skeletal configuration is found, the character mesh is deformed. In this sense the setup is like a one way function going from rig attributes, to skeletal joints, and finally to the character mesh.

3.2 Rig Function & Inversion

We now describe about the mathematical characteristics of the rig function, and the requirements of its inversion.

Given a vector representing a rig configuration $\mathbf{y}$ and a vector representing the corresponding skeletal structure configuration $\mathbf{x}$, the rig computation, performed internally inside *Maya* for each frame of the animation, can be represented as the function $\mathbf{x} = f(\mathbf{y})$.

We represent the skeletal configuration of the character using a vector of the global joint positions, relative to the character's centre of gravity $\mathbf{x} \in \mathbb{R}^{3j}$ where j is the number of joints. The advantage of this representation is that the euclidian distance between two poses closely matches the visual distance between those poses. Additionally, unlike joint angles, each pose in this representation cooresponds to only a single numerical encoding (there is no double-cover). The downside of this representation is that it loses the twist information of joints and the orientation of the end effectors. We see no reason why our approach could not alternatively be applied to the joint angles of the skeletal configuration but in this paper we only discuss the use of joint positions.

Our interest in this research is in the inverse computation $\mathbf{y} = f^{-1}(\mathbf{x})$, where we compute the rig values given the skeletal posture. This is rather difficult due to the following characteristics of f, and the requirements that need to be satisfied as a tool-kit for animation purposes.

The function f is not one-to-one. For any skeletal pose there are several possible rig configurations that could create it. This is intuitively apparent from the fact that IK and FK controls can be used in conjunction on the same section of character. Some user-defined controls can manipulate multiple joints and constraints at the same time through custom expressions and chains of computational nodes. When inverting f we should not just pick a correct $\mathbf{y}$, but also the $\mathbf{y}$ which an animator would naturally specify.

The function f is relatively slow to compute. Evaluation of f in our setup requires interaction with *Maya* which has a fairly large fixed overhead associated [21]. But in any 3D package, a complex rig will also contain non negligible computation in its evaluation. It may contain several complex systems working in conjunction, which may be computationally intensive.

The solutions to the inversion of f must be accurate. If the result requires too much manual correction by animators it may be discarded. Any inversion should be able to find an accurate solution that satisfies the equation.

The function f must be invertible at interactive rates. Animation is an interactive task which requires a feedback loop between the tools and the animators. Any synthesis tools that rely on this system should have its parameters editable in real-time, so animators can view and edit the results in conjunction with the rest of the scene.

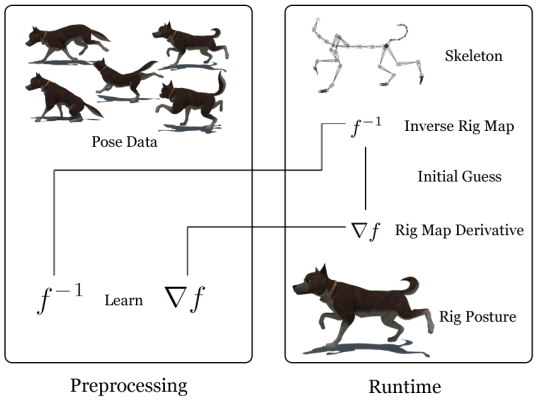


Fig. 3. Method Overview. We learn an approximation of the inverse of the rig function and its derivative and use this to accurately find rig attributes that match some corresponding joint positions.

4 Inverse Rig Mapping by Gaussian Processes

In this section, we review our original technique [2] that applies Gaussian processes regression (GPR) to the inverse rig problem. We first describe how to learn the inverse rig function and its derivative by GPR. We then describe how to refine the mapping using the learned derivatives during run-time. The summary of our method is shown in Algorithm 1.

4.1 Gaussian Processes Regression

Here we describe the mathematical framework of GPR from the viewpoint of applying it to the inverse rig mapping. A good introduction of Gaussian processes can be found in Rasmussen and Williams [1].

Given a dataset of rig configurations denoted as $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$ and the corresponding joint positions denoted as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, we are interested in predicting the rig parameters $\mathbf{y}_*$ at arbitrary configuration of joint positions $\mathbf{x}_*$.

We start by defining the covariance function, $k(\mathbf{x}, \mathbf{x}')$ using the following multiquadric kernel (see Discussion), where θ_0 is the "length scale" parameter found via optimisation (see Section 4.1.1):

$$k(\mathbf{x}, \mathbf{x}') = \sqrt{||\mathbf{x} - \mathbf{x}'||^2 + {\theta_0}^2} \tag{1}$$

Using the covariance function, we can define the following covariance matrix:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}, \tag{2}$$

$$\mathbf{K}_* = [k(\mathbf{x}_*, \mathbf{x}_1) k(\mathbf{x}_*, \mathbf{x}_2) \dots k(\mathbf{x}_*, \mathbf{x}_n)], \mathbf{K}_{**} = k(\mathbf{x}_*, \mathbf{x}_*). \tag{3}$$

It is then possible to represent each dimension i of the output $\mathbf{y}_*$ as a sample from a multivariate Gaussian distribution N:

$$\begin{bmatrix} \mathbf{Y}^i \\ y_*^i \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_*^\intercal \\ \mathbf{K}_* & \mathbf{K}_{**} \end{bmatrix}\right), \tag{4}$$

where $\mathbf{Y}^i$ is a vector of the i-th dimension of the data points in $\mathbf{Y}$, and y_*^i is the i-th dimension of $\mathbf{y}_*$. The likelihood of some prediction for y_*^i is then given by the following distribution:

$$y_*^i | \mathbf{Y}^i \sim N(\mathbf{K}_* \mathbf{K}^{-1} \mathbf{Y}^i, \mathbf{K}_{**} - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*^\intercal) \tag{5}$$

To compute our final prediction of y_*^i, we take the mean of this distribution subject to Tikhonov regularization.

$$y_*^i = \mathbf{K}_* (\mathbf{K} + \theta_1 \mathbf{I})^{-1} \mathbf{Y}^i \tag{6}$$

Where θ_1 is the "regularisation" parameter and can be set to some very small value such as 1×10^{-5}.

4.1.1 Length Scale Optimisation

The "length scale" parameter θ_0 needs to be set effectively to ensure good interpolation by the Gaussian Process. Because this is a single scalar value we perform a simple line search to find it's optimum value. We regularly take values from from the range $[1 \times 10^{-4}, 1 \times 10^{2}]$ and perform cross validation on the model. For 10 iterations we randomly remove half of the samples from the full data set, train on the remaining data and validate against the samples removed. We take the average error over the iterations to decide which value of θ_0 is best. In our case, for the quadruped character shown in the evaluation, we found a value of 0.0225 was optimum.

4.2 Subsampling

In general, the more data supplied to GPR, the more accurately it will perform. But memory usage increases quadratically with the number of data points, so we perform a greedy active learning-based algorithm to subsample the data if it grows too large.

Given the full data set $\mathbf{X}, \mathbf{Y}$ we aim to construct a subsampled data set $\hat{\mathbf{X}}, \hat{\mathbf{Y}}$. We start by including the rest post $\hat{\mathbf{X}} = \{\mathbf{x}_0\}, \hat{\mathbf{Y}} = \{\mathbf{y}_0\}$ and then heuristically picking several points to include in our subsampled data set. We iteratively pick the sample in the full data set furthest from all the included samples in the subsampled data set, and move it from the full data set to the subsampled data set. After some small number of iterations we terminate.

$$\mathbf{x}_i = \arg\max(\min(||\mathbf{x}_j - \mathbf{x}_i||) \mid \mathbf{x}_i \in \mathbf{X}, \mathbf{x}_j \in \hat{\mathbf{X}}) \tag{7}$$

$$\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} \cup \{\mathbf{x}_i\}, \mathbf{X} \leftarrow \mathbf{X} \setminus \{\mathbf{x}_i\} \tag{8}$$

$$\hat{\mathbf{Y}} \leftarrow \hat{\mathbf{Y}} \cup \{\mathbf{y}_i\}, \mathbf{Y} \leftarrow \mathbf{Y} \setminus \{\mathbf{y}_i\} \tag{9}$$

We then construct a Gaussian Process conditioned on our subsampled data. We regress each of the remaining data points in the full data set and look at the error of the result. The data point with the highest error is then moved from the full data set to the subsampled data set.

$$\mathbf{y}_i = \arg\max(||\mathbf{y}_i - \mathbf{y}_{i*}|| \mid \mathbf{y}_i \in \mathbf{Y}, \mathbf{y}_{i*} \in GPR(\mathbf{X}|\hat{\mathbf{X}})) \tag{10}$$

$$\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} \cup \{\mathbf{x}_i\}, \mathbf{X} \leftarrow \mathbf{X} \setminus \{\mathbf{x}_i\} \tag{11}$$

$$\hat{\mathbf{Y}} \leftarrow \hat{\mathbf{Y}} \cup \{\mathbf{y}_i\}, \mathbf{Y} \leftarrow \mathbf{Y} \setminus \{\mathbf{y}_i\} \tag{12}$$

This step is repeated until we've reached the required number of samples.

4.3 Learning the Derivative

The derivative of the rig function (denoted here as $\mathbf{J}$, where $\mathbf{J} = \frac{\partial \mathbf{x}}{\partial \mathbf{y}}$) can be used in conjunction with gradient descent to further improve the precision of the mapping.

At runtime, given a new target posture $\mathbf{x}_*$, a corresponding Jacobian $\mathbf{J}$ (calculated as explained in Section 4.4), and the rig values $\mathbf{y}_*$ computed using GPR, we can evaluate the rig function for the given rig controls to get the associated posture of the character $\mathbf{x} = f(\mathbf{y}_*)$. Assuming there is some error in this prediction $\mathbf{x} \neq \mathbf{x}_*$, and we can find the difference between the target posture $\mathbf{x}_*$ and the actual posture of the character $\mathbf{x}$ given by $\Delta\mathbf{x} = \mathbf{x}_* - \mathbf{x}$. This difference can be used to compute the change in rig parameters that should be applied to further minimize the error in positioning:

$$\Delta\mathbf{y} = (\mathbf{J}^\intercal\mathbf{J} + \lambda^2\mathbf{I})^{-1}\mathbf{J}^\intercal\Delta\mathbf{x} \tag{13}$$

To ensure stability around singularities we use some damping constant λ. This can be tuned by hand, or automatically selected by examining the error using the SVD of the pseudo-inverse. For more information see [12] [23]. This process is repeated until $\Delta\mathbf{x}$ is below a threshold or some maximum number of iterations is reached.

4.4 Learning the Jacobian

We treat the rig function as a black-box, so no analytical form of the Jacobian is available. Therefore, we use finite differences to compute an approximation of the Jacobian of the rig function at some given pose.

Due to the large number of interactions with Maya required, the calculation of the Jacobian in this way is extremely slow, particularly when there are a large number of rig parameters [21]. This process becomes intractable for large sequences of animation. Therefore, we additionally learn a function to predict the Jacobian alongside the rig values, again using GPR.

The formulation of learning the Jacobian is almost exactly the same as learning the rig values. After computing the Jacobian at each example pose, we flatten each Jacobian matrix $\mathbf{J}_i$ to create a single vector $\mathbf{j}_i$, and substitute it in the place of $\mathbf{y}_i$.

Some of the rig attributes are not used by the animators, yet taking the pseudo inverse may lead to these attributes being modified. Instead, we removed any rig attributes not used by the animators from the Jacobian matrix. This results in a gradient descent during the refinement only changing controls which are present in the data.

Algorithm 1 Inverse Rig Mapping

procedure PRE-PROCESSING
  Sub-sample the given animation data.
  Calculate Jacobian for each pose in the sub-sampled data
  Learn f^{-1} using GPR.
  Learn ∇f using GPR.
end procedure

procedure RUNTIME
  Predict Rig Attributes $\mathbf{y}_*$.
  Predict Jacobian $\mathbf{J}_*$.
  Initialise rig attributes with predicted values $\mathbf{y}_*$.
  Repeatedly calculate $\Delta\mathbf{x}$ and integrate $\mathbf{y}$.
end procedure

5 Inverse Rig Mapping by Feedforward Neural Networks

In this section we propose the use of a feedforward neural network for the inverse rig function. We first discuss the motivation of using the neural network for the inverse rig mapping and then describe our methodology.

5.1 Motivation

Although GPR is suitable for the inverse rig mapping when there is not much training data, in many situations the amount of the training data required to cover the rig space becomes large. The memory usage in GPR grows $O(n^2)$ with the number of samples which can cause issues with large data sets. Additionally, when the precision of the initial guess is low this approach can require the use of iteration using the predicted Jacobian. Because this involves interacting with the Maya scene this considerably slows down the process of the inverse rig mapping.

Neural networks can learn from very large sets of training data. By supersampling the example training data we can construct a much larger dataset and ensure the precision of the computed results is high for many locations near the animator supplied data.

5.2 Supersampling

Without a lot of data neural networks are susceptible to overfitting and poor generalisation. Due to this, before training, we generate an expanded data set by sampling many rig control configurations from the rig space and evaluating the rig function. This allows us to gather more data than just that provided by the animators.

For a small rig function it may be possible to exhaustively cover the rig space with this technique, effectively fully computing the rig function, but most often the rig space is

Algorithm 2 Sampling Points by PCA for NN, where the *choice* function returns a random row from the given matrix.

$\mathbf{M} \leftarrow PCA(\mathbf{Y})$
$\mathbf{X}' \leftarrow \{\}$
$\mathbf{Y}' \leftarrow \{\}$
while $|\mathbf{X}'| < n$ **do**
  $\mathbf{g} \sim \mathcal{N}$
  $\mathbf{y}' \leftarrow \mathbf{M}^{-1}(\mathbf{M}\ choice(\mathbf{Y}) + \mathbf{g})$
  $\mathbf{x}' \leftarrow f(\mathbf{y}')$
  $\mathbf{Y}' \leftarrow \mathbf{Y}' \cup \{\mathbf{y}'\}$
  $\mathbf{X}' \leftarrow \mathbf{X}' \cup \{\mathbf{x}'\}$
end while
$\mathbf{Y} \leftarrow \mathbf{Y} \cup \mathbf{Y}'$
$\mathbf{X} \leftarrow \mathbf{X} \cup \mathbf{X}'$

high dimensional, which makes the sampling process very challenging. For a small rig with just 5 parameters, a regular sampling of 10 samples on each axis would require 100000 samples. For a rig vector of 200 degrees of freedom this jumps to 10^{200} - which clearly makes a regular sampling intractable. For this reason it is important to adapt a smart sampling method which samples rig control configurations the animators are likely to use in future.

To do this we perform PCA on the data given by artists and sample around the given data points in the PCA manifold space. This allows us to sample more frequently on axes of the data with the largest variance, and therefore gather more natural samples which might be poses the animators would themselves produce in the future. The algorithm for this is shown in Algorithm 2.

Using this algorithm we sample an extra 100000 data points from the rig function for training the neural network. This process takes approximately 1 hour and greatly inceases the accuracy of the neural network regression.

In Section 6.2 we evaluate our sampling method against several other sampling methods including Uniform, Univariate Gaussian, Multivariate Gaussian, and Gaussian Mixture Model. Our evaluation is both qualitative and quantative by visualising the rig poses produced by the sampling and comparing the relative errors.

5.3 Training a Feedforward Neural Network

Given the training data produced by sampling, which is denoted by $\mathbf{X}$ as in the previous sections, we want to produce a regression that maps to the corresponding rig values $\mathbf{Y}$. Here we construct a neural network with parameters θ defined as $\mathbf{Y} = \mathbf{\Phi}(\mathbf{X}; \theta)$.

Experimentally we found it is possible to effectively perform the regression using only a small neural network with a single hidden layer. Using a small network ensures training time does not take too long given the large amount of data produced via supersampling. A deeper network may be required if it is suspected the inverse function has many highly non-linear components. We define the forward function of the neural network $\mathbf{\Phi}$ as follows:

$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{W}_1 ReLU(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1, \tag{14}$$

where the weights and biases are defined as $\theta = \{\mathbf{W}_0 \in \mathbb{R}^{h \times 3j}, \mathbf{b}_0 \in \mathbb{R}^h, \mathbf{W}_1 \in \mathbb{R}^{c \times h}, \mathbf{b}_1 \in \mathbb{R}^c\}$, j is the number of joints, c is the number of rig controls, and h is the number of hidden units in the network (in our work set to 2048).

The activation function used is the rectified linear activation $ReLU(x) = max(x, 0)$. This introduces non-linearity into the mapping. The rectified linear operation has been shown to have good performance as general purpose activation function by Nair and Hinton [24]. This network is trained using stochastic gradient descent with respect to the following cost function.

$$Cost(\mathbf{x}, \mathbf{y}, \theta) = \|\mathbf{y} - \mathbf{\Phi}(\mathbf{x}))\|_2^2 + \alpha \|\theta\|_1 \tag{15}$$

In this equation $\|\mathbf{y} - \mathbf{\Phi}(\mathbf{x})\|_2^2$ measures the squared regression error and $\|\theta\|_1$ is a sparsity term that ensures the minimum number of network parameters are used to perform the regression. This term is controlled by some small scalar α, which in this work we set to 0.1.

Before training all data is normalized by subtracting the mean and dividing by the standard deviation. We train the network by taking random minibatches of size 8 from the enlarged data set $\mathbf{X}$, and using derivatives calculated automatically using Theano [25] we adjust the network parameters θ. To increase the speed and quality of the training we use the adaptive gradient descent algorithm *Adam* [26]. Training is performed for 20 epochs and takes approximately 4 hours and 20 minutes on a 4 core Intel i7-4600U 2.1Ghz CPU.

6 Evaluation

In this section, we evaluate our method by comparing the error (described below) and the performance with other existing solutions. We then show results of our technique using animation of quadruped, biped, deformable and facial rigs.

6.1 Performance

In this section we compare our technique to existing methods of approaching this problem. Presented methods include a rig specific script constructed in Maya, and several variations of our own method. As interpretation of the numerical errors is difficult, for more qualitative results with visualisation of the errors please see the supplementary video.

Comparisons are done using the quadruped character, and biped characters shown in the results. The quadruped character has 78 joints, resulting in 234 degrees of freedom in the joint space. It has 642 degrees of freedom in animation rig, but only 218 are used in the animation data so we limit our system to only consider these. The biped character has 48 joints, resulting in 144 degrees of freedom in the joint

space. It has 804 degrees of freedom in the animation rig, but only 204 are used in the animation data so we only consider these.

A set of animation sequences of the quadruped and biped character produced by an animator each using 200 keyframes are provided as the training data. We train the Gaussian Process using 750 subsampled data points extracted from the data set including frames that are produced by interpolating the keyframes, and train the neural network on the full dataset with additional samples found via supersampling as explained in Section 5.2.

The rig-specific Maya script, to which we compare our method, is constructed using several of Maya's built-in tools. This script is specific to the quadruped character and is intended to represent existing approaches that have been used for rig retargeting. Primarily it makes use of positional and rotational constraints to place the main rig controls at corresponding joint positions, oriented in the correct directions. These constraints work by traversing the scene hierarchy to calculate transforms for the controls such that they are either placed in a specified location, or oriented in a specified direction. Many of these scripts work in a similar fashion and have to be written manually for every new rig making them labour intensive.

We apply each method to three short animation clips. These clips are from a different data set to the training data, chosen such that the character is making large, fast, or extreme motions. This ensures there exists some poses not found in the training data, and that the retargeting task is difficult. For the quadruped these clips include a motion where the dog is running around, jumping and playing, a motion where the dog is galloping and making various sharp turns, and a motion where the dog is begging in an excited way. For the biped these clips involved a short motion of the character making a variety of random movements, a swing dance motion, and a tai-chi motion. To evaluate the performance of each approach on each clip we make two comparisons. First we compare the resulting rig attributes found by the approach to those set by the animators. This we call the *Ground Truth Error*. Secondly we apply these rig attributes to the rig function to get joint positions and compare these to the target joint positions given as the original input. This we call the *Joint Error*.

Joint Error shows the mean squared error of the difference between the method's resulting joint positions, and those of the target. This is the *visual error* of the method, and also represents the amount of manual correction an animator may have to perform on the result. We regard this error as the most important as it says how closely the character follows the target positions.

Ground Truth Error shows the mean squared error of the difference between the rig attributes found by the method, and those set by the animators. For rotational rig attributes this is measured in radians and normalized to a similar range as the translational controls by dividing by the standard deviation. This error represents the *naturalness* of the key-frames produced by the method, and shows how comfortable the animators might be to use the results. But because there are multiple ways to configure a rig, and because doing comparisons in the rig space may be unreliable, this error may not be indicative of a bad result - it can be considered a secondary objective of the mapping.

Method	Joint Error	Ground Truth Error
No Extra Samples	0.235	1.196
Uniform	0.057	0.915
Univariate Gaussian	0.049	0.517
Multivariate Gaussian	0.085	1.366
Gaussian Mixture Model	0.077	0.992
PCA & Unit Gaussian	0.044	1.482

TABLE 2
Comparison of sampling methods.

We now explain the results shown in Table 1.

Maya Script - This method has the largest joint error. Because the script is a heuristic method, it does not try to find an exact solution, and so small errors accumulate over frames, even if the general shape of the character is accurate.

GPR - Using the approximate inverse rig function is very fast because there is no interaction with the Maya scene but has some residual joint error because the mapping is not exact.

GPR & Learned Jacobian - Using the approximate inverse rig function and then additionally learning an approximation of the Jacobian performs two to three times as fast as computing the Jacobian at each frame, and results in significantly less joint error than just using the approximate inverse rig function. But this approach is still somewhat slow compared to GPR alone, as each iteration of gradient descent requires evaluation of the difference in joint positions, which means interaction with the Maya scene.

GPR & Computed Jacobian - Using the approximate inverse rig function, and calculating the Jacobian manually at each frame is the most accurate approach using GPR, with the smallest joint error. But this approach is also the slowest variation of GPR, resulting in only one to two frames per second.

NN - Using neural networks to approximate the inverse rig function is very fast because there is no interaction with the Maya scene. It is more accurate than GPR, even with iterations using the Learned Jacobian. On average using neural networks also produces lower Ground Truth Error than GPR.

NN & Computed Jacobian - Using neural networks as an initial guess for the Jacobian iterations can increase the accuracy of the result further. This approach consistently gets the lowest joint error of all of the approaches.

These results were collected on a Windows 7 Laptop using a Intel Core i7 2.7Ghz CPU with 16GB of RAM.

6.2 Sampling Comparison

In this section we compare different supersampling techniques used for training the neural network. We compare our approach of PCA & Unit Gaussian sampling against

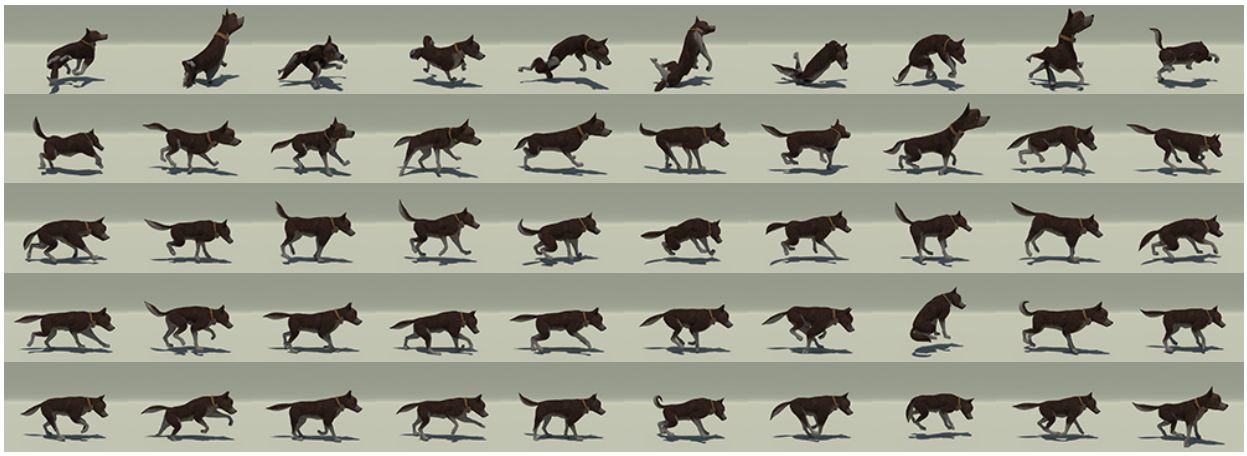


Fig. 4. Examples of samples from various methods. From Top to Bottom: Uniform, Univariate Gaussian, Multivariate Gaussian, Gaussian Mixture Model, PCA + Unit Gaussian.

Uniform sampling, Univariate Gaussian sampling, Multivariate Gaussian sampling, and Gaussian Mixture Model. The joint error and ground truth error when using each sampling method for training the neural network are shown in Table 2. For a visual comparison of the samples please see Fig. 4.

For simple models such as Uniform sampling and Univariate Gaussian sampling, the correlation between the rig controls are not considered, and therefore the generated samples appear extreme and unnatural (see Fig. 4 first and second row). Despite such visual appearance, the Joint and Ground Truth Error of the sequences produced by the neural network trained by the samples are rather low. We can assume this is due to the samples covering a wide range of rig space, allowing the system to produce postures very different from the original example data provided by the animators.

We find more complex models such as Multivariate Gaussian sampling and Gaussian Mixture Model sampling (25 multivariate Gaussians) actually perform worse than the simple models. Although they produce visually more natural samples (see Fig. 4 third and fourth row), we assume they overfit to the example data provided by the animators, and as a result produce samples that do not generalize well.

In our experiments, PCA + Normal Gaussian sampling performed the best. Here we first perform PCA on the example data set and then sample around each of the data points given by the artist using a sample from the Unit Gaussian distribution. As with the Multivariate Gaussian approach, this captures the covariance between rig controls, but also ensures dense sampling around all of the artist given data. The samples produced for training the neural network appear visually natural (see Fig. 4 bottom row), and the joint error is also low, despite the fact the Ground Truth Error is relatively large.

6.3 Results

In this section we present results of applying our system to characters such as quadrupeds, bipeds, deformable characters and facial rigs. The readers are referred to the supplementary video for the details. Numerical characteristics of the characters and the training data are shown in Table 3.

	Training Frames	Rig DOFs	Joint DOFs
Quadruped	750	642	234
Biped	750	697	144
Facial	300	49	18
Squirrel	500	36	3375

TABLE 3
Numerical characteristics of the rigs used in the experiments.



Fig. 5. Result of Rig-space Full Body IK. From seven end effectors placed at four feet, head, hip, and tail, the optimal rig attributes are approximated by GPR and the solution is further refined by gradient descent. The animators can interactively pose the character using seven end effectors while the rig attributes are updated in real time.

In Fig. 5 we apply a full body inverse kinematics system to a character using our technique. Given some user-positioned end effectors we move a copy of the characters underlying skeleton using Jacobian full body inverse kinematics toward the end effectors. We extract the global joint positions from the full body IK system and input them into our method to generate corresponding rig parameters. The generated rig parameters accurately follow the skeleton state.

In Fig. 6 we show an example of using an inverse mapping where the inputs are only the foot positions. Instead of learning the mapping using all of the joint positions we learn it from just the four foot positions. A trajectory of the foot positions is then generated from a dog dancing sequence and is fed into the system to compute the rig parameters. It can be observed that our system produces sensible prediction of the full body motion compared to the



Fig. 6. The posture of a dog predicted from the foot positions (left) and the ground truth designed by the animator (right).

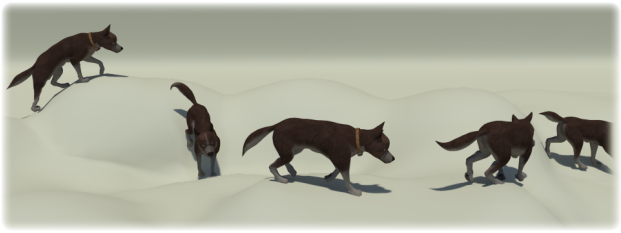


Fig. 7. Result of Motion Editing. Joint positions are synthesized by generating a locomotion animation, bending it along a curve, and projecting it onto terrain. For all the edited poses, rig attributes are found that match the new joint positions accurately.

ground truth motion designed by the animator.

In Fig. 7 we show the application of motion editing using our technique. We synthesize some locomotion using animator supplied data and use Spatial Descriptors [27] to edit the result. This advanced motion editing technique expresses an animation in terms of its environment which allows an animation to be naturally deformed to follow some terrain. First we generate a long locomotion clip in a straight line. Then we generate Spatial Descriptors on the floor, which we bend into a curve and project onto some terrain. After projecting the descriptors, we integrate them to get the new joint positions deformed to fit the terrain. This is performed for each frame. Once we have the final joint positions our method accurately updates the character rig attributes to match these new joint positions.

In Fig. 8 we show an example of importing motion capture data and mapping it to a biped character by our method. The character follows the joint positions well, the resulting trajectories of the rig controllers are smooth and continuous and ready for further edits by an animator.

In Fig. 9 we show an example of applying our system to a deformable mesh character. A specialized rig that deforms the entire surface of a squirrel character is prepared in this example. Using the rig, the posture of the squirrel can be adjusted and the entire shape of the squirrel is deformed in a cartoonish way. The rig is designed such that collisions are avoided when the deformation happens, i.e., the neck part sinks when the neck bends forward so that the teeth do not penetrate the body. Various example poses of the squirrel are designed by the animator and used as examples for training. Next, a deformable model of the squirrel is automatically produced from the default pose of the squirrel using the Maya nCloth functionality and its deformation is simulated. We place joints at all mesh vertices, pass their positions to our system and the rig parameters are computed using

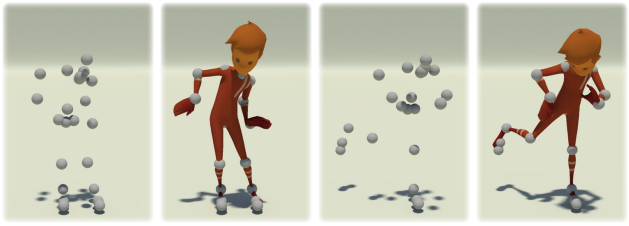


Fig. 8. Application to Biped. Our approach is generalizable across all rig types. Given data, it immediately works on a character with a different rig, and unique controls.

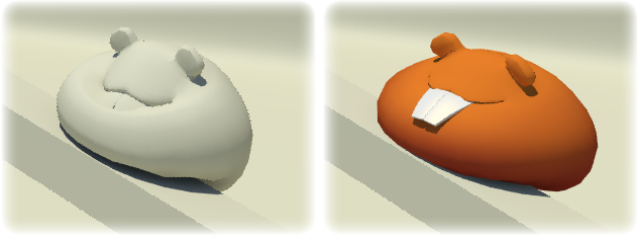


Fig. 9. A snapshot of a deformable character whose movements are computed by physics simulation (left). The vertex positions of the mesh are used as the input for the inverse mapping and the rig animation is produced (right). The rig is carefully designed such that the teeth do not penetrate the body; this effect can be observed in the animation produced by the rig.

the learned inverse mapping. The poses of the squirrel are produced which match the deformation, but also express characteristic features of the rig as can be observed in Fig. 9 - the teeth do not penetrate the body when it is squashed.

Finally, an example of applying our method to facial animation is shown in Fig. 10. Facial rigs have very complex structures that are composed of multiple controllers including shape deformers and blend shapes. Here, we present the results in a form of FaceIK. This is comparable to [4], [18] except that it is not limited facial rigs using the blendshapes deformation model. Joints are placed at a few facial feature points and the user specifies the desired location of these joints. The facial expression that satisfies these constraints is automatically computed.

The strength of our method is that it is highly general and can be applied to various types of characters controlled by different types of rigs. Specialized scripts for each type of character can be written, but they are not applicable once the rig structure changes. In contrast, our method can be applied under the same principle, irrespective of degrees of freedom of the model.

7 Discussion

In this section, we first discuss about the framework that we have chosen. We then briefly discuss the applications of our system.

7.1 Framework

Regression Methods for Inverse Mapping



Fig. 10. Application to facial model in a FaceIK fashion. The user moves the control points and the rig parameters of the face are computed to satisfy the constraints.

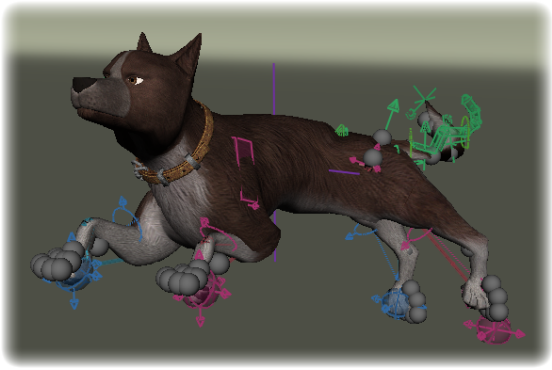


Fig. 11. Purely optimisation based techniques can result in the joints technically ending up near their targets, but the results are unusable due to rig controls drifting along manifolds in the rig space, away from valid values the animators might set.

Among the methods that we tested, feedforward neural networks produce the highest precision when no fine tuning is applied using the Jacobian iteration. The output is precise enough for practical usage and so it is not necessary to further fine tune the results. The ground truth error is also smaller than GPR. The only disadvantage of feedforward neural networks is the cost of producing more training data via supersampling and a much longer training time compared to GPR.

Sampling Methods for the Neural Network

In our experiments the proposed sampling method (using PCA with a Normal distribution) produced visually meaningful poses and the lowest numerical error. Although complex models such as multivariate Gaussians and GMMs which consider the covariance of the rig controls may appear like they should be better, the precision of the results produced by them are lower. This can potentially be due to overfitting. One possibility to improve the results from complex models is to optimize the meta parameters such as the number of Gaussians for GMM. However, such parameters will vary according to training examples and the nature of the character rig. We find our solution here based on PCA + Uniform Gaussian is a good compromise, as it is simple and requires no complex parameter tuning.

Multiquadric Kernel for GPR

We find the performance of the multiquadric kernel that we have adopted is better than other kernels including Gaussian, Linear and Polynomial kernels. Other kernels perform poorer in terms of interpolation and extrapolation, often resulting in instability and large error even after optimizing kernel parameters. The reason that the multiquadric kernel performs the best can be considered as follows: When interpolating with lots of data present, the multiquadric kernel is smooth and approximates the popular squared exponent kernel, which has proven effective for many machine learning tasks. Yet, when there are large gaps between data points, or the function is extrapolating, the squared exponent kernel results in the interpolated value tending toward zero. The multiquadric kernel on the other hand begins to approximate a linear kernel, producing a result more similar to a piecewise linear interpolation of the data points. Due to the high dimensionality, and sparsity of our data, there are often irregular gaps and spacing. Using a squared exponent kernel would therefore result in a landscape with large peaks and troughs when it irregularly drops to zero. The multiquadric kernel instead results in a stiffer landscape, and therefore extrapolates more accurately. The multiquadric function is conditionally positive definite rather than positive semi-definite, and thus is not strictly speaking a valid covariance - but the increasing nature of the kernel is what results in the behaviour similar to a piecewise linear interpolation.

Inverse Rig Function Derivative

When using GPR for the inverse mapping, we approximate the inverse rig function, and the rig function derivative separately, before taking the pseudo-inverse of the predicted Jacobian. It can be observed that it should be possible to take the first derivative of the approximate inverse rig function instead.

We predict the Jacobian separately because kernel based methods such as GPR or RBF are known to approximate the 0th order derivative (the actual function values) much more accurately than they approximate 1st, 2nd, or following derivatives [28]. This is something we confirmed in our initial experiments.

One common way to reduce this error is to incorporate gradient constraints into the GP formulation. This results in having to invert a matrix which is of the order $O(N^2D^2)$ where N is the number of samples and D is the dimensionality of the input space. In our problem, where D can be as large as 250 this quickly becomes intractable. In this sense, interpolating the Jacobian independently has remarkable performance, because although (when flattened) it can be 50000 dimensions in size, the matrix to invert is only proportional to the number of data samples.

Rig Function Ambiguity

There are many possible ways to set the rig controls to construct the same pose, but lots of these configurations are undesirable because they are not how an animator would naturally animate. Using purely optimisation-based approaches results in the rig controls drifting along manifolds which technically result in accurate joint positions, but have terrible rig values. This is shown in Fig. 11. Using

machine learning we can solve this implicitly as from the animator supplied data we additionally learn what are "valid" or "sensible" rig control settings. Even if we perform a small amount of gradient descent, using an initial guess generated from animator data ensures the error in the rig does not get too high to make the result unusable.

7.2 Applications

Our work has a large number of applications in key-framed animation environments, as it allows for the better use of character animation research and technology on rigged characters. Primarily it means that data-driven animation techniques can be effectively combined with animator artistry to save time and cost in the production of animated entertainment.

Our work allows animators to use Motion Capture, Motion Warping, and Motion Editing techniques on the motions they construct for characters. For example our work could be used in conjunction with Motion Layers [29], Motion Warping [30], Full-Body Inverse Kinematics [8], and Relationship Descriptors [27]. Because our approach is real-time it allows for a tight feedback loop between these tools and animator edits. This greatly increases the speed and efficiency at which animators can work.

8 Conclusion

We present a link between a character rig, and its underlying skeleton in the form of a rig function f. We show our method for inversion of this rig function, and evaluate it against potential alternatives. The resulting ability to quickly and effectively invert this rig function has broad applications in key-framed animation environments as it allows for a tight feedback loop between animators, and animation tools that work in the space of joint positions.

Acknowledgement

We would like to thank Marza Animation Planet [31] for their support of this research as well as their permission to use the Dog character rig and animation data. We thank Faceware Technologies [32] for permission to use their facial rig, as well as their facial animation data for training and demonstrating our system. We also thank Animation Mentor [33] for permission to use their two character rigs Stewart and Squirrel in our research.

Dog character rig and animation (© Marza Animation Planet, Inc. 2014) Stewart character used with permission (© Animation Mentor 2013). No endorsement of sponsorship by Animation Mentor. Stewart can be downloaded at http://www.animationmentor.com/free-maya-rig. Squirrels character used with permission (© Animation Mentor 2013). No endorsement of sponsorship by Animation Mentor. Squirrels can be downloaded at http://www.animationmentor.com/free-maya-rig. Old Man facial rig and animation (© Faceware Technologies, Inc. 2012).

References

[1] C. E. Rasmussen and C. K. Williams, "Gaussian processes for machine learning (adaptive computation and machine learning)," 2005.

[2] D. Holden, J. Saito, and T. Komura, "Learning an inverse rig mapping for character animation," in *Proceedings of the 14th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2015, pp. 165–173.

[3] F. H. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin, "Synthesizing realistic facial expressions from photographs," in *Proceedings of SIGGRAPH*, 1998, pp. 75–84.

[4] L. Zhang, N. Snavely, B. Curless, and S. M. Seitz, "Spacetime faces: High-resolution capture for˜ modeling and animation," in *Data-Driven 3D Facial Animation*. Springer, 2007, pp. 248–276.

[5] J. P. Lewis, M. Cordner, and N. Fong, "Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation," in *Proceedings of SIGGRAPH*, 2000, pp. 165–172.

[6] P.-P. J. Sloan, C. F. Rose, III, and M. F. Cohen, "Shape by example," in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 2001, pp. 135–143.

[7] K.-J. Choi and H.-S. Ko, "On-line motion retargetting," *Journal of Visualization and Computer Animation*, vol. 11, pp. 223–235, 1999.

[8] K. Yamane and Y. Nakamura, "Natural motion animation through constraining and deconstraining at will," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 9, no. 3, pp. 352–360, 2003.

[9] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *Journal of Graphics Tools*, vol. 10, pp. 37–49, 2004.

[10] Y. Nakamura, H. Hanafusa, and T. Yoshikawa, "Task-priority based redundancy control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 2, pp. 3–15, 1987.

[11] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," *Journal of dynamic systems, measurement, and control*, vol. 108, no. 3, pp. 163–171, 1986.

[12] S. Chan and P. Lawrence, "General inverse kinematics with the error damped pseudoinverse," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, Apr 1988, pp. 834–839 vol.2.

[13] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," in *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 559–568.

[14] C. F. Rose III, P.-P. J. Sloan, and M. F. Cohen, "Artist-directed inverse-kinematics using radial basis function interpolation," in *Computer Graphics Forum*, vol. 20, no. 3, 2001, pp. 239–250.

[15] T. Mukai and S. Kuriyama, "Geostatistical motion interpolation," in *Proceedings of SIGGRAPH*, 2005, pp. 1062–1070. [Online]. Available: http://doi.acm.org/10.1145/1186822.1073313

[16] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," in *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3. ACM, 2004, pp. 522–531.

[17] B. Bickel, M. Lang, M. Botsch, M. A. Otaduy, and M. Gross, "Pose-space animation and transfer of facial details," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2008, pp. 57–66.

[18] J. Lewis and K.-i. Anjyo, "Direct manipulation blendshapes," *IEEE Computer Graphics and Applications*, vol. 30, no. 4, pp. 42–50, 2010.

[19] Y. Seol and J. P. Lewis, "Tuning facial animation in a mocap pipeline," in *ACM SIGGRAPH Talks*, 2014, pp. 13:1–13:1.

[20] X. Xian, S. H. Soon, T. Feng, J. P. Lewis, and N. Fong, "A powell optimization approach for example-based skinning in a production animation environment," in *Computer Animation and Social Agents*, 2006.

[21] F. Hahn, S. Martin, B. Thomaszewski, R. Sumner, S. Coros, and M. Gross, "Rig-space physics," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 72:1–72:8, Jul. 2012.

[22] F. Hahn, B. Thomaszewski, S. Coros, R. W. Sumner, and M. Gross, "Efficient simulation of secondary motion in rig-space," in *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2013.

[23] S. Chiaverini, "Estimate of the two smallest singular values of the jacobian matrix: Application to damped least-squares inverse kinematics," *Journal of Robotic Systems*, vol. 10, no. 8, pp. 991–1008, 1993. [Online]. Available: http://dx.doi.org/10.1002/rob.4620100802

[24] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[25] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[27] R. A. Al-Asqhar, T. Komura, and M. G. Choi, "Relationship descriptors for interactive motion adaptation," in *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2013.

[28] N. Mai-Duy and T. Tran-Cong, "Approximation of function and its derivatives using radial basis function networks," *Applied Mathematical Modelling*, vol. 27, no. 3, pp. 197–220, 2003.

[29] J. Lee and S. Y. Shin, "A coordinate-invariant approach to multiresolution motion analysis," *Graphical Models*, vol. 63, no. 2, pp. 87 – 105, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S152407030190548X

[30] A. Witkin and Z. Popovic, "Motion warping," in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, 1995, pp. 105–108.

[31] "Marza animation planet inc." http://www.marza.com/en/.

[32] "Faceware technologies inc." http://facewaretech.com/.

[33] "Animation mentor," http://www.animationmentor.com/.

Daniel Holden Daniel Holden is a PhD student at the Institute of Perception, Action and Behavior, School of Informatics, University of Edinburgh. His research has focused on applying Machine Learning to the task of character animation in keyframed animation environments. Recently he has worked on developing deep learning frameworks for character animation editing and synthesis.

Jun Saito Jun Saito is a Research Lead at Marza Animation Planet focusing on solving practical problems in the entertainment industry using advanced mathematics. His research interests are character animation, deformation, geometry processing, computer vision, rendering, and machine learning. He is credited numerous films and games including "Space Pirate: Captain Harlock (2013)" for his contributions to the facial animation system and "Final Fantasy: The Spirits Within (2001)" for his contributions in developing massively parallel global illumination renderer.

Taku Komura Taku Komura is a Reader (associate professor) at the Institute of Perception, Action and Behavior, School of Informatics, University of Edinburgh. As the head of the Computer Animation and Robotics Group his research has focused on data-driven character animation, physically-based character animation, crowd simulation, cloth animation, anatomy-based modeling, and robotics. Recently, his main research interests have been in indexing and animating complex close interactions, which includes character-character interactions and character-object interactions

Character/Motion	Method	Joint Error	Ground Truth Error	Total Time (s)	Frames per Sec
Quadruped - *Jumping & Playing*	Maya Script	3.566	1.406	2.79	25.04
	GPR	0.153	3.877	0.21	315.06
	GPR & Learned Jacobian	0.147	3.877	10.73	6.42
	GPR & Computed Jacobian	0.071	3.864	41.24	1.67
	NN	0.087	2.604	0.21	319.44
	NN & Computed Jacobian	0.050	2.597	40.63	1.69
Quadruped - *Begging Excitedly*	Maya Script	0.161	0.424	2.68	30.95
	GPR	0.022	0.565	0.24	340.24
	GPR & Learned Jacobian	0.020	0.566	11.51	7.12
	GPR & Computed Jacobian	0.007	0.56	32.75	2.50
	NN	0.011	1.005	0.28	283.73
	NN & Computed Jacobian	0.005	1.002	33.02	2.48
Quadruped - *Turning & Galloping*	Maya Script	1.023	0.972	1.27	29.13
	GPR	0.041	1.547	0.18	195.65
	GPR & Learned Jacobian	0.039	1.547	6.29	5.71
	GPR & Computed Jacobian	0.023	1.537	23.86	1.50
	NN	0.036	0.839	0.17	208.09
	NN & Computed Jacobian	0.020	0.836	24.29	1.48
Quadruped - **Average**	Maya Script	1.583	0.934	2.24	28.38
	GPR	0.072	1.996	0.21	283.65
	GPR & Learned Jacobian	0.068	2.000	9.51	6.41
	GPR & Computed Jacobian	0.033	1.987	32.61	1.89
	NN	0.044	1.482	0.22	270.42
	NN & Computed Jacobian	0.025	1.478	32.64	1.88
Biped - *Range of Motion*	Maya Script	10.833	3.143	9.40	14.46
	GPR	0.460	0.214	0.22	608.10
	GPR & Learned Jacobian	0.326	0.215	22.75	5.93
	GPR & Computed Jacobian	0.121	0.206	53.74	2.51
	NN	0.339	0.157	0.25	527.34
	NN & Computed Jacobian	0.052	0.151	60.58	2.22
Biped - *Swing Dance*	Maya Script	8.871	2.852	4.56	14.23
	GPR	0.133	0.051	0.15	426.66
	GPR & Learned Jacobian	0.089	0.051	11.56	5.53
	GPR & Computed Jacobian	0.037	0.048	24.45	2.61
	NN	0.198	0.076	0.17	357.54
	NN & Computed Jacobian	0.044	0.073	29.18	2.19
Biped - *Tai chi*	Maya Script	8.198	3.091	13.94	14.33
	GPR	0.361	0.233	0.30	656.76
	GPR & Learned Jacobian	0.282	0.233	37.26	5.34
	GPR & Computed Jacobian	0.083	0.227	85.71	2.32
	NN	0.183	0.125	0.30	646.10
	NN & Computed Jacobian	0.031	0.122	89.88	2.21
Biped - **Average**	Maya Script	9.300	3.028	9.3	14.34
	GPR	0.318	0.166	0.22	563.80
	GPR & Learned Jacobian	0.232	0.166	23.85	5.6
	GPR & Computed Jacobian	0.080	0.160	54.63	2.48
	NN	0.240	0.119	0.240	510.32
	NN & Computed Jacobian	0.042	0.115	59.88	2.20

TABLE 1
Comparison of different methods.