# Learning an Inverse Rig Mapping for Character Animation

Daniel Holden[*] University of Edinburgh
Jun Saito[†] Marza Animation Planet
Taku Komura[‡] University of Edinburgh

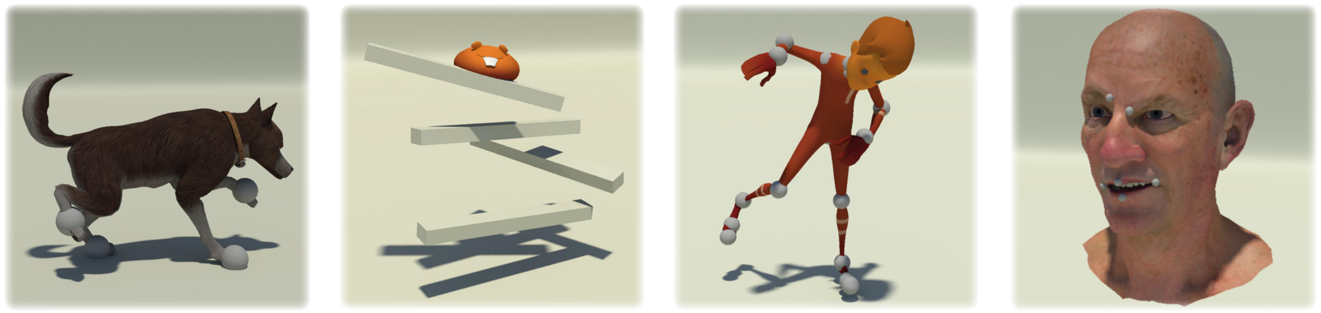

**Figure 1:** *Results of our method: animation is generated in the rig space for several different character rigs including a quadruped character, a deformable mesh character, a biped character, and a facial rig. This animation is generated via some external process, yet because it is mapped to the rig space, remains editable by animators. Dog rig and animation (© Marza Animation Planet, Inc. 2014). Squirrel and Stewart (© Animation Mentor 2013). Old Man facial rig and animation (© Faceware Technologies, Inc. 2012).*

## Abstract

We propose a general, real-time solution to the inversion of the *rig function* - the function which maps animation data from a character's rig to its skeleton. Animators design character movements in the space of an animation rig, and a lack of a general solution for mapping motions from the skeleton space to the rig space keeps the animators away from the state-of-the-art character animation methods, such as those seen in motion editing and synthesis. Our solution is to use non-linear regression on sparse example animation sequences constructed by the animators, to learn such a mapping offline. When new example motions are provided in the skeleton space, the learned mapping is used to estimate the rig space values that reproduce such a motion. In order to further improve the precision, we also learn the derivative of the mapping, such that the movements can be fine-tuned to exactly follow the given motion. We test and present our system through examples including full-body character models, facial models and deformable surfaces. With our system, animators have the freedom to attach any motion synthesis algorithms to an arbitrary rigging and animation pipeline, for immediate editing. This greatly improves the productivity of 3D animation, while retaining the flexibility and creativity of artistic input.

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

**Keywords:** rig, animation, machine learning, approximation

[*]email:s0822954@staffmail.ed.ac.uk
[†]email:saito@marza.com
[‡]email:tkomura@ed.ac.uk

## 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, through user-defined control parameters. 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 parameters. For example, our quadruped rig in the examples has six hundred degrees of freedom.

Yet, most character animation research and technologies use a skeletal representation for the character. This makes them difficult to apply in the production of animated films. After motion data has been captured, synthesized or edited in the skeleton 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 user-defined 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.

The objective of this research is to bridge this gap between character animation research and 3D film production. More specifically, we propose a framework to map the state of some character skeleton to the state of some character rig. Given a set of animator-constructed example postures, the 3D joint positions, and the corresponding rig parameters, can be extracted. Our system then learns the mapping from the joint positions to the rig parameters in an offline stage, employing Gaussian Process Regression [Rasmussen and Williams 2005].

In order to cope with the inexactness of the mapping, especially when the posture is far from the example data, we also learn the derivative of the rig function and utilizes it to fine tune the result. Numerical computation of Jacobian requires a series of slow evaluations of a rig in a 3D animation tool [Hahn et al. 2012]. To cope with this problem, we propose to learn the mapping from the skeleton posture to the Jacobian of the rig function, that linearizes the relationship of the joint positions with respect to the rig parameters. The postures computed by the initial inverse mapping are then adjusted such that the character accurately follows the 3D trajectories given in the input.

Our method can be used to apply any animation techniques that represent a character using joint angles, joint positions or a mesh structure, to characters driven by animation rigs. This includes, but is not limited to, full body motion editing and synthesis, facial animation and 3D shape deformation. In the paper we show some of these applications as results.

The rest of the 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 angles can be equivalent to the inversion of some *rig function*. Next, we will demonstrate this rig function, its behaviours, and present the technique we use for approximating the inverse of it. Finally, we evaluate our method, present a number of applications of our method, and explain our results.

Our contribution is a method to invert any character *rig function* and generate accurate rig attributes from joint positions in real-time.

## 2 Related Work

In this section, we first briefly review research related to data-driven animation where mesh surfaces are produced by controlling 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 about the work related to animation rigs, which is a professional pipeline for animating characters.

**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, where the degrees of freedom of the system are too high to be entirely modelled by the animators [Pighin et al. 1998; Zhang et al. 2007]. 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 become the control parameters. Such data-driven approaches are also applied for other purposes such as skinning; Pose-space deformation [Lewis et al. 2000] maps the joint angles to the vertex positions using radial basis functions. Sloan et al. [2001] extend such an approach for arbitrary applications of mesh deformation. These methods are for conducting a forward mapping from the control parameters to the surfaces, while we attempt the inverse mapping.

**Inverse Mapping to 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 [Choi and Ko 1999], singularity robust inverse [Yamane and Nakamura 2003], and damped least squares [Buss and Kim 2004], which originally come from robotics research [Nakamura et al. 1987; Nakamura and Hanafusa 1986; Chan and Lawrence 1988].

Researchers in computer graphics propose to directly map the joint positions to the joint angles, using radial basis functions [Kovar and Gleicher 2004; Rose III et al. 2001], Gaussian processes [Mukai and Kuriyama 2005] and GPLVM [Grochow et al. 2004]. Similarly in facial animation, researchers compute the blending weights of different expressions from a number of landmark positions, which allows animators to control the face in an inverse kinematics fashion [Zhang et al. 2007; Bickel et al. 2008; Lewis and Anjyo 2010; Seol and Lewis 2014]. Xian et al. [2006] proposed an optimisation based method for the inverse mapping specific to Example Based Skinning. The previous studies assume certain articulation or deformation models such as articulated joint skeletons or blend shapes. 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 embedded with various animation mechanisms, such as skeletal structure, constraints, and deformers, and then wrapped with 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 and 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. [2012; 2013] introduced the idea of the *rig function*, which is a black-box mapping from user-defined controls to mesh vertex positions. Where black-box means that there is only a forward mapping provided by the system, and there is no analytical inverse mapping available for computing the rig parameters. The major bottleneck in inverse mapping of such black-box rig function, as discussed in [Hahn et al. 2012; Hahn et al. 2013], 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. [2014] is one of the few papers inversely mapping the face landmarks while treating the face rig as a black-box. Their objective, however, is on retargeting 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 that satisfies constraints are 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 rig functions that treats them as black-boxes and can compute the inverse at an interactive rate. Our framework increases the precision of such an inverse mapping by learning the Jacobian for fine tuning.

## 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.

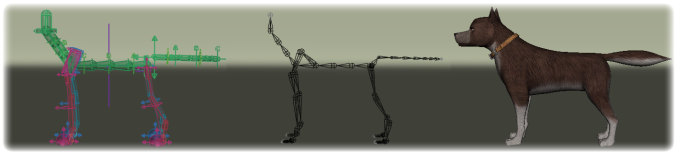
**Figure 2:** *Typical setup of rigged character, showing animation rig, underlying skeletal structure, and mesh.*

### 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 a 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. The *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

Now we 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. It is worth noting that it is also possible to construct $\mathbf{x}$ using the local joint angles of a skeletal configuration. For simplicity's sake we will only discuss the construction using global 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 [Hahn et al. 2012]. 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.

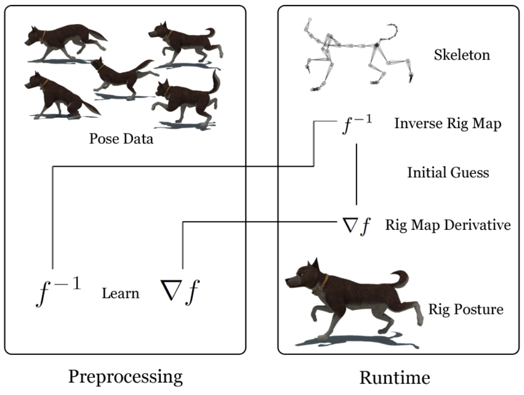

**Figure 3:** *Method Overview. We learn an approximation of the inverse of the rig function and its derivative and use this to accurately find rig attributes that match some corresponding joint positions.*

**The solutions to the inversion of $f$ must be accurate.** If the result requires too much manual correction by animators it may be discarded. In a film environment even small errors in the final product are unacceptable. 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 and make appropriate changes.

## 4 Inverse Rig Mapping

In this section we first describe how to learn the inverse rig function and its derivative by non-linear regression. We then describe how to refine the mapping using the learned values and derivatives during run-time. The summary of our method is shown in Algorithm 1.

### 4.1 Learning the Inverse Rig Function

We conduct a non-linear regression to approximate the inverse rig function using Gaussian Processes Regression (GPR). A good introduction to this can be found in Rasmussen and Williams [2005]. 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 $\theta_0$ is the "length scale" parameter and can be set to the average distance between data points in $\mathbf{X}$:

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

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

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & ... & k(\mathbf{x}_1,\mathbf{x}_n) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & ... & k(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n,\mathbf{x}_1) & k(\mathbf{x}_n,\mathbf{x}_2) & ... & k(\mathbf{x}_n,\mathbf{x}_n) \end{bmatrix}, \mathbf{K} \in \mathbb{R}^{n\times n} \quad (2)$$

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

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

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

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:

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

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

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

Where $\theta_1$ is the "smoothing" parameter and can be set to some very small value such as $1 \times 10^{-5}$ as our data is noiseless.

## 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.

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

$$\hat{\mathbf{X}} := \hat{\mathbf{X}} \cup \{\mathbf{x}_i\}, \mathbf{X} := \mathbf{X} \setminus \{\mathbf{x}_i\} \quad (8)$$

$$\hat{\mathbf{Y}} := \hat{\mathbf{Y}} \cup \{\mathbf{y}_i\}, \mathbf{Y} := \mathbf{Y} \setminus \{\mathbf{y}_i\} \quad (9)$$

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.

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

$$\hat{\mathbf{X}} := \hat{\mathbf{X}} \cup \{\mathbf{x}_i\}, \mathbf{X} := \mathbf{X} \setminus \{\mathbf{x}_i\} \quad (11)$$

$$\hat{\mathbf{Y}} := \hat{\mathbf{Y}} \cup \{\mathbf{y}_i\}, \mathbf{Y} := \mathbf{Y} \setminus \{\mathbf{y}_i\} \quad (12)$$

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

**Algorithm 1** Inverse Rig Mapping

```
 1: procedure PRE-PROCESSING
 2:     Sub-sample the given animation data.
 3:     Calculate Jacobian for each pose in the sub-sampled data
 4:     Learn $f^{-1}$ using GPR.
 5:     Learn $\nabla f$ using GPR.
 6: end procedure

 7: procedure RUNTIME
 8:     Predict Rig Attributes $\mathbf{y}_*$.
 9:     Predict Jacobian $\mathbf{J}_*$.
10:     Initialise rig attributes with predicted values $\mathbf{y}_*$.
11:     Repeatedly calculate $\Delta\mathbf{x}$ and integrate $\mathbf{y}$.
12: end procedure
```

## 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 apply the rig values to the scene and evaluate the rig function to get the actual posture of the character $\mathbf{x} = f(\mathbf{y}_*)$. There might be some error in this prediction in which case $\mathbf{x} \neq \mathbf{x}_*$, and we then 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 a change in rig parameters that should be applied to minimize the error in positioning:

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

To ensure stability around singularities we use some damping constant $\lambda$. This can be tuned by hand, or automatically selected by examining the error using the SVD of the pseudo-inverse. For more information see [Chan and Lawrence 1988] [Chiaverini 1993]. 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 [Hahn et al. 2012]. 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.

| Motion | Method | Joint Error | Ground Truth Error | Total Time (s) | Frames per Sec |
|---|---|---|---|---|---|
| *Jumping & Playing* | Maya Script | 209.48 | 37.02 | 2.68 | 26.03 |
| | GP | 41.02 | 37.68 | 3.72 | 18.51 |
| | GP & Learned Jacobian | 29.73 | 56.56 | 14.30 | 4.82 |
| | GP & Computed Jacobian | 20.17 | 49.94 | 44.33 | 1.55 |
| *Turning & Galloping* | Maya Script | 113.02 | 32.12 | 1.17 | 31.54 |
| | GP | 20.99 | 28.69 | 1.76 | 20.38 |
| | GP & Learned Jacobian | 17.02 | 32.59 | 7.20 | 4.99 |
| | GP & Computed Jacobian | 12.13 | 29.93 | 22.47 | 1.60 |
| *Begging Excitedly* | Maya Script | 42.02 | 17.26 | 2.50 | 33.16 |
| | GP | 14.39 | 22.31 | 3.95 | 20.74 |
| | GP & Learned Jacobian | 11.52 | 40.33 | 15.87 | 5.16 |
| | GP & Computed Jacobian | 6.94 | 35.73 | 36.50 | 2.24 |

**Table 1:** *Comparison of different methods.*

## 5 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, mesh and facial characters.

### 5.1 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. For more qualitative results please see the supplementary video.

All comparisons are done using the quadruped character shown in the results. This character has 78 joints, resulting in 234 degrees of freedom in the joint space. In total there are 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. A set of animation sequences of the quadruped, which are produced by the animator in total 200 keyframes, are provided as the training data. We train our system using 750 subsampled data points extracted from the data set. The data points include frames that are produced by interpolating the keyframes.

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. A script has to be written manually for every new rig and therefore is labour intensive. 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 approaches work in a similar fashion.

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. 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.

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** therefore shows the average distance 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.

**Ground Truth Error** shows the average distance between the rig attributes found by the method, and those set by the animators. For rotational rig attributes this is measured in radians. 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 many ways to configure a rig, high ground truth error alone may not imply an undesirable result.

We now explain the results shown in Table 1.

**Maya Script** - This is the quickest method to compute, but it has 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.

**GP** - Using the approximate inverse rig function is fairly fast, with low ground truth error, but with some joint error. In most cases it can be used as a replacement to the Maya script when strictly real-time performance is required.

**GP & Computed Jacobian** - Using the approximate inverse rig function, and calculating the Jacobian manually at each frame is the most accurate approach, with the smallest joint error. But this approach is also the slowest, resulting in only one to two frames per second. This approach can be used when computation time is less important, as it achieves the most accurate results.

**GP & Learned Jacobian** - Our approach of 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. It is still somewhat slow compared to GP alone, as each iteration of gradient descent requires evaluation of the difference in joint positions, which means evaluation of the rig function in Maya.

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

**Table 2:** *Numerical data of the models used in the experiments.*

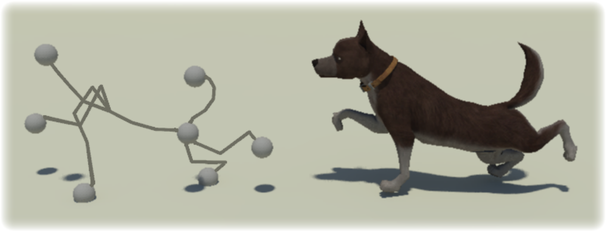

**Figure 4:** *Result of Rig-space Full Body IK. From seven end effectors placed at four feet, head, hip, and tail, the optimal rig attributes are approximated by GPR and the solution is further refined by gradient descent. The animators can interactively pose the character using seven end effectors while the rig attributes are updated in real time.*

All approaches show roughly similar Ground Truth Error. This shows that using any approach animators should be fairly happy with the settings of the rig controls. Using the learned Jacobian can sometimes result in larger Ground Truth Error. This is because the pseudo-inverse is not based on the training data, but computed so Joint Error is decreased with the smallest rig value updates. This is a downside of using the Jacobian approximation.

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

### 5.2 Results

In this section, we present results of applying our system to character models such as quadrupeds, bipeds, deformable models and facial models. The readers are referred to the supplementary video for the details. Numerical values of the models and the training data are shown in Table 2.

In Fig. 4 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 simply learn it from 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 ground truth motion designed by the animator.

In Fig. 5 we show the application of motion editing using our technique. We synthesize some locomotion using animator supplied data and use Spatial Descriptors [Al-Asqhar et al. 2013] to edit the result. This advanced motion editing technique expresses an animation in terms of its environment which allows an animation to

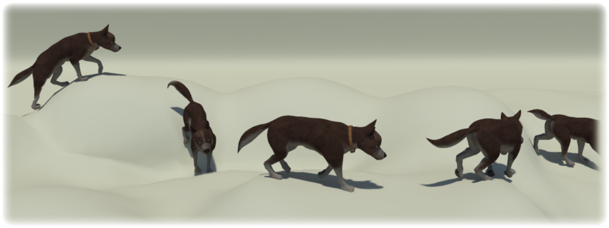

**Figure 5:** *Result of Motion Editing. Joint positions are synthesized by generating a locomotion animation, bending it along a curve, and projecting it onto terrain. For all the edited poses, rig attributes are found that match the new joint positions accurately.*

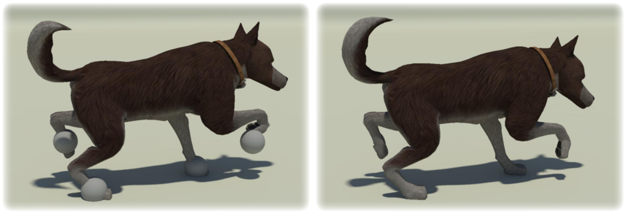

**Figure 6:** *The posture of a dog predicted from the foot positions (left) and the ground truth designed by the animator (right).*

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. 7, 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, while the resulting trajectories of the rig controllers are smooth and continuous, and as such they are ready for further edits by the animator.

In Fig. 8, 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 fashion. Also, 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 rig and are used as examples for the 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 each mesh vertex, pass their positions to our system and the rig parameters are computed using the learned inverse mapping. The poses of the squirrel match the deformation, but also express characteristic features of the rig. As can be observed in Fig. 8, the shape of the rigged character is deformed such that the teeth does not penetrate the body when it is squashed.

Finally, an example of applying our method to facial animation is shown in Fig. 9. 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 [Zhang et al. 2007; Lewis and Anjyo 2010] except that it is not limited facial rigs using the blendshapes deformation

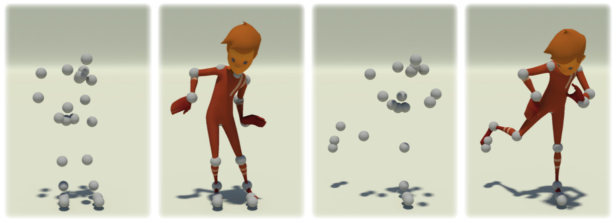

**Figure 7:** *Application to Biped. Our approach is generalizable across all rig types. Given data, it immediately works on a character with a different rig, and unique controls.*

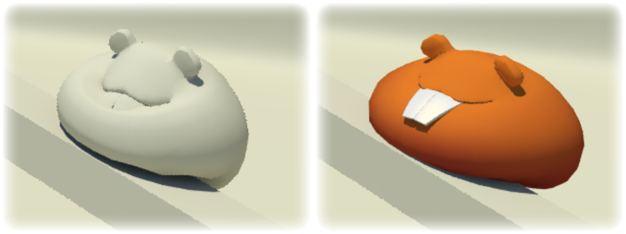

**Figure 8:** *A snapshot of a deformable character whose movements are computed by physics simulation (left). Some represented vertices are used as the input for the inverse mapping and the rig animation is produced (right). The rig is carefully designed such that the teeth does not penetrate the body; this effect can be observed in the animation produced by the rig.*

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, as it 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.

## 6 Discussion

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

### 6.1 Framework

**Multiquadric Kernel** 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 so 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 function, and so results in 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

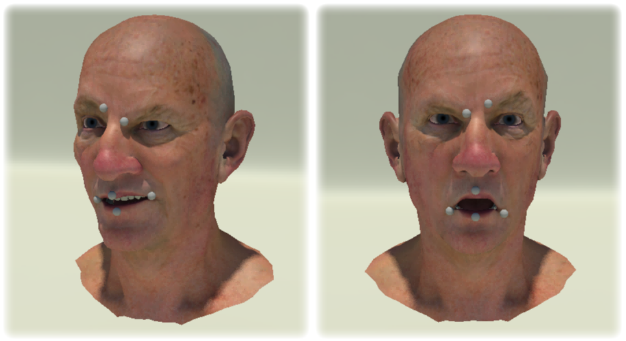

**Figure 9:** *Application to facial model in a FaceIK fashion. The user moves the control points and the rig parameters of the face are computed to satisfy the constraints.*

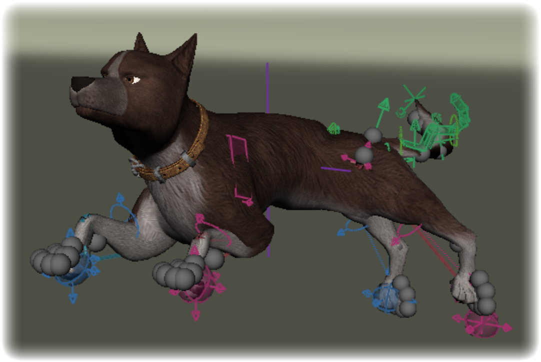

**Figure 10:** *Purely optimisation based techniques can result in the joints technically ending up near their targets, but the results are unusable due to rig controls drifting along manifolds in the rig space, away from valid values the animators might set.*

large peaks and troughs where 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 approximate of piecewise linear interpolation instead of dropping to zero. And has no impact on the computational results.

**Inverse Rig Function Derivative** In our method 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 [Mai-Duy and Tran-Cong 2003]. 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 solve 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 solve is only proportional to the number of data samples. Even so, this is an area we are very interested in and wish to do more research to try to find an appropriate technique which can be used under realistic memory constraints.

**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. 10. Using machine learning solves this implicitly by using the animator supplied data to 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 data ensures the error in the rig does not get too high to make the result unusable.

**Limitations** The memory usage of methods such as Gaussian Processes increase quadratically with respect to the number of sample points, which limits the approach to be applied to relatively small problems. For larger scale problems such as production facial rigs with thousands of poses, approaches which classify the data and conduct learning in local regions are required, such as regression forests [Breiman 2001], or Local Gaussian Process Regression [Nguyen-tuong et al. 2009].

### 6.2 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 [Lee and Shin 2001], Motion Warping [Witkin and Popovic 1995], Full-Body Inverse Kinematics [Yamane and Nakamura 2003], and Relationship Descriptors [Al-Asqhar et al. 2013]. Because our approach is in real-time it allows for a tight feedback loop between these motion tools and animator edits. This greatly increases the speed and efficiency at which animators can work.

## 7 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 for their support of this research as well as their permission to use the Dog character rig and animation data. We thank Faceware Technologies 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 for permission to use their two character rigs Stewart and Squirrel in our research.

## References

AL-ASQHAR, R. A., KOMURA, T., AND CHOI, M. G. 2013. Relationship descriptors for interactive motion adaptation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

BICKEL, B., LANG, M., BOTSCH, M., OTADUY, M. A., AND GROSS, M. 2008. Pose-space animation and transfer of facial details. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 57–66.

BREIMAN, L. 2001. Random forests. *Machine learning 45*, 1, 5–32.

BUSS, S. R., AND KIM, J.-S. 2004. Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools 10*, 37–49.

CHAN, S., AND LAWRENCE, P. 1988. General inverse kinematics with the error damped pseudoinverse. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, 834–839 vol.2.

CHIAVERINI, S. 1993. Estimate of the two smallest singular values of the jacobian matrix: Application to damped least-squares inverse kinematics. *Journal of Robotic Systems 10*, 8, 991–1008.

CHOI, K.-J., AND KO, H.-S. 1999. On-line motion retargetting. *Journal of Visualization and Computer Animation 11*, 223–235.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 522–531.

HAHN, F., MARTIN, S., THOMASZEWSKI, B., SUMNER, R., COROS, S., AND GROSS, M. 2012. Rig-space physics. *ACM Trans. Graph. 31*, 4 (July), 72:1–72:8.

HAHN, F., THOMASZEWSKI, B., COROS, S., SUMNER, R. W., AND GROSS, M. 2013. Efficient simulation of secondary motion in rig-space. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 559–568.

LEE, J., AND SHIN, S. Y. 2001. A coordinate-invariant approach to multiresolution motion analysis. *Graphical Models 63*, 2, 87 – 105.

LEWIS, J., AND ANJYO, K.-I. 2010. Direct manipulation blendshapes. *IEEE Computer Graphics and Applications 30*, 4, 42–50.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of SIGGRAPH*, 165–172.

MAI-DUY, N., AND TRAN-CONG, T. 2003. Approximation of function and its derivatives using radial basis function networks. *Applied Mathematical Modelling 27*, 3, 197–220.

MUKAI, T., AND KURIYAMA, S. 2005. Geostatistical motion interpolation. In *Proceedings of SIGGRAPH*, 1062–1070.

NAKAMURA, Y., AND HANAFUSA, H. 1986. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of dynamic systems, measurement, and control 108*, 3, 163–171.

NAKAMURA, Y., HANAFUSA, H., AND YOSHIKAWA, T. 1987. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research 6*, 2, 3–15.

NGUYEN-TUONG, D., PETERS, J. R., AND SEEGER, M. 2009. Local gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 1193–1200.

PIGHIN, F. H., HECKER, J., LISCHINSKI, D., SZELISKI, R., AND SALESIN, D. 1998. Synthesizing realistic facial expressions from photographs. In *Proceedings of SIGGRAPH*, 75–84.

RASMUSSEN, C. E., AND WILLIAMS, C. K. 2005. Gaussian processes for machine learning (adaptive computation and machine learning).

ROSE III, C. F., SLOAN, P.-P. J., AND COHEN, M. F. 2001. Artist-directed inverse-kinematics using radial basis function interpolation. In *Computer Graphics Forum*, vol. 20, 239–250.

SEOL, Y., AND LEWIS, J. P. 2014. Tuning facial animation in a mocap pipeline. In *ACM SIGGRAPH Talks*, 13:1–13:1.

SLOAN, P.-P. J., ROSE, III, C. F., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 135–143.

WITKIN, A., AND POPOVIC, Z. 1995. Motion warping. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, 105–108.

XIAN, X., SOON, S. H., FENG, T., LEWIS, J. P., AND FONG, N. 2006. A powell optimization approach for example-based skinning in a production animation environment. In *Computer Animation and Social Agents*.

YAMANE, K., AND NAKAMURA, Y. 2003. Natural motion animation through constraining and deconstraining at will. *Visualization and Computer Graphics, IEEE Transactions on 9*, 3, 352–360.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2007. Spacetime faces: High-resolution capture for˜ modeling and animation. In *Data-Driven 3D Facial Animation*. Springer, 248–276.