Integrated Honours Thesis

# Machine Learning Methods in Character Animation

By

Gnoh Cheng Yi

Department of Computer Science

School of Computing

National University of Singapore

2023/2024

Integrated Honours Thesis

# Machine Learning Methods in Character Animation

By

Gnoh Cheng Yi

Department of Computer Science

School of Computing

National University of Singapore

2023/2024

Project No: H032920

Advisors: Assoc Prof Huang Zhiyong, Prof Ji Hui

Deliverables:

Report: 1 Volume

Program: 1 Repository

## Abstract

Machine Learning methods are widely applied in Character Animation. This project consists of two parts. The first part explores Reinforcement Learning in Character Animation. A comprehensive literature review is conducted, and some of the works are reproduced. The second part explores the application of data imputation methods for inbetweening. Imputation is the task of replacing missing values with meaningful artificial values. Experiments are conducted to compare the inbetweening results between selected imputation methods and conventional inbetweening methods. The code can be found at `https://github.com/GnohChengYi/XFC4101`.



Subject Descriptors:

Animation
Machine learning approaches
Motion path planning

Keywords:

imputation, inbetweening, interpolation, keyframe, machine learning

Implementation Software and Hardware:

NumPy, Matplotlib, SciPy, scikit-learn, Python, Windows 11

## Acknowledgments

I would like to express my deepest appreciation to my project supervisor Associate Professor Huang Zhiyong for his unwavering support and guidance throughout the course of my final year project. His patience and encouragement during the challenging times of the project were a source of motivation for me.

Additionally, I am truly grateful to Professor Ji Hui for co-supervising this project. Besides, I extend my heartfelt gratitude to the friends who provided valuable feedback during the proofreading process.

Furthermore, I would like to extend my sincerest gratitude to the main evaluator, Associate Professor Lee Gim Hee, for his invaluable feedback and constructive criticism.

# Contents

# Chapter 1

# Introduction

Machine Learning methods are widely applied in Character Animation. Initially, this project focuses on the investigation of Reinforcement Learning methods in Character Animation. Then, the interim feedback directs the author to investigate methods for Artificial Intelligence Generated Content (AIGC). During the investigation, the author notices the similarity between data imputation and animation inbetweening. This observation leads to the second part of this project, which is to investigate the application of data imputation methods on the task of inbetweening.

Objectives:

1. Investigate existing Reinforcement Learning methods in Character Animation with a focus on skeletal animation. Comprehensive literature review will be conducted, and some work will be reproduced.

2. Investigate the application of data imputation methods on the task of inbetweening. Ways to represent animation in tabular format for data imputation will be proposed. Comparative evaluation of inbetweening quality between data imputation methods and conventional inbetweening methods will be conducted.

# Chapter 2

# Reinforcement Learning in Character Animation

Reinforcement Learning (RL) is one of the three categories of modern machine learning, along with Supervised Learning and Unsupervised Learning. RL uses trial-and-error to interact with the environment, receives feedback through a reward function, and updates its controller to maximize the reward.

There are two main problems in Character Animation: Skeletal Animation and Character Motion Planning. Skeletal Animation deals with internal motions of an agent – how the individual limbs move, while the position in the global frame may be of secondary concern. Character Motion Planning is the opposite -it concentrates on the character's movement throughout the scene.

RL stands out as a promising approach for Character Animation because it provides a versatile framework to learn motor skills without the need of labelled data. RL is particularly useful when the dynamic equations of the environment are unknown or non-differentiable, to which conventional gradient-based optimal control algorithms do not apply.

This part of the project will study RL for character animation with focus on Skeletal Animation and Character Motion Planning. Investigations will be conducted on existing RL methods in character animation.

## 2.1 Literature Review

This literature review takes Reinforcement Learning (RL) as the overarching theme, with a top down approach from Character Animation to Skeletal Animation, and then to Motion Synthesis. After that, this review zooms into a specific task: chopsticks manipulation, before reflecting on the insights gathered.

### 2.1.1 RL Methods in Character Animation

Character Animation involves everything about animating virtual characters, in the broad sense. In this review, I focus on the behavioural aspects of the character. Last year, Kwiatkowski et al. [19] surveyed the modern Deep Reinforcement Learning (DRL) methods and how they could possible be applied in Character Animation. Most of this section is based on their survey.

They described some ways to classify RL algorithms, but warned that the separation could be ambiguous. The ways to classify RL algorithms include:

- Policy-based vs Value-based: Policy-based (PB) algorithms train their neural networks to return the action that maximizes reward, while value-based (VB) algorithms train their networks to produce the expected reward value of each action.

- Actor-Critic: Instead of one, two networks are trained. One of them is the Actor, which estimates what action. The other one is the Critic, which estimates the reward value of performing an action.

- On-policy vs Off-policy: In On-policy, the policy that selects actions is the same as the policy that is being optimized. But in Off-policy, these two policies can be different.

- Model-free vs Model-based: In Model-based approaches, a model of the environment is learned, allowing the algorithm to do some planning based on the learned environment model. On the other hand, in Model-free approaches, the

agent has no way of forecasting the consequences of an action before actually executing it in the environment.

- Single-agent vs Multiagent: Many algorithms are designed to work for one agent in the environment. Some algorithms can additionally work for multiple agents being in the same environment at the same time.

Kwiatkowski et al. [19] noted that many of the most successful approaches drew ideas from various categories, resulting in algorithms that are technically speaking, actor-critic and off-policy. They also suggested that value-based methods are more appropriate when additional data is hard to collect, whereas policy-based methods are more applicable to when hardware needs is a bigger concern.

There are two main problems in Character Animation: Skeletal Animation and Character Motion Planning.

Skeletal Animation focuses on the internal motion of a virtual character, such as the movements of each single limbs. The character's position in the global setting is often viewed as less significant. A literature review on this domain is provided in section § 2.1.2.

On the other hand, Character Motion Planning is the inverse. It does not concentrate on the specifics of the character pose, but rather emphasizes the character's displacement through the scene. Very often, we need to consider Character Motion Planning for many interacting characters. This consideration changes the problem into Crowd Animation. Usually, each character has a source (its current position) and destination, and their motions should be planned such that each character reaches its destination without colliding with other characters and obstacles in the environment.

Long et al. [27] used the Proximal Policy Optimization algorithm for collision avoidance, which is a sub-task of Crowd Animation. They showed that the resulting policies are better than Optimal Reciprocal Collision Avoidance (ORCA). In the same year, Lee et al. [20] applied the Deep Deterministic Policy Gradient (DDPG) algorithm to Crowd Animation and obtained sufficient but imperfect results.

One shortcoming of approaches purely based on RL, is the lack of consideration for human-like behaviours. This is because the characteristics of human-like behaviours are not well-defined [19]. Future research can focus on generating reward functions from real-world data that reflects desired human-like characteristics.

### 2.1.2 RL Methods in Skeletal Animation

Skeletal Animation is primarily concerned with the individual limb movements of a virtual character. The character's position in the global context is often considered less important. In 2022, Mourot et al. [30] presented a comprehensive survey on the state-of-the-art approaches based on Deep Learning (and DRL) in skeleton-based human Character Animation. Most of this section is based on this survey.

One critical aspect of DRL in Skeletal Animation is the pose representation. The pose representation determines the input and output spaces, and thus directly affects the aspects of the knowledge that can be learned. Pose representations can be mainly categorized into two types: positional pose representation and angular pose representation. In a positional pose representation, the location of each joint is explicitly defined, in the coordinate system of the body. The disadvantages of this representation include missing information of bone orientations and unfixed bone lengths. On the other hand, angular pose representation hierarchically encodes the orientation of joints and their children, while fixing the bone lengths [30]. The main disadvantage of angular pose representation is the propagation and accumulation of errors due to its hierarchical nature. To address the pros and cons of positional and angular pose representations, some work proposed and improved hybrid representations successfully [1] [40].

Motion Synthesis is an area of Skeletal Animation where DRL can be applied. The task of Motion Synthesis aims to generate motion sequences that are convincing, usually embodying a specified style. A critical literature review is provided in section § 2.1.3.

Another application of RL in Skeletal Animation is the physics-based approaches in character control. Character control involves controlling the motions of a virtual

character based on user inputs. The character should respond as naturally as possible to user inputs, without violating environmental constraints [30]. Physics-based approaches aim to generate physically realistic animations. Usually, the reward function combines physics-based penalties (e.g. losing balance) and task-based rewards.

One constraint of RL approaches in Skeletal Animation is the difficulty to mimic the natural behaviour of humans. A relatively successful approach was proposed by Peng et al. [34], which depends more on data from the real world in addition to their physical model.

### 2.1.3 RL Methods in Motion Synthesis

Motion synthesis aims to discover a model which is capable of generating new movements. In this review, we focus specifically on human motions. Very recently, Loi et al. [26] gave a review and classification of the most recent works regarding motion synthesis using machine learning techniques. Most of this section is based on their survey.

A popular approach is demonstrated by Ling et al. [24], who employed DRL to control a variational autoencoder (VAE) that synthesize motions directed towards a goal. The RL model is trained to learn control policies that utilize an action space defined by latent variables to manage the variational autoencoder. Li et al. [22] and Cai et al. [6] also used variational autoencoders to synthesize new motions.

Another popular approach is the physics-based approach. Lee et al. [21] introduced an algorithm that learns a parameterized set of motor skills from just one motion clip. This algorithm incorporates a continuous-time RL network to capture both the temporal and spatial variations of the reference motion. In [52], Yuan et al. proposed a unique method known as residual force control (RFC) to overcome the discrepancy in dynamics between the humanoid model and actual humans. In both of these work, DRL models are trained to learn control policies for physically plausible movements, as part of motion imitation tasks. On the other hand, Bergamin et al. [3] used RL to offer a feedback mechanism to preserve the

balance of a character.

Peng et al. [35] took a step further, by proposing adversarial motion priors (AMP) to eliminate the need for manually creating reward functions for different movements. They used motion clips to train an adversarial system, which defines style-rewards for the character's training via RL.

One shortcoming of AMP is its vulnerability to mode collapse, similar to many other techniques based on Generative Adversarial Networks (GANs). When given a large and diverse data set of motion clips, the policy tends to mimic only a small subset of the example behaviours.

### 2.1.4 RL Methods in Chopsticks Manipulation

Chopsticks are pairs of sticks of equal length, usually used as a tool for cooking and eating. The task of chopsticks manipulation involves using a hand to hold the chopsticks, while manipulating it to achieve a certain task. For example, chopsticks can be used to pick up a small object, moving the object to another place while preventing it from falling down, and putting the object to its target destination.

Many researchers approached this task from a robotic perspective [32] [8] [53]. Zhang et al. [53] presented a RL system that uses chopsticks for fine manipulation. They achieved an almost 100% success rate on the task of using chopsticks to grasp small objects swinging in the air. However, the chopsticks were attached directly to the arm of the robot, which skipped the challenge of keeping the chopsticks in the hand when manipulating object. Besides, approaches from the robotic perspective often do not require the resulting motions to be human-like, which is different from the needs of a chopstick animation.

The author only found one closely related work which focus on animating chopsticks manipulation: [50]. Yang et al. [50] dealt with tasks that involve moving objects with chopsticks. They used Bayesian Optimization (BO) and DRL to optimize gripping poses, plan motion trajectories, and train hand controllers. Their framework is able to relocate objects of various shapes and sizes, in diverse gripping styles and holding positions, for multiple hand morphology. With the

consideration of gripping styles, their work inherently results in more human-like motions compared to robotic approaches.

A constraint of the work of Yang et al. [50] is their choice of widely accepted design options whenever feasible. For example, they used Proximal Policy Optimization (PPO) for DRL, and employed MuJoCo with its default parameters for simulations. It might be worth to explore whether PPO is truly the best choice for chopsticks-based object relocation tasks [10] [45]. Similarly, the default simulation parameters of MuJoCo might not be the best for this scenario.

### 2.1.5 Reflection

A recurring issue of RL in character animation is the challenge in replicating human behaviour. This is because the characteristics of human-like behaviours are not well-defined [19]. Peng et al. proposed several data-driven approaches [34] [35] to address this issue, but there are some limitations.

On the other hand, the task of chopsticks manipulation is largely unexplored in the field of character animation. This presents a good opportunity for me to contribute. Some directions that might be worth exploring are:

- Different shapes of chopsticks: In the real-world, the chopsticks are not always perfectly cylindrical. Chinese chopsticks are longer and thicker, Japanese chopsticks have a pointer tip, while Korean chopsticks appear flat.

- Meat flipping task: Beyond object relocation, meat flipping poses the challenge of rotating the lifted object (meat) upside down. The meat can also undergo deformation, as opposed to a rigid body. This distinction might make it more difficult to lift the meat.

## 2.2 Reproduced Work

The first work that the author tried to reproduce was DeepPhase [43]. It took very long time to clone the code repository for this work, because it is part of a

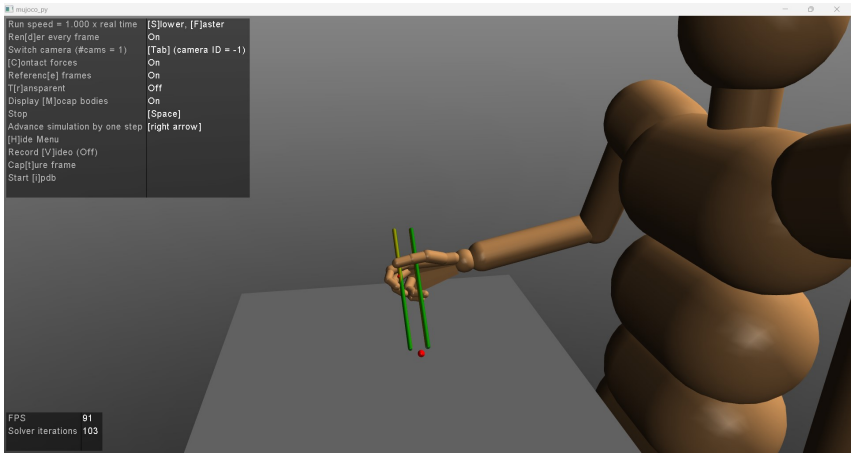


Figure 2.1: A demonstration of chopsticks-based object relocation task.

bigger project named AI4Animation. It also took a lot of time to train the model and run it in Unity, both of which are computationally expensive tasks. The author noticed part of the reason is the limited RAM of his laptop, so the author upgraded the laptop memory.

Next, the author tried to reproduce the work of Bailando [41]. The author realized that it is more reasonable to train the model using GPUs on SoC Compute Cluster instead of locally on his laptop. So, the author learned to use Slurm, the job scheduler of SoC Compute Cluster. Next, there were many versioning issues with the Python packages used by this work. After debugging the versions for a long time, the author discovered that PyTorch 1.6 is required by the work, which is incompatible with the Python version (3.10.12) on SoC nodes. Unfortunately, the author could neither install Conda nor another version of Python, both of which require a higher access privileges. So, the author moved on to explore cloud providers.

Finally, the author reproduced the work of [50], see Figure 2.1. The authors of [50] did not specify the version of Python that they used, so the author reviewed all the versions of the required packages to determine a suitable version for Python. There were also some installation errors which the authors might have fixed at the beginning of the project and forgot to document them. To fix these issues, the author read many documentations, Stack Overflow answers and GitHub issues.

# Chapter 3

# Imputation Methods for Inbetweening

After the CA presentation of last semester, the author was encouraged in the feedback to investigate methods for artificial intelligence generated content (AIGC). This investigation revealed various AIGC methods, with diffusion models standing out as a very promising area due to their impressive results [37].

However, a key limitation of diffusion models is their high requirements of computational resource. Upon closer inspection, the author noticed that the denoising step within diffusion models always rely on neural networks. This is questionable because the theoretical specification of diffusion models do not explicitly require neural networks. Instead, they simply need Universal Function Approximators (UFA) [16] to learn the denoising step.

Further research led the author to a paper proposing the use of XGBoost [9] for the denoising step [16]. XGBoost is a popular Gradient-Boosted Tree (GBT) technique. In the paper [16], this technique was applied to the generation and imputation of tabular data. Notably, the concept of imputation reminded the author about inbetweening in animation. This connection inspired the author to investigate the potential application of imputation methods on inbetweening tasks.

## 3.1 Literature Review

Two distinct reviews will be conducted initially: one for imputation methods and one for inbetweening methods. The focus of imputation methods section will be

on their application to tabular data. On the other hand, the inbetweening methods will center on their use in skeletal animation.

Following the two sections mentioned, the review will continue with existing research that has utilized imputations within the context of inbetweening tasks. Finally, the review ends with the prominent data representations employed in skeletal animation.

### 3.1.1 Imputation Methods

Real-world datasets often contain missing values. This can occur for a variety of reasons, such as incomplete surveys where respondents skip questions. During data analysis, these missing values need careful consideration.

One possible method is to exclude rows with missing data. However, this method might lead to substantial data loss, potentially compromising the reliability of the analysis results.

A more common method is data imputation, which involves replacing missing values with plausible guesses. This method aims to create a complete dataset (also known as imputation) that allows for meaningful statistical analysis.

One commonly used method for imputation involves replacing the missing values of a specific variable with the mean of all known values for that variable [5]. This method works well when the missing data are strongly linked to weak relations, which is what Burt [5] observed in a general social survey. Unfortunately, this assumption might not hold in many other scenarios.

The Fully Conditional Specification (FCS), also known as Multivariate Imputation by Chained Equations (MICE) is another popular imputation method [25, 39]. This method imputes variables one at a time from a series of univariate conditional distributions [29]. An R implementation of MICE can be found at [47].

Troyanskaya et al. [46] implemented and assessed data imputation methods based on the K Nearest Neighbors (KNN) and Singular Value Decomposition (SVD) in gene microarray data. These two methods have demonstrated superior performance compared to conventional approaches of filling missing data with means, by leveraging

the correlation structure of the data to estimate the missing values more effectively. They recommend KNN-based method for its robustness and sensitivity.

The Soft-Impute algorithm introduced by Mazumder et al. [28] is a matrix completion algorithm that leverages a unique "sparse plus low-rank" structure associated with the Singular Value Thresholding (SVT) algorithm to efficiently compute the SVD. It iteratively replaces missing values in a matrix with values obtained from a soft-thresholded SVD. The process of filling in missing values of a partially observed matrix is equivalent to data imputation.

Continuous and categorical variables each require distinct approaches for imputation. To seamlessly handle both types of variables, Stekhoven et al. [44] introduced an iterative imputation method called missForest that relies on a random forest. Notably, they utilized the out-of-bag error estimates provided by the random forest, allowing them to estimate the imputation error without requiring a separate test set.

Using the widely recognized Generative Adversarial Nets (GAN), Yoon et al. [51] proposed a method known as Generative Adversarial Imputation Nets (GAIN). The generator imputes the missing values based on what is actually observed to produce a completed vector, while the discriminator attempts to discern which values were genuinely observed and which were imputed.

A comparison of machine learning models for data imputation by Platias et al. [36] found missForest [44] to be the overall best. Another good choice they found is Sciblox Mice, which is an implementation of MICE in Python with XGBoost [9] integrated. Their comparison did not include GAIN.

Jäger et al. [15] performed an extensive set of experiments that includes GAIN. Their findings indicate that high-capacity deep learning models (e.g. GAIN) do not consistently outperform conventional methods (e.g. random forests). For numerical data that are missing not at random (MNAR) with the largest missing fraction (50%), random forest performs the best, closely followed by KNN. Note that they implemented their own random forest method instead of using missForest.

### 3.1.2 Inbetweening Methods

In traditional animation, each frame is a static snapshot within a sequence which creates the illusion of movement. For continuous actions, consecutive frames have very little differences. While generating every individual frame guarantees a smooth animation, it is often impractical due to resource constraints.

This is where the inbetweening technique comes in. Instead of meticulously crafting every frame, animators create keyframes - representative points that depicts the start and end of an action. The missing intermediate frames are then in generated computationally using various methods, such as simple interpolation. This approach significantly decreases the workload while generating a smooth animation.

This section focuses on inbetweening methods within the domain of skeletal animation. Here, each keyframe provides the complete graphic parameters defining the scene, exceeding the limitations of a simple pixel table (image).

One of the most conventional inbetweening method is linear interpolation (LERP). The graphic parameters are linearly interpolated between keyframes to generate new frames between them. However, the resulting animation might be unrealistic, especially when motion acceleration is involved between keyframes. Nevertheless, LERP is still widely used due to its simplicity and satisfactory quality.

Another conventional inbetweening method is spherical linear interpolation (SLERP) [18]. As its name suggests, this method borrows the idea of LERP. Instead of interpolating on the Cartesian coordinates, SLERP transforms Cartesian coordinates into spherical coordinates, applies linear interpolation, then transforms the spherical coordinates back to Cartesian coordinates. This method is very widely used as it makes rotational motions more realistic.

In a survey paper in 2018, Haarbach et al. [12] discussed higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation. In particular, they compared the Euclidean, orientation and rigid body interpolation methods on different manifolds. For the Euclidean space, two most popular higher order interpolation methods are LERP and Bezier curve.

A Bezier curve is defined by a set of discrete control points, which together create a smooth, continuous curve.

More recently, new methods that take advantage of modern machine learning ideas are proposed. Oreshkin et al. [31] proposed a deep learning based interpolator that uses the Transformer. The Transformer's global attention mechanism is good at handling long-term dependencies, which means it is good at generating longer transitions. Qin et al. [38] presented a deep learning-based framework to synthesize motion inbetweening via two-stage Transformers. Starke et al. [42] introduced another motion inbetweening system which makes use of Periodic Autoencoder, and achieved better results than the two-stage Transformers in some scenarios. These methods are much more complex than LERP and SLERP, and take significantly longer to finish the inbetweening tasks [42].

### 3.1.3 Application of Imputation in Inbetweening

This part of the project proposes a novel approach by reformulating the inbetweening task within skeletal animation as an imputation problem. A review of existing literature reveals no prior research explicitly addressing inbetweening as an imputation task. However, imputation methods have been employed in a limited capacity within the broader field of animation research.

Karunratanakul et al. [17] applied imputation to generate human motion sequences from partial observations. In their Diffusion-based Probabilistic Model (DPM), they performed imputation on the sample after every denoising step. On the other hand, Beserra et al. [4] performed imputation when they have features with different sizes. The imputation method they considered was simply filling in zeros.

## 3.2 Methodology & Implementation

This section outlines the selected imputation and inbetweening methods for experiments. It also discusses some design considerations of the experiments.

| $t_1$ | $a_{t1}$ | $b_{t_1}$ | $c_{t_1}$ | $d_{t_1}$ | $\cdots$ | $t_n$ | $\cdots$ | $d_{t_n}$ |
|---|---|---|---|---|---|---|---|---|

Table 3.1: One-line representation of animation.

| $t_1$ | $a_1$ | $b_1$ | $c_1$ | $d_1$ |
|---|---|---|---|---|
| $t_2$ | $a_2$ | $b_2$ | $c_2$ | $d_2$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |

Table 3.2: Rectangular representation of animation.

### 3.2.1 Data

#### 3.2.1.1 Data Specification

Imputation methods traditionally operate on tabular data only. However, for their application in inbetweening, a suitable tabular representation capturing the animation data is necessary. This project focuses on skeletal animation, where the frame information is encoded in various parameters such as joint positions and bone lengths.

One way of representing an animation in a tabular format is to simply encode the whole animation in a single row. Assuming a frame is specified by four parameters, this can be done as shown in Table 3.1, where $a, b, c, d$ are the four parameters that define a scene, $t$ is the time frame, while $n$ is the number of frames.

However, this representation is not suitable for imputation methods, which considers each column a variable. With only a single row, imputation methods "see" a single observation of a lot of variables, with no other observation helping to derive the relationships between variables. When one or more values are missing (as in the case of inbetweening task), imputation methods cannot produce a reasonable imputation. Similarly, using a single-column representation is incompatible with imputation methods as they interpret it as a single variable.

A more reasonable representation is to use each row to represent a frame (Table 3.2). This representation enables imputation methods to meaningfully interpret each column as a variable (e.g. the position of a specific joint), while having many rows to learn the relationship between variables.

| $t$ | $x_{At}$ | $y_{At}$ | $z_{At}$ | $x_{Bt}$ | $y_{Bt}$ | $z_{Bt}$ |
|---|---|---|---|---|---|---|
| 0 | 5.0 | 0.0 | 0.0 | -5.0 | 0.0 | 0.0 |
| 1 | 4.0 | 3.0 | 0.0 | -4.0 | -3.0 | 0.0 |
| 2 | 3.0 | 4.0 | 0.0 | -3.0 | -4.0 | 0.0 |
| 3 | 0.0 | 5.0 | 0.0 | 0.0 | -5.0 | 0.0 |
| 4 | -3.0 | 4.0 | 0.0 | 3.0 | -4.0 | 0.0 |
| 5 | -4.0 | 3.0 | 0.0 | 4.0 | -3.0 | 0.0 |
| 6 | -5.0 | 0.0 | 0.0 | 5.0 | 0.0 | 0.0 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Table 3.3: Rectangular representation of animation. First line is the header.

For example, consider a 3D animation of a stick. Suppose the stick can be specified by its two endpoints $A$ and $B$. Let $x_{At}$ represent the $x$-coordinate of point $A$ at time $t$. Similarly, we define $y_{At}, z_{At}, x_{Bt}, y_{Bt}, z_{Bt}$. Then, Table 3.3 is an animation of a rotating stick.

#### 3.2.1.2 Type of Motion

To conduct a comparative analysis between imputation and inbetweening methods, a simple setup serves as a good starting point. One such setup involves a point traversing a two-dimensional space or three-dimensional space.

To assess the discrepancies between inbetweening frames and the ground truth, it is necessary to have the ground truth for all intermediate frames to be generated. Ideally, the motion should be described by exact equations, allowing for computation of any true frame with known time values. To make the experiments more meaningful, this project explores real-world motion equations established in physics. Discussions on each motion and their corresponding equations follows.

**Uniformly Accelerated Motion (UAM)** A point traversing space at constant velocity might be the simplest possible motion. Some real-life scenarios include:

- A car moving on a straight road at constant speed.
- An ice skater skating forward on the smooth ice surface.

Building upon this foundation, the assumption of constant velocity can be relaxed to incorporate non-zero acceleration. This can be observed in a lot of scenarios:

- Free falling, e.g. a ball released from a cliff.
- Projectile motion, e.g. throwing a ball into the basket.
- A car accelerating from rest.

This gives rise to a type of motions known as the uniformly accelerated motions (UAMs). These motions are characterized by its fixed acceleration (hence the term "uniform"). Their parametric vector equation is given by

$$p(t) = p(0) + v(0)t + \frac{1}{2}at^2 \tag{3.1}$$

where

- $t$ represents the time
- $p(t)$ represents the position of the point at time $t$
- $v(t)$ represents the velocity of the point at time $t$
- $a$ represents the uniform acceleration

Note that $p$, $v$, $a$ are vectors while $t$ is scalar. Also observe that when $a = 0$, Equation 3.1 is reduced to the motion equation of a point moving at fixed velocity:

$$p(t) = p(0) + v(0)t \tag{3.2}$$

**Elliptical Motion** Another natural motion to come up with is the circular motion. This type of motion can be observed in real life:

- Spin a wheel or a merry-go-round, each point on the wheel moves in a circular path
- Satellite orbiting around the Earth.

The following assumptions are made for the sake of clarity:

1. The circular motion is confined to the $xy$-plane. Mathematically, this simplification is valid as any planar orbit can be mapped onto the $xy$ plane through rotation. This assumption simplifies the process of inbetweening. Further experiments with circular motion that is not confined to the $xy$-plane might be needed to investigate the effect of this assumption.

2. The orbit of the circular motion is centered at the origin $(0, 0)$. This assumption is valid because any circular orbit can be translated to have its center at the origin. While it may simplify the process of inbetweening, the impact is likely less significant compared to the previous assumption which eliminated an entire dimension ($x$).

3. The starting point lies on the positive $y$-axis or positive $x$-axis. There are no "weird" initial phases, e.g. starting at $\frac{\pi}{6}$ radians. This assumption has minimal impact on the experiments, as the circular motion is a repeating motion so the starting point does not really matter for the purpose of inbetweening.

Under the assumptions above, the parametric equations of the circular motion is given by

$$\begin{aligned} x(t) &= R\sin(t) \\ y(t) &= R\cos(t) \end{aligned} \tag{3.3}$$

where

- $t$ represents the time
- $x(t)$ represents the $x$-coordinate of the point at time $t$
- $y(t)$ represents the $y$-coordinate of the point at time $t$
- $R$ represents the radius of the circle

Given the potential limitations of strictly circular motion, this project extends the scenario by allowing for non-uniform radii. Incorporating this variation enables

the generation of elliptical motion. The revised parametric equations are presented as follows:

$$\begin{aligned} x(t) &= R_x \sin(t) \\ y(t) &= R_y \cos(t) \end{aligned} \tag{3.4}$$

where $R_x, R_y$ are the $x, y$ radii respectively, while the remaining variables follow the definitions from Equation 3.3.

**Simple Harmonic Motion (SHM)** Another important type of motion is SHM. SHM is a specific kind of repetitive motion caused by a restoring force that is directly proportional how far the object is from a position of equilibrium, and always acts to return the object to the equilibrium position. This leads to a continuous oscillation that can be represented by a sine wave, and will persist indefinitely unless energy is lost through friction or other means. To understand SHM better, here are some real-life examples

- A pendulum swinging back and forth.
- An undamped spring holding a mass, which is pulled slightly and then released to oscillate.

In the interest of clarity, the experiments are conducted with one-dimensional SHM. Its parametric equation governing this motion is given by:

$$x(t) = A \cos(\omega t - \phi) \tag{3.5}$$

where

- $t$ represents the time
- $x(t)$ represents the position of the point at time $t$
- $A$ represents the amplitude of oscillation
- $\omega$ represents the angular frequency
- $\phi$ represents the initial phase

**Learning to Use Chopsticks** In addition to motions with physics equations above, a more complex scenario is considered. The task of learning to use chopsticks [50] involves synthesizing a kinematic trajectory for the arm and chopsticks Figure 2.1. Some parameters of a trajectory are the position and the velocity of the target (i.e. arm or chopsticks) at each time frame.

#### 3.2.1.3 Data Preparation

The motion equations presented in § 3.2.1.2 are used to generate a series of keyframes between a designated start time and end time. For the case of learning to use chopsticks, the keyframes are obtained from the demo trajectory provided by [50]. These keyframes serve the dual purpose of animation data and ground truth for evaluation.

For scenes consisting of only a point, a reasonable way to represent each keyframe is a tuple $(t, x, y, z)$ (or $(t, x, y)$ for two dimensions), where $t$ represents the time frame while $x, y, z$ are the $x, y, z$-coordinates of the point respectively.

For a complex scene such as learning to use chopsticks [50], each keyframe is a tuple that combines the time frame and relevant graphic parameters. If a parameter consists of a list of values, it is unpacked and each value is treated as a column on its own. For example, suppose there are only two graphic parameters: position $(p_x, p_y, p_z)$ and velocity $(v_x, v_y, v_z)$. Then a keyframe is specified by $(t, p_x, p_y, p_z, v_x, v_y, v_z)$.

In a typical inbetweening task, a lot of frames need to be generated between predetermined keyframes. However, the number of frames between each pair of consecutive keyframes might not be constant for the task. For a long linear motion, simply fixing the start and the end frames as keyframes could yield a good result, even though many frames need to be generated between them. On the other hand, a short period of complex motion might need more keyframes to better characterize the complex nature of the motion, even though there is only a small number of frames between pairs of consecutive keyframes. Many literature extract keyframes from a full video with the idea of keeping the feature difference between each consecutive

pairs of keyframes small [2] [7].

The sheer number of keyframe extraction methods renders evaluating them all impractical. Therefore, this study adopts an idea inspired by the evaluation of imputation methods [15] by considering the fraction of missing frames.

A complete sequence of frames (video) serves as the starting point. A predetermined fraction of frames is randomly removed, while the remaining frames are designated as keyframes for subsequent experiments. By its very nature, inbetweening necessitates the generation of at least one frame between keyframes. Consequently, a reasonable lower bound of missing fraction is approximately 50%. An obvious upper bound is 100%, but it is not meaningful to remove all the frames.

### 3.2.2 Imputation Methods

The literature review reveals several candidate imputation methods:

- Mean imputation
- MICE [25]
- KNN [46]
- Soft-Impute [28]
- missForest [44]
- GAIN [51]

The benchmarking results from Platias et al. [36] and Jäger at al. [15] suggest that MICE, KNN and missForest are the best performing methods. Note that even though the comparison by Yoon et al. [51] in their paper (which proposed GAIN) showed it outperformed MICE and missForest, the benchmarking results by Jäger at al. [15] showed otherwise. Even if the results of GAIN is comparable with the performance of other simpler methods (e.g. missForest), its nature of high-capacity deep learning model needs significantly more time [15] than conventional model to perform inbetweening, outweighing its potential benefits from high quality.

Based on the aforementioned review, the following imputation methods were selected for experiments:

- KNN
- missForest
- MICE

### 3.2.3 Inbetweening Methods

The literature review identified conventional inbetweening methods such as linear interpolation (LERP) and spherical linear interpolation (SLERP), alongside various more advanced methods.

However, the more advanced methods are difficult to implement and might be an overkill for our simple scenarios (discussed in § 3.2.1.2). Our scenarios are very simple compared to the datasets used in the papers of the fancy methods.

On the other hand, conventional methods like LERP and SLERP maintain widespread adoption due to their effectiveness in addressing simple scenarios, such as those explored in this project.

Therefore, the focus of the experiments will be on LERP and SLERP.

### 3.2.4 Evaluation

The Root Mean Square Error (RMSE) metric is employed to assess the quality of inbetweening results. When there is only one variable, the RMSE is defined as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)^2} \tag{3.6}$$

where

- $N$ is the number of values (time frames in our case)
- $x_i$ are the ground truth
- $y_i$ are the inbetweened values

For scenarios involving multiple variables, the uniform averaging method is adopted to combine the individual RMSE values. This method assigns same weight to each RMSE during the averaging process. This computation can be achieved using the scikit-learn [33]. For simplicity, the time frame variable and non-missing keyframes are included during the computation of RMSEs. Since this simplification has the same effect on all methods, the relative performance among methods will not be affected.

In addition to RMSE, the rank (with respect to RMSE) of each method is determined for each trial. This gives us a better comparison between methods with similar RMSE. For example, there might be a case where two methods have very similar RMSE, but one is consistently better than another in all trials. Using only RMSE, it is very difficult to identify this distinction.

Box plots are employed to visualize the results across settings. This choice aligns with the visualization adopted in [15], as box plots effectively reveal the distribution of the results based on quantiles, offering a clear understanding of data. This helps us understand whether one method is always better than another, or simply better when averaged.

To have a better understanding of the methods' performance, the results are presented segregated by motion type. This approach enables a clearer view of the strengths and weaknesses associated with each imputation/inbetweening method across various types of motion.

In addition, graphs are also plotted to show the execution time. Note that the unit of execution time for all relevant graphs is in seconds.

Many of the computations including RMSE are done using NumPy [13], while all plots are generated using Matplotlib [14].

## 3.3 Experiments

### 3.3.1 Setting

On a high level, there are three variables: type of motion, fraction of missing frames, and inbetweening method. For each type of motion, a series of keyframes is generated encompassing the time interval from $t = 0$ to $t = 1000$ (both inclusive) These keyframes serve the dual purpose of input data and ground truth for subsequent evaluation.

To reduce random error, each experimental setting is executed 10 times. The observed stability in method rankings across these trials for a given setting suggests little variability in relative performance.

#### 3.3.1.1 Type of Motion

**UAM** There are three parameters that define a UAM: initial position $p_0$, initial velocity $v_0$, and acceleration $a$. Each of these parameters are vectors of the same dimension.

To make the experiments more meaningful, a scenario mimicking projectile motion is incorporated. The experiment uses a two-dimensional space defined by the $x$ and $y$ coordinates. Empirically, the following set of parameters was identified to generate a parabolic motion as shown in Figure 3.1:

- initial position $p_0 = (0, 0)$
- initial velocity $v_0 = (10, 10)$
- acceleration $a = (0, -0.02)$

**Elliptical Motion** Our model of elliptical motion is extended from the circular motion by scaling the $x$ and $y$ axes. To achieve an elliptical orbit different from a circle, the $x$ and $y$ radii must employ differing values. However, an excessively large ratio between these axes would cause the resulting trajectory to be relatively flat, resembling two straight lines, deviating from the characteristic form of an ellipse.

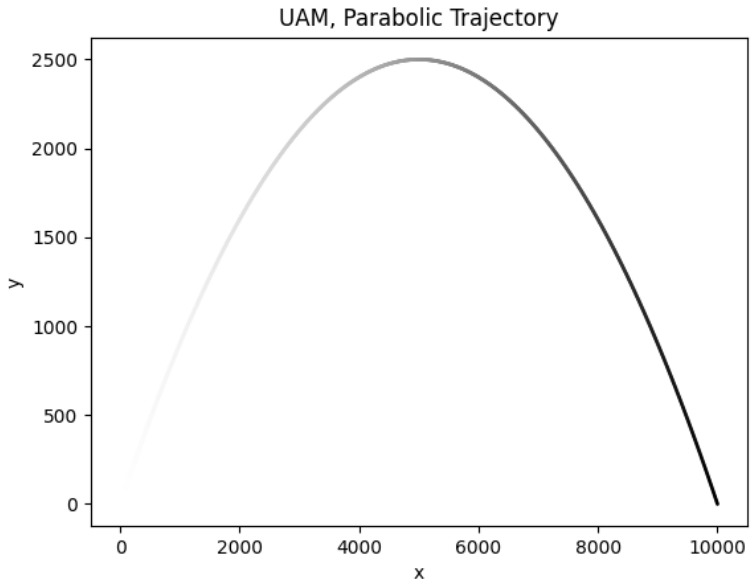


Figure 3.1: Parabolic trajectory of a projectile motion. The darkness of each point indicates the time frame (darker means larger $t$).

Keeping the considerations above in mind, the radii of $x$ and $y$ are chosen to be 2 and 3, respectively.

**SHM** The parameters that define a simple harmonic motion are: amplitude, frequency, and phase. SHM corresponds the the motion a single direction. However, instead of using only a single variable ($x$ axis), two additional columns were incorporated to represent fixed values of $y$ and $z$ axes. Empirically, it was observed that the value of frequency cannot be too large, otherwise it is hard to visualize the motion. In such cases, the point's oscillatory movement across time frames becomes too drastic. This is supported by the formula relating frequency and period of oscillation $f = \frac{1}{T}$ where $f$ and $T$ are the frequency and period of oscillation respectively.

Similarly, the experiment uses non-conventional values. The final pick for the

experiments is presented below:

$$\begin{aligned} \text{amplitude} &= 2 \\ \text{frequency} &= 0.1 \\ \text{phase} &= 3 \\ \text{fixed y} &= 4 \\ \text{fixed z} &= 5 \end{aligned} \tag{3.7}$$

**Learning to Use Chopsticks** The relevant parameters that define a keyframe are openloop-motion, vel-openloop, openloop-arm, vel-openloop-arm, motion-object, motion-chopsticks, vel-chopsticks [50]. Many of these parameters consists of multiple values (e.g. openloop-motion consists of a list of 9 values at each time frame). Multi-value parameters are unpacked and treated as described in § 3.2.1.3.

#### 3.3.1.2 Fraction of Missing Frames

As discussed in § 3.2.1, a reasonable starting value of fraction of missing frames is 50%. Working from 50% towards 100%, a reasonable step size is 10%. The experiments focus on missing fractions of 50%, 60%, 70%, 80%, and 90%. There are five values of missing fractions, which is just enough to make some inference on the performance of different methods as the number of missing frames increases.

#### 3.3.1.3 Inbetweening Method

This section commences with a discussion on the settings of imputation methods, followed by inbetweening methods.

**KNN Imputation** The `KNNImputer` class from the scikit-learn library's imputation module is utilized for imputation tasks. All default parameters associated with this class are retained.

**MissForest Imputation** The code is taken from [16], which uses Scipy [49] and scikit-learn [33]. The missforest functionality is employed without any parameter modifications.

**MICE Imputation** The `IterativeImputer` class from scikit-learn [33] is employed to generate 5 imputations. Subsequently, the average of these imputations is

computed to form the final imputation.

**Linear Interpolation (LERP)** The implementation of this method is directly derived from its definition. There are no parameters for this method.

**Spherical Linear Interpolation (SLERP)** By definition, SLERP works in the three-dimensional space. In our implementation, if there is only one variable excluding the time frame $t$, the variable is designated as $x$, and both $y$ and $z$ are consistently set at 0. Similarly, if there are only two non-time variables, they are assigned to $x$ and $y$, respectively, while $z$ remains zero throughout. Any scenario exceeding three non-time variables triggers an exception.

There is a caveat in this method. When $x = y = z = 0$, computing its spherical coordinates leads to division by zero at $\theta = \arccos \frac{z}{r}$, where $r = \sqrt{x^2 + y^2 + z^2}$. To solve this, a small value ($10^{-6}$) is added to $r$. This value is many orders of magnitude smaller than the values of $x, y, z$ in this experimental setting, so its effect on the results should be insignificant.

Note that this method is not applicable for more complex motions, e.g. learning to use chopsticks.

## 3.3.2 Results

### 3.3.2.1 Uniformly Accelerated Motion (UAM)

Begin with the UAM experimental results (Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6), the 50% missing data case (Figure 3.2) reveals that the conventional inbetweening method SLERP ranked the best, closely followed by LERP, which is also one of the most conventional methods. The imputation methods KNN, missForest and MICE ranked worse than both interpolation methods SLERP and LERP, with MICE being always the worst in all 10 trials.

Next, for settings with 60%, 70%, 80% and 90% missing data (Figure 3.3, Figure 3.4, Figure 3.5, Figure 3.6), the box plots are identical. Compared to the setting with 50% missing data, note that now LERP is the best, closely followed by SLERP. Again, the imputation methods ranked worse with MICE being the worst.

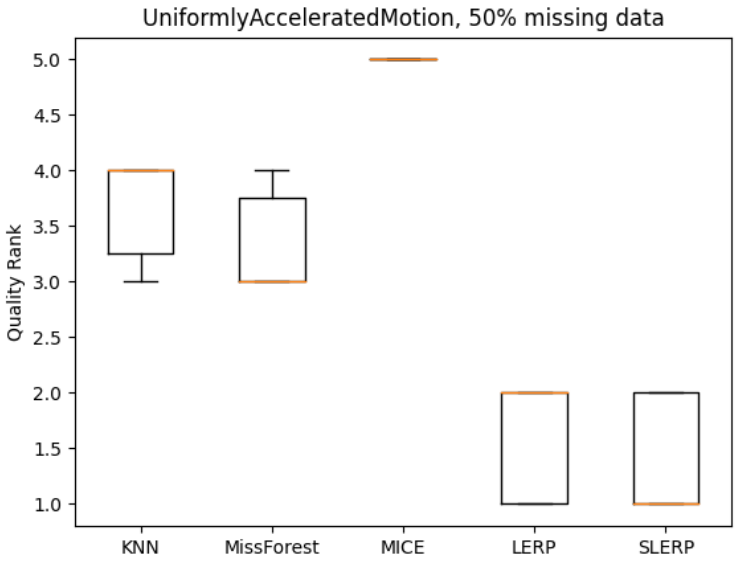


Figure 3.2: UAM quality results for 50% missing data. Lower is better.

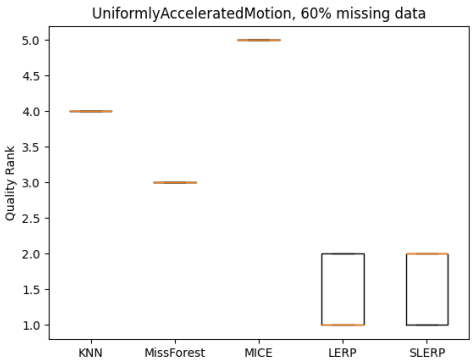


Figure 3.3: UAM quality results for 60% missing data. Lower is better.

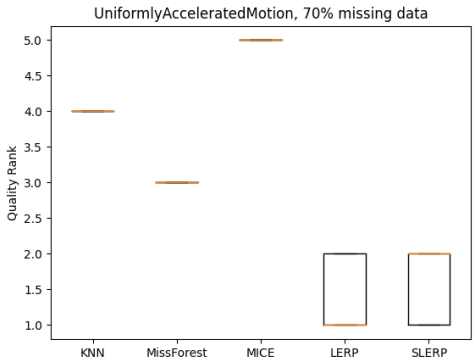


Figure 3.4: UAM quality results for 70% missing data. Lower is better.

Figure 3.7 shows that MICE generates significantly worse results compared to other methods. At low missing fraction, the quality of results by KNN and missForest are comparable to that of LERP and SLERP. However, as the fraction of missing frames increases, the quality of results by KNN and missForest becomes increasingly worse than LERP and SLERP. The quality of results by KNN also becomes increasingly worse than missForest.

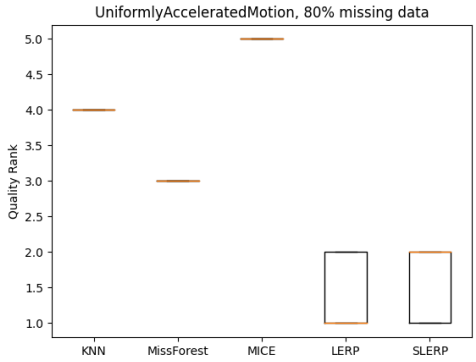


Figure 3.5: UAM quality results for 80% missing data. Lower is better.

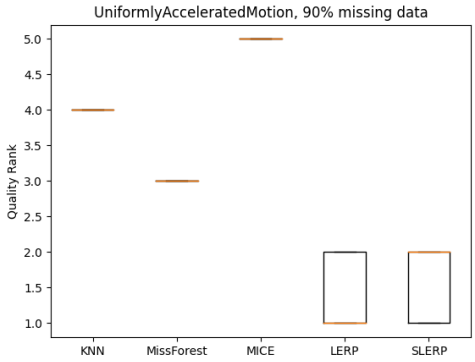


Figure 3.6: UAM quality results for 90% missing data. Lower is better.

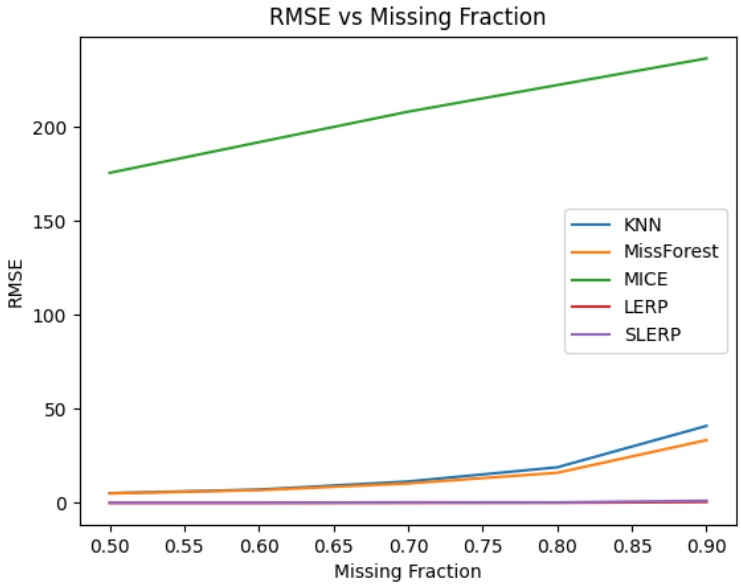


Figure 3.7: RMSE of all methods for UAM. The values at each missing fraction are averaged over all trials.

The rankings (Figure 3.8) complement Figure 3.7 by showing the distinction between methods that produce almost the same result quality. It is now clear that missForest is consistently better than KNN. On the other hand, it is also now obvious that LERP produces results with quality at least as good as SLERP from a missing fraction of 60% onwards.

A plot of execution time for all methods (Figure 3.9) shows that missForest takes

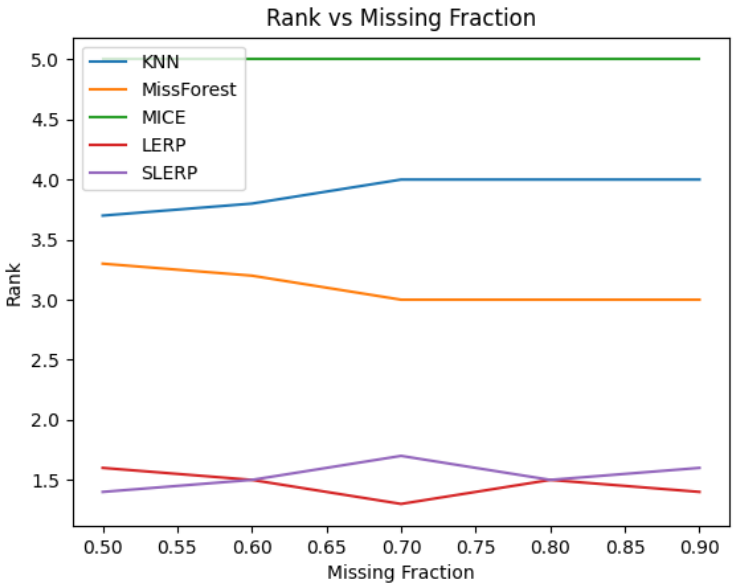


Figure 3.8: Rankings of all methods for UAM. The values at each missing fraction are averaged over all trials.

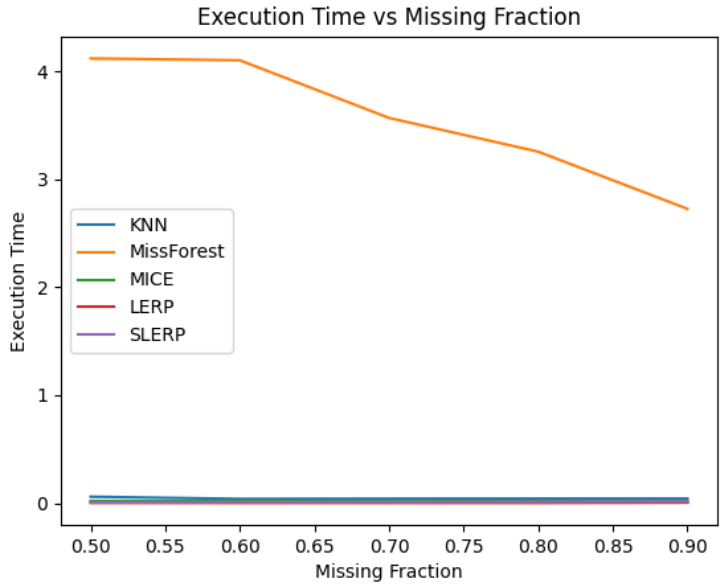


Figure 3.9: Execution time of all methods for UAM. The values at each missing fraction are averaged over all trials.

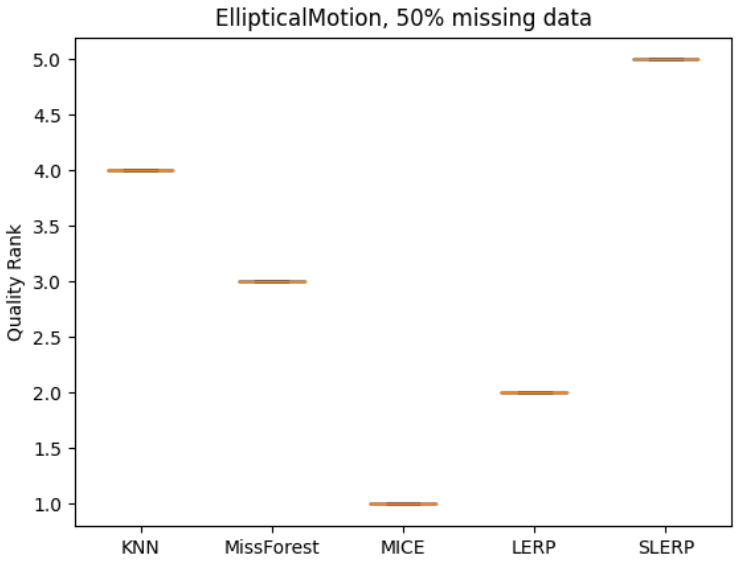


Figure 3.10: Elliptical motion quality results for 50% missing data. Lower is better.

significantly longer time to perform inbetweening than the rest of the methods.

#### 3.3.2.2 Elliptical Motion

For the experimental results of elliptical motion (Figure 3.10, Figure 3.11, Figure 3.12, Figure 3.13 and Figure 3.14), for the 50% missing data case (Figure 3.10), it is evident that MICE ranked the best in all trials. The second best method is LERP, followed by missForest and KNN at the third and fourth places respectively. Interestingly, the spherical counterpart of LERP (SLERP) ranks the worst in all trials, even though LERP ranks much higher at the second place.

With 10% more data being missing (Figure 3.11), MICE is still the best method. Now, KNN performs better than missForest. missForest is now slightly worse than LERP. The relative performance of SLERP is still consistently worst, same as that of 50% missing data. Notably, KNN went from rank 4 to 2.5, having the largest improvement in rank.

This time, the rankings are less consistent across trials. KNN, missForest and LERP have large spreads. MICE and SLERP have no spread at all (same rank

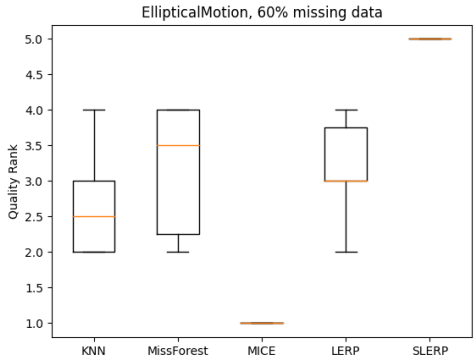


Figure 3.11: Elliptical motion quality results for 60% missing data. Lower is better.

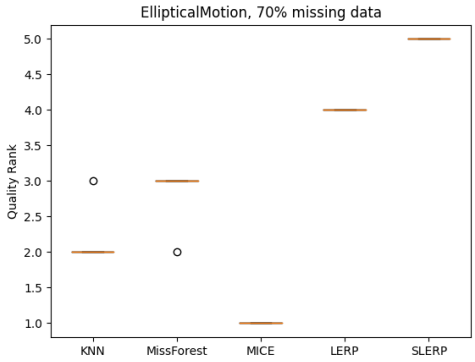


Figure 3.12: Elliptical motion quality results for 70% missing data. Lower is better.

in every trial). The consistency of relative ranks over different trials reduced significantly compared to the case of 50% missing data.

When 70% data is removed (Figure 3.12), MICE is still the best. On the other hand, LERP and SLERP ranked fourth and firth respectively in every trial. KNN and missForest are ranked second and third respectively. LERP performed better than missForest when 60% data were missing, but now missForest performs better than LERP. This plot suggests that imputation methods perform better than LERP and SLERP in this setting.

In terms of consistency of rankings over many trials, all methods become more consistent.

With even more data (Figure 3.13, 80% missing data), all of the relative rankings remain the same (MICE, KNN, missForest, LERP, SLERP). All methods become always consistent in their rankings across trials.

At the most missing data in our experiments (90%, Figure 3.14), the box plots look almost the same as in the case of 80% missing data. The only differences are missForest occasionally performing slightly worse while LERP occasionally performing slightly better.

The plot of RMSE Figure 3.15 shows that MICE is consistently the best, while SLERP is consistently the worst. The plot of rankings Figure 3.16 shows that

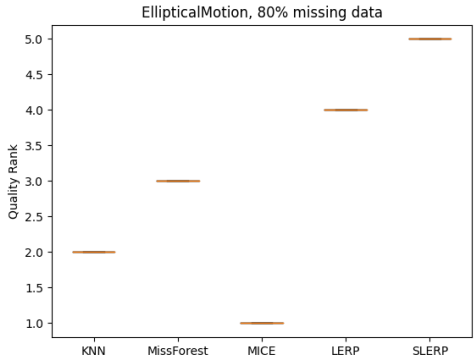


Figure 3.13: Elliptical motion quality results for 80% missing data. Lower is better.

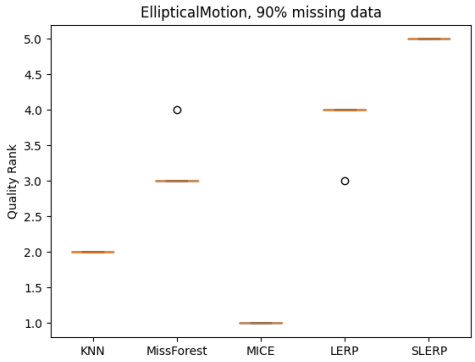


Figure 3.14: Elliptical motion quality results for 90% missing data. Lower is better.

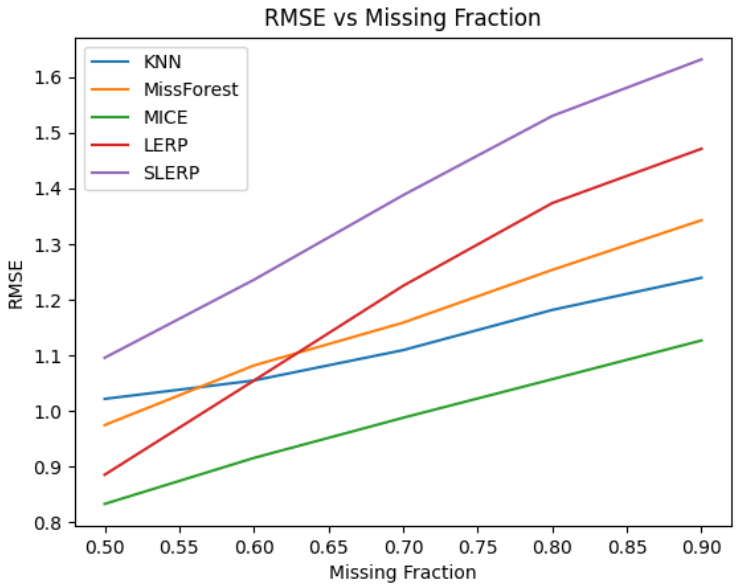


Figure 3.15: RMSE of all methods for elliptical motion. The values at each missing fraction are averaged over all trials.

LERP becomes worse relatively as the missing fraction, though never as worse as SLERP. Besides, KNN becomes better relatively as the missing fraction increases, but is never as good as MICE. At high levels of missing frames, it is clear that the imputation methods (MICE, KNN, missForest) generate better intermediate frames than conventional interpolation methods (LERP, SLERP).

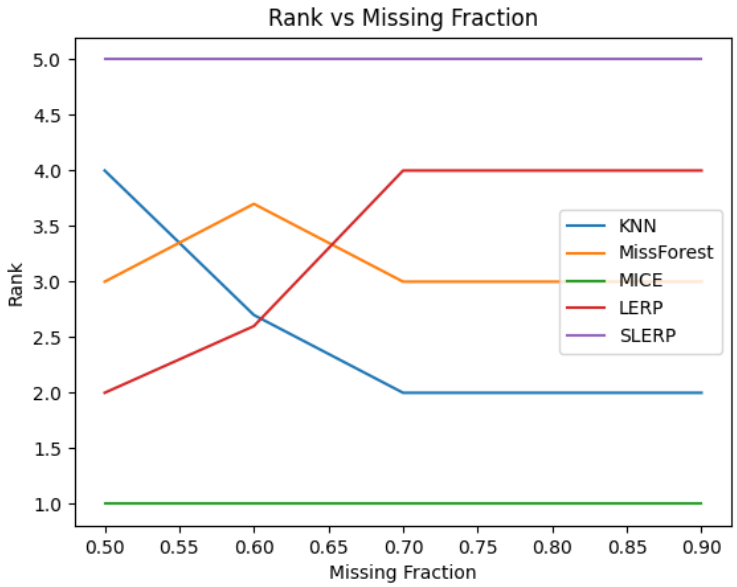


Figure 3.16: Rankings of all methods for elliptical motion. The values at each missing fraction are averaged over all trials.

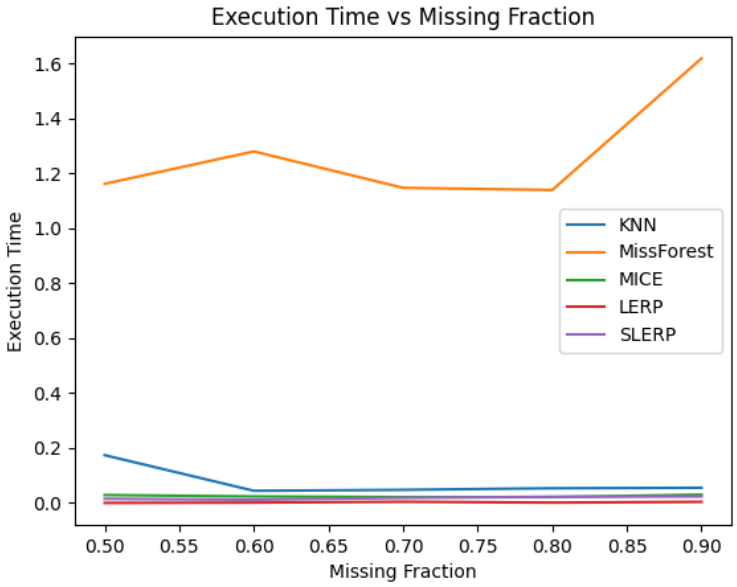


Figure 3.17: Execution time of all methods for elliptical motion. The values at each missing fraction are averaged over all trials.

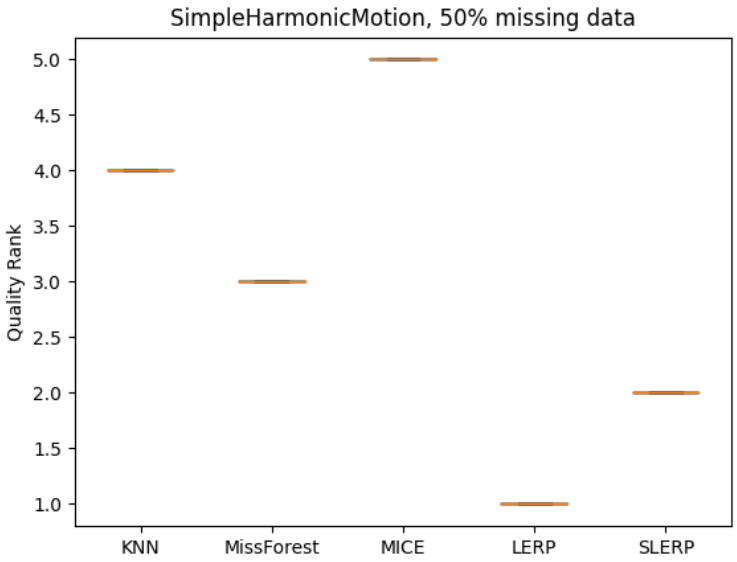


Figure 3.18: Simple harmonic motion quality results for 50% missing data. Lower is better.

In terms of execution time (Figure 3.17), missForest is significantly worse than the other methods. The execution time of KNN is much longer than LERP, SLERP and MICE when 50% of the frames are missing, but has a lower execution time at 60% missing data and beyond. MICE always have a short execution time which is comparable to the execution times of LERP and SLERP.

#### 3.3.2.3 Simple Harmonic Motion (SHM)

When 50% of data are missing for SHM, the box plots for all methods (Figure 3.18) showed very high consistencies of relative ranks over trials. The ranking of all methods turned out to be the same for all trials. The inbetweening quality ranked from best to worst are: LERP, SLERP, missForest, KNN and MICE. The rankings showed that conventional interpolation methods (LERP and SLERP) outperform the imputation methods, for SHM.

Interestingly, the box plots of quality ranks for fractions of missing data from 50% to 80% are identical (Figure 3.18, Figure 3.19, Figure 3.20, Figure 3.21). This implies the relative performances of all methods remain invariant to the fraction of

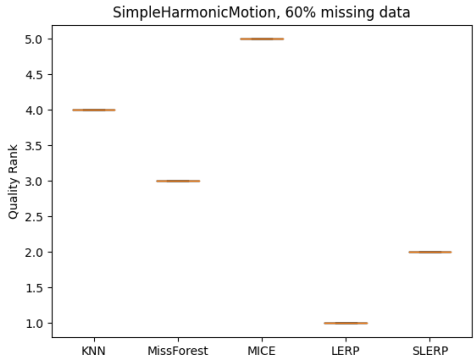


Figure 3.19: SHM quality results for 60% missing data. Lower is better.

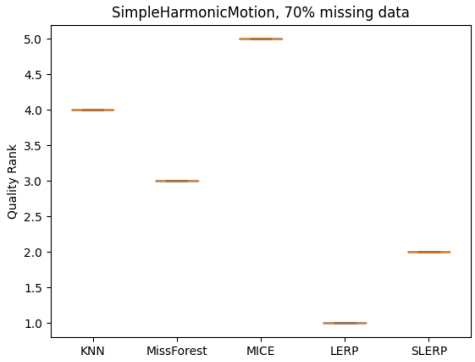


Figure 3.20: SHM quality results for 70% missing data. Lower is better.

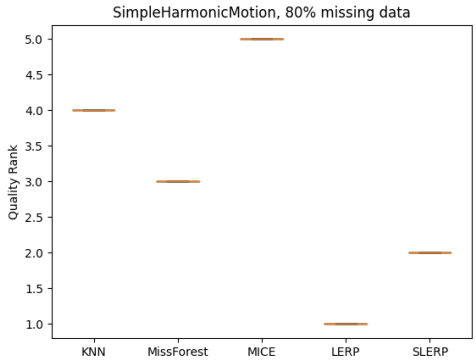


Figure 3.21: SHM quality results for 80% missing data. Lower is better.

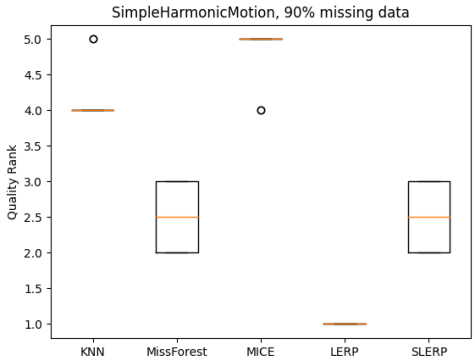


Figure 3.22: SHM quality results for 90% missing data. Lower is better.

missing data and the randomness of removed data.

For the case of 90% missing data (Figure 3.22), the box plots finally look different. While the rankings of LERP, MICE and KNN remain the same as that in 50%-80% missing data, the methods missForest and SLERP are now equally good, instead of missForest being consistently worse than SLERP. There is also now a spread of rankings for all methods except LERP, indicating less consistency across trials.

The graphs of RMSE (Figure 3.23) and rankings (Figure 3.24) consistently show the following rankings across different fraction of missing frames (from best to worst): LERP, SLERP, missForest, KNN, MICE. Interestingly, the RMSE of MICE seems to increase linearly with missing fraction (Figure 3.23), but the RMSEs of all other

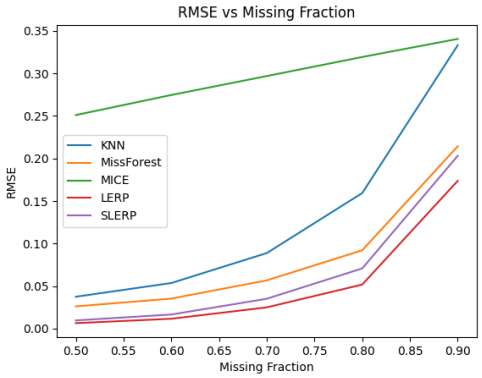


Figure 3.23: RMSE of all methods for SHM. The values at each missing fraction are averaged over all trials.

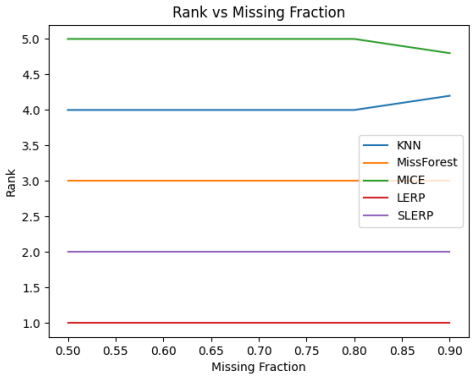


Figure 3.24: Rankings of all methods for SHM. The values at each missing fraction are averaged over all trials.

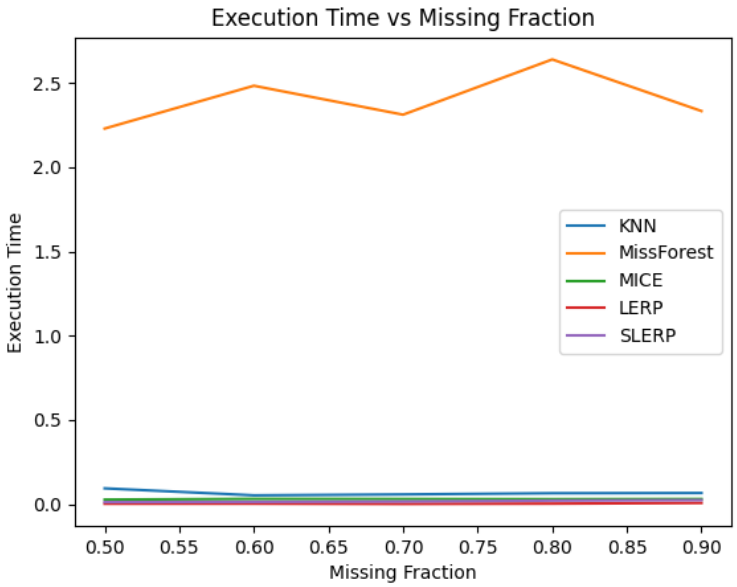


Figure 3.25: Execution time of all methods for SHM. The values at each missing fraction are averaged over all trials.

methods seem to increase faster than linearly. In particular, the RMSE of KNN increases the fastest with missing fraction, almost catching up to the RMSE of MICE at a missing fraction of 90%.

On the other hand, the graph of execution time (Figure 3.25) shows missForest takes a significantly longer amount of execution time for inbetweening. The execution

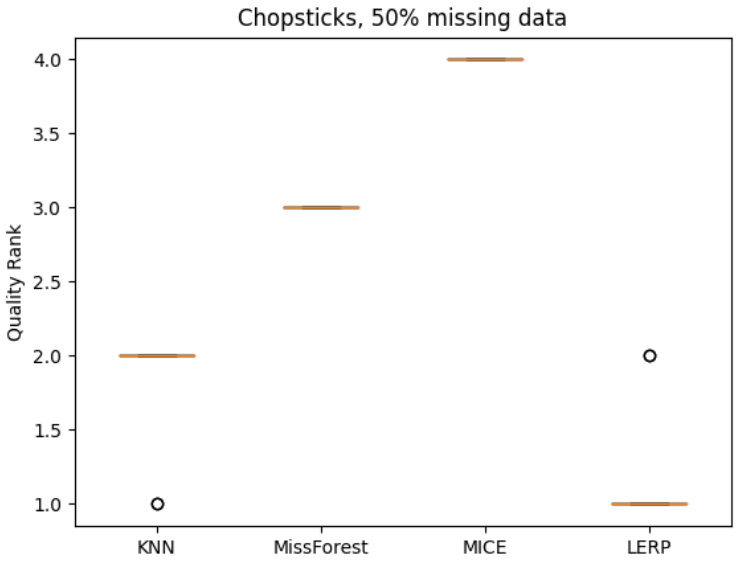


Figure 3.26: Quality results of chopsticks animation for 50% missing data. Lower is better.

time of all other methods are very similar, with KNN taking slightly longer time than MICE, LERP and SLERP.

#### 3.3.2.4 Learning to Use Chopsticks

The experimental results of all fractions of missing frames (Figure 3.26, Figure 3.27, Figure 3.28, Figure 3.29 and Figure 3.30) reveals that the conventional inbetweening method LERP ranked the best, followed by KNN, missForest and MICE, in that order. The rankings are highly consistent, with the exception that KNN performs better than LERP in one out of ten trials at low missing fractions (50%, 60%).

The graphs of RMSE (Figure 3.31) and rankings (Figure 3.32) consistently show the following rankings across different fraction of missing frames (from best to worst): LERP, KNN, missForest, MICE. The RMSE of all methods increase at a similar rate with respect to the fraction of missing frames.

On the other hand, the graph of execution time (Figure 3.33) shows missForest takes a significantly longer amount of execution time for inbetweening. The execution

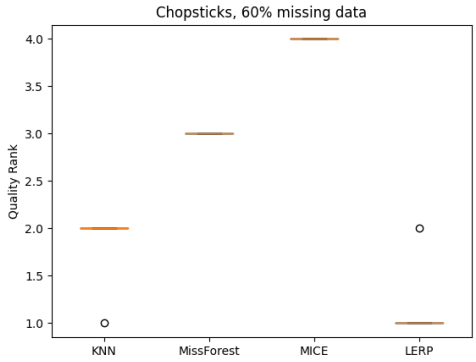


Figure 3.27: Quality results of chopsticks animation for 60% missing data. Lower is better.

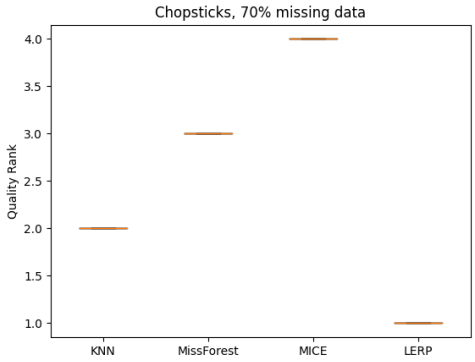


Figure 3.28: Quality results of chopsticks animation for 70% missing data. Lower is better.

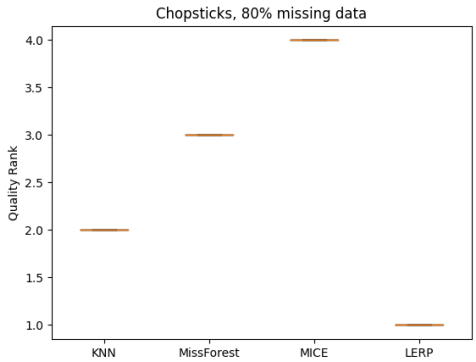


Figure 3.29: Quality results of chopsticks animation for 80% missing data. Lower is better.

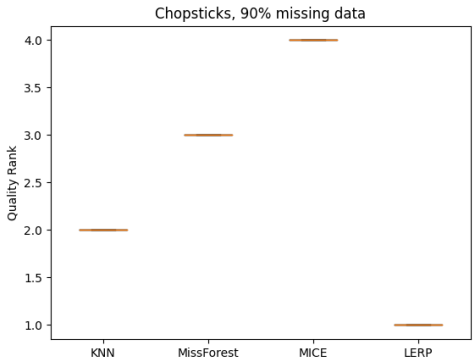


Figure 3.30: Quality results of chopsticks animation for 90% missing data. Lower is better.

time of missForest decreases rapidly as the fraction of missing frames increases. The execution time for all other methods are similar.

## 3.4 Discussion & Analysis

This analysis prioritizes settings with the highest missing fraction (90%). It is not uncommon for a practical inbetweening task to generate 20 frames between keyframes (also known as keyframe interval). With our setting of highest missing

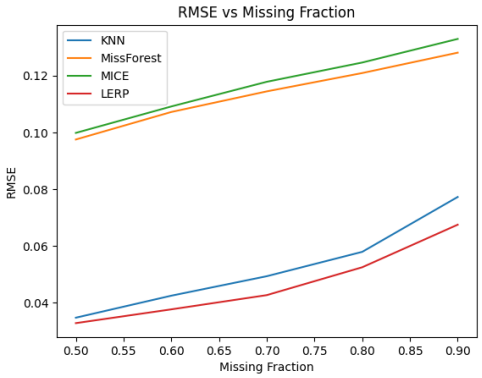


Figure 3.31: RMSE of all methods for learning to use chopsticks. The values at each missing fraction are averaged over all trials.

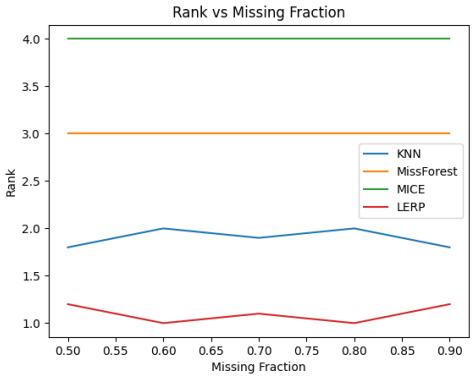


Figure 3.32: Rankings of all methods for learning to use chopsticks. The values at each missing fraction are averaged over all trials.

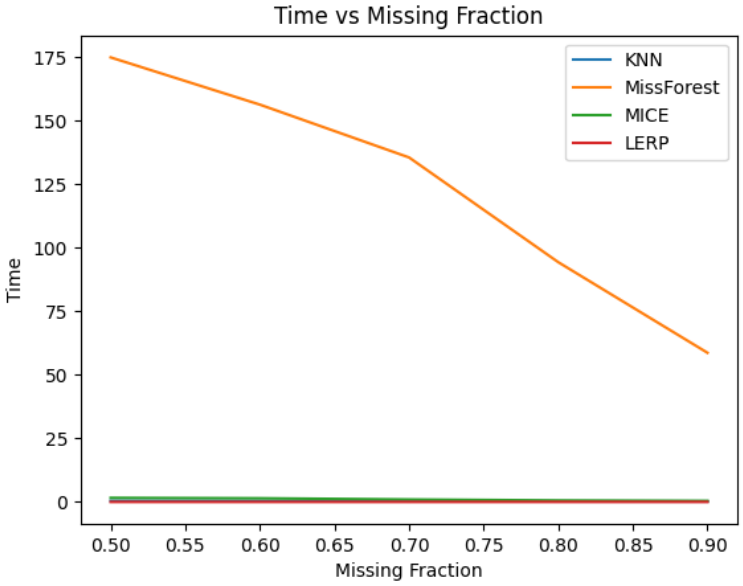


Figure 3.33: Execution time of all methods for learning to use chopsticks. The values at each missing fraction are averaged over all trials.

fraction, the keyframe interval is only $\frac{1}{1-90\%} = 10$. This smaller keyframe interval is just sufficient for less demanding inbetweening tasks. Hence, it is more meaningful to focus our discussion and analysis on the settings with 90% missing data.

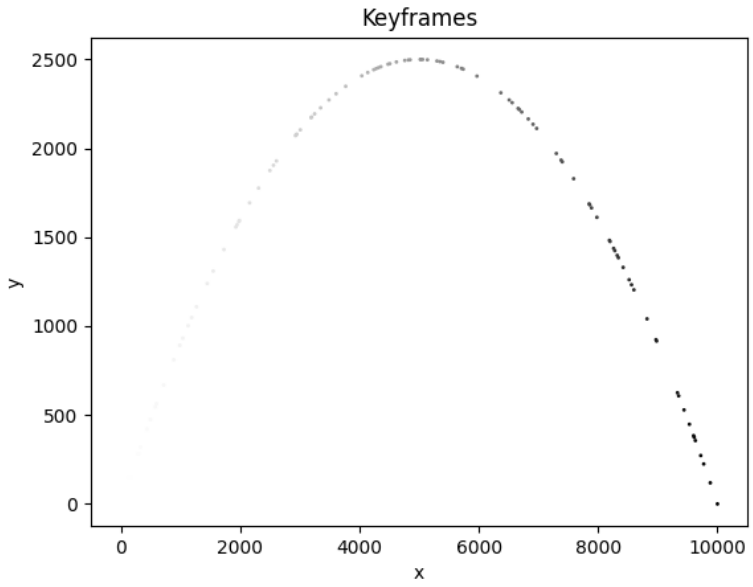


Figure 3.34: Keyframes of UAM in one of the trials at a missing fraction of 90%. The darkness of each point indicates the time frame (darker means larger t).

### 3.4.1 Uniformly Accelerated Motion (UAM)

Conventional methods for inbetweening (LERP, SLERP) yield higher quality results than imputation methods (KNN, missForest, MICE). In particular, LERP is the best (when missing fraction is 60% and above) while MICE is the worst.

One reason why conventional methods (LERP, SLERP) are better than imputation methods is because there are sufficient keyframes for them to interpolate. This can be seen from Figure 3.34, where the parabolic trajectory is still clearly visible even when 90% of frames are missing.

MissForest has a very long execution time compared to all other methods, which all have very similar short execution time. Overall, LERP is the best for UAM with its best quality and extremely short execution time.

### 3.4.2 Elliptical Motion

At high levels of missing data (70%, 80%, 90%), the following ranking (Figure 3.16) is clear (best to worst): MICE, KNN, missForest, LERP, SLERP. Observe that the

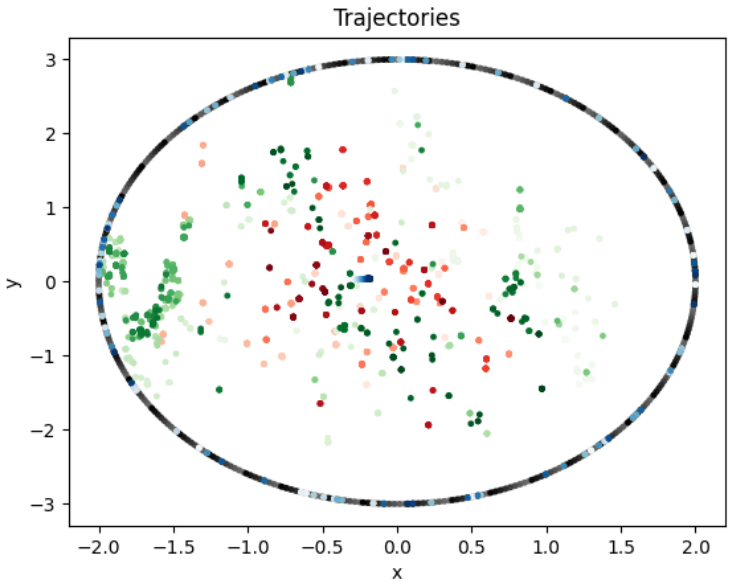


Figure 3.35: Trajectories representing the results of inbetweening by imputation methods for elliptical motion, at 90% missing data. The darkness of each point indicates the time frame (darker means larger $t$). The true elliptical motion is colored in grey. The imputations by KNN are colored in red. The imputations by missForest are colored in green. The imputations by MICE are colored in blue.

imputation methods perform better than conventional methods. MICE is consistently the best for all missing fractions experimented, while SLERP is consistently the worst. KNN and missForest becomes better than LERP at high levels of missing data. One explanation could be that relationships between variables are harder to recover for simple methods such as LERP and SLERP when there are too much missing data.

One trial result of inbetweening at high level of missing data (90%) are shown at Figure 3.35 and Figure 3.36. This trial result is likely representative because of the high consistency observed in rankings (Figure 3.14).

Observe that in the results of imputation methods (Figure 3.35), most points actually lie on the elliptical orbit. In contrast, most intermediate frames of conventional inbetweening methods (LERP and SLERP) are more scattered and do not follow the elliptical orbit (Figure 3.36). This observation supports the rankings

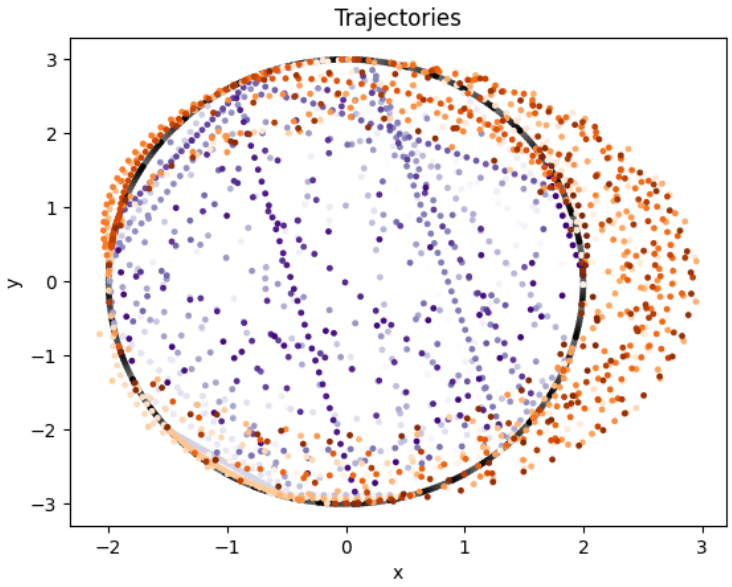


Figure 3.36: Trajectories representing the results of inbetweening by conventional interpolation methods for elliptical motion, at 90% missing data. The darkness of each point indicates the time frame (darker means larger $t$). The true elliptical motion is colored in grey. The results of LERP are colored in purple. The results of SLERP are colored in orange.

where imputation methods (KNN, missForest, MICE) outperform interpolation methods (LERP, SLERP).

A closer inspection of Figure 3.35 reveals that most blue points lie on the elliptical orbit. This observation agrees with the ranking that MICE is the best (Figure 3.14).

A closer inspection of Figure 3.36 discloses a greater dispersion of the SLERP results outside of the area enclosed by the elliptical orbit. Conversely, the results of LERP are more enclosed within the orbit. This observation supports the higher ranking of LERP than SLERP, as reported in Figure 3.14.

One reason of conventional methods having worse quality can be inferred from their trajectories in Figure 3.36. The purple dots are the results of LERP, which interpolates the keyframes directly. When there are a lot of missing frames, it is possible for consecutive keyframes to be too far apart, skipping certain important patterns. In this case, a large part of the arc is skipped occasionally, as can be seen

by the purple points forming straight lines across the elliptical orbit. In comparison, imputation methods are better suited in this scenario, as they might infer the arc from another repetition of the elliptical motion.

On the other hand, the resulting frames of SLERP are concentrated at the right. This deviation is caused by SLERP repeatedly learning the wrong direction of rotation, thinking it is the shortest path in spherical coordinates. The asymmetry is likely caused by the angular frequency of elliptical motion, which happens to favour a certain direction for shortest path in spherical coordinates.

On a side note, comparing the RMSEs for all motions (Figure 3.7, Figure 3.15, Figure 3.23), the disparity between the best method and the worst performing methods appears minimal elliptical motion. This smaller gap suggests that the best method (MICE) is only slightly better than the rest of the methods.

In terms of execution time (Figure 3.17), missForest shows terrible execution time compared to all other methods, same as the case of UAM (Figure 3.9). All other methods took similar time to complete the inbetweening task.

Notably, KNN has a distinctively longer execution time at a missing fraction of 50%. One might hypothesize that with longer execution time, KNN should perform better with 50% missing frames compared to higher number of frames. However, Figure 3.16 does not support this hypothesis. The explanation for this observation is unclear and might need further analysis.

Overall, MICE performs better than conventional methods (LERP, SLERP) consistently (Figure 3.15) while having a comparable execution time (Figure 3.17). Thus, it might be worth to consider using MICE for elliptical motions.

### 3.4.3 Simple Harmonic Motion (SHM)

Figure 3.23 and Figure 3.24 clearly show conventional methods (LERP and SLERP) being superior than imputation methods (KNN, missForest and MICE) in terms of quality. Furthermore, the execution time (Figure 3.25) of LERP and SLERP are the shortest two. Hence, conventional methods are better choices than imputation methods. Specifically, LERP is a better choice than SLERP since it

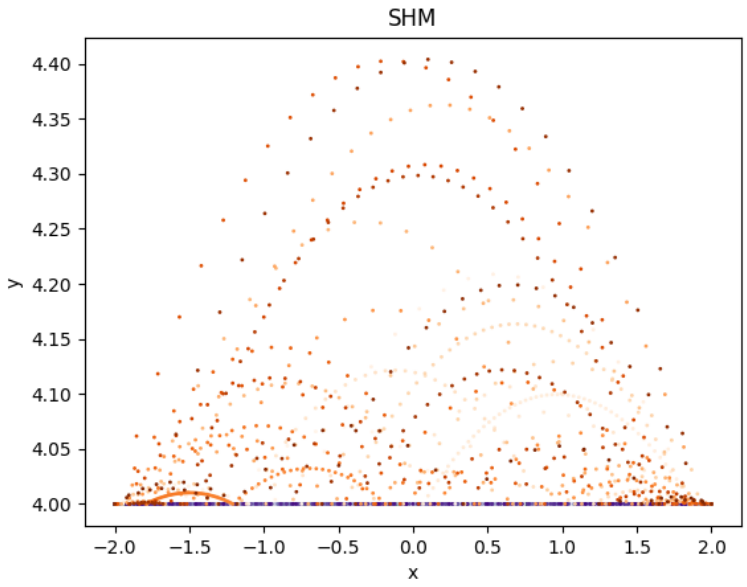


Figure 3.37: Trajectories representing the results of inbetweening by all methods for SHM, at 90% missing data. The darkness of each point indicates the time frame (darker means larger $t$). The true elliptical motion is colored in grey. The results of KNN are in red. The results of missForest are in green. The results of MICE are in blue. The results of LERP are colored in purple. The results of SLERP are colored in orange. All results except those by SLERP lie on the same line so they are hard to see.

consistently ranks the best.

One reason why LERP is much better than other methods for SHM is because of the straight trajectory. By design, LERP interpolates in a straight line between variables. This interpolation method is better than SLERP, which interpolates in the spherical coordinates.

The results produced by SLERP form an arc (Figure 3.37) as they move along the line formed by SHM. Examining solely the $x$ axis, it can be observed that the $x$-coordinates of the results of SLERP are always between the $x$-coordinates of both keyframes, similar to LERP. Hence, the RSME on the $x$ axis is very small. Along the $y$ axis, the RSME is not negligible. However, observe that the trajectory interpolated by SLERP seldom travel very far above the line formed by SHM. The magnitude of divergence in the $y$ axis is relatively small, as can be seen from the

vertical scale, which is roughly one-tenth of the horizontal scale.

On the other hand, imputation methods perform a lot worse, as there is no relationship between the $x$ and $y$ variables. Having one less variable to rely on compared to other motions, imputation methods perform badly. In the event that they are unable to learn the relationship between the time frame $t$ and $x$, their imputations are similar to a random guess along the $x$-axis.

In terms of execution time, again, missForest is the worst, while all other methods have very similar execution time. Overall, LERP remains the best method for SHM with its best quality and extremely short execution time.

### 3.4.4 Learning to Use Chopsticks

Figure 3.31 and Figure 3.32 show conventional method LERP gives the highest quality results. The quality of KNN is slightly worse than LERP, but the other two imputation methods (missForest and MICE) perform significantly worse than LERP and KNN.

In terms of execution time, LERP is the best. The high-dimensionality of this complex case causes missForest to take significantly longer for imputation. Some tweaking might be possible to improve the execution time of missForest (e.g. decrease the number of trees in the forest). However, given the huge gap, it is unlikely for missForest to achieve a comparable execution time with LERP and other methods.

Overall, LERP is better than imputation methods for a complex scenario such as learning to use chopsticks. KNN is also a good choice, being slightly worse than LERP.

# Chapter 4

# Conclusion

In the first part of this project, an investigation of Reinforcement Learning methods in Character Animation is conducted. Many literature are reviewed and reproduced, including DeepPhase [43], Bailando [41] and learning to use chopsticks [50].

Feedback for the first part of this project inspired a novel idea of framing the inbetweening task as a data imputation problem. This reformulation enables the application of imputation methods on inbetweening from a domain external to animation. A comparative analysis was conducted on the results of inbetweening for three simple motions (UAM, elliptical motion, SHM), and a complex motion involving a humanoid arm (learning to use chopsticks). Our Python [48] code can be found at `https://github.com/GnohChengYi/XFC4101`.

The findings demonstrate that imputation methods perform better than conventional inbetweening methods for elliptical motion. The imputation methods MICE and KNN perform better than conventional inbetweening methods LERP and SLERP at high level of missing frames for elliptical motion, without having too much degradation in execution time. Our experiments show that LERP is the best method to perform inbetweening of keyframes for UAM and SHM.

For the more complex scenario of learning to use chopsticks, it is demonstrated that the conventional method LERP is the best, followed by the imputation method KNN.

While imputation methods are useful, there are some limitations. While the application of imputation methods yielded superior results for elliptical motion

compared to conventional inbetweening techniques, LERP and SLERP are still better for UAM and SHM. In addition, because of more complicated algorithms, good imputation methods usually require a longer execution time.

Future work could consider hyperparameter tuning for high quality results. Other possible further experiments include:

- more complex dataset such as AIST++ [23] or HumanML3D [11]; or
- other imputation methods such as GAIN [51] or Forest-VP [16]; or
- better inbetweening methods using phase manifolds [42] or $\Delta$-interpolator [31]

# Bibliography

[1] K. Aberman, P. Li, D. Lischinski, O. Sorkine-Hornung, D. Cohen-Or, and B. Chen, "Skeleton-aware networks for deep motion retargeting", *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 62–1, 2020.

[2] M. K. Asha Paul, J. Kavitha, and P. A. Jansi Rani, "Key-frame extraction techniques: A review", *Recent Patents on Computer Science*, vol. 11, no. 1, pp. 3–16, 2018.

[3] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, "Drecon: Data-driven responsive control of physics-based characters", *ACM Transactions On Graphics (TOG)*, vol. 38, no. 6, pp. 1–11, 2019.

[4] A. A. Beserra, R. M. Kishi, and R. Goularte, "Evaluating early fusion operators at mid-level feature space", in *Proceedings of the Brazilian Symposium on Multimedia and the Web*, 2020, pp. 113–120.

[5] R. S. Burt, "A note on missing network data in the general social survey", *Social Networks*, vol. 9, no. 1, pp. 63–73, 1987.

[6] Y. Cai, Y. Wang, Y. Zhu, T.-J. Cham, J. Cai, J. Yuan, J. Liu, C. Zheng, S. Yan, H. Ding, *et al.*, "A unified 3d human motion synthesis model via conditional variational auto-encoder", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 645–11 655.

[7] J. Calic and E. Izuierdo, "Efficient key-frame extraction and video analysis", in *Proceedings. International Conference on Information Technology: Coding and Computing*, IEEE, 2002, pp. 28–33.

[8] Y. Chebotar, K. Hausman, Z. Su, G. S. Sukhatme, and S. Schaal, "Self-supervised regrasping using spatio-temporal tactile features and reinforcement learning", in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 1960–1966.

[9] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[10] R. F. J. Dossa, S. Huang, S. Ontañón, and T. Matsubara, "An empirical investigation of early stopping optimizations in proximal policy optimization", *IEEE Access*, vol. 9, pp. 117 981–117 992, 2021.

[11] C. Guo, S. Zou, X. Zuo, S. Wang, W. Ji, X. Li, and L. Cheng, "Generating diverse and natural 3d human motions from text", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 5152–5161.

[12] A. Haarbach, T. Birdal, and S. Ilic, "Survey of higher order rigid body motion interpolation methods for keyframe animation and continuous-time trajectory estimation", in *2018 International Conference on 3D Vision (3DV)*, IEEE, 2018, pp. 381–389.

[13] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy", *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: `https://doi.org/10.1038/s41586-020-2649-2`.

[14] J. D. Hunter, "Matplotlib: A 2d graphics environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[15] S. Jäger, A. Allhorn, and F. Bießmann, "A benchmark for data imputation methods", *Frontiers in big Data*, vol. 4, p. 693 674, 2021.

[16] A. Jolicoeur-Martineau, K. Fatras, and T. Kachman, "Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees", *arXiv preprint arXiv:2309.09968*, 2023.

[17] K. Karunratanakul, K. Preechakul, S. Suwajanakorn, and S. Tang, "Guided motion diffusion for controllable human motion synthesis", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2151–2162.

[18] V. E. Kremer, "Quaternions and slerp", in *Embots. dfki. de/doc/seminar ca/Kremer Quaternions. pdf*, 2008.

[19] A. Kwiatkowski, E. Alvarado, V. Kalogeiton, C. K. Liu, J. Pettré, M. van de Panne, and M.-P. Cani, "A survey on reinforcement learning methods in character animation", in *Computer Graphics Forum*, Wiley Online Library, vol. 41, 2022, pp. 613–639.

[20] J. Lee, J. Won, and J. Lee, "Crowd simulation by deep reinforcement learning", in *Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2018, pp. 1–7.

[21] S. Lee, S. Lee, Y. Lee, and J. Lee, "Learning a family of motor skills from a single motion clip", *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pp. 1–13, 2021.

[22] J. Li, R. Villegas, D. Ceylan, J. Yang, Z. Kuang, H. Li, and Y. Zhao, "Task-generic hierarchical human motion prior using vaes", in *2021 International Conference on 3D Vision (3DV)*, IEEE, 2021, pp. 771–781.

[23] R. Li, S. Yang, D. A. Ross, and A. Kanazawa, "Learn to dance with aist++: Music conditioned 3d dance generation", 2021. arXiv: `2101.08779 [cs.CV]`.

[24] H. Y. Ling, F. Zinno, G. Cheng, and M. Van De Panne, "Character controllers using motion vaes", *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 40–1, 2020.

[25] R. Little and D. Rubin, "Multiple imputation for nonresponse in surveys", *John Wiley & Sons, Inc.. doi*, vol. 10, p. 9 780 470 316 696, 1987.

[26] I. Loi, E. I. Zacharaki, and K. Moustakas, "Machine learning approaches for 3d motion synthesis and musculoskeletal dynamics estimation: A survey", *IEEE transactions on visualization and computer graphics*, 2023.

[27] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning", in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 6252–6259.

[28] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices", *The Journal of Machine Learning Research*, vol. 11, pp. 2287–2322, 2010.

[29] S. A. Mistler and C. K. Enders, "A comparison of joint model and fully conditional specification imputation for multilevel missing data", *Journal of Educational and Behavioral Statistics*, vol. 42, no. 4, pp. 432–466, 2017.

[30] L. Mourot, L. Hoyet, F. Le Clerc, F. Schnitzler, and P. Hellier, "A survey on deep learning for skeleton-based human animation", in *Computer Graphics Forum*, Wiley Online Library, vol. 41, 2022, pp. 122–157.

[31] B. N. Oreshkin, A. Valkanas, F. G. Harvey, L.-S. Ménard, F. Bocquelet, and M. J. Coates, "Motion in-betweening via deep Δ-interpolator", *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[32] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation", in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 3828–3834.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine, "Sfv: Reinforcement learning of physical skills from videos", *ACM Transactions On Graphics (TOG)*, vol. 37, no. 6, pp. 1–14, 2018.

[35] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, "Amp: Adversarial motion priors for stylized physics-based character control", *ACM Transactions on Graphics (ToG)*, vol. 40, no. 4, pp. 1–20, 2021.

[36] C. Platias and G. Petasis, "A comparison of machine learning methods for data imputation", in *11th Hellenic Conference on Artificial Intelligence*, 2020, pp. 150–159.

[37] R. Po, W. Yifan, V. Golyanik, K. Aberman, J. T. Barron, A. H. Bermano, E. R. Chan, T. Dekel, A. Holynski, A. Kanazawa, *et al.*, "State of the art on diffusion models for visual computing", *arXiv preprint arXiv:2310.07204*, 2023.

[38] J. Qin, Y. Zheng, and K. Zhou, "Motion in-betweening via two-stage transformers." *ACM Trans. Graph.*, vol. 41, no. 6, pp. 184–1, 2022.

[39] D. B. Rubin, "Multiple imputation after 18+ years", *Journal of the American statistical Association*, vol. 91, no. 434, pp. 473–489, 1996.

[40] M. Shi, K. Aberman, A. Aristidou, T. Komura, D. Lischinski, D. Cohen-Or, and B. Chen, "Motionet: 3d human motion reconstruction from monocular video with skeleton consistency", *ACM Transactions on Graphics (TOG)*, vol. 40, no. 1, pp. 1–15, 2020.

[41] L. Siyao, W. Yu, T. Gu, C. Lin, Q. Wang, C. Qian, C. C. Loy, and Z. Liu, "Bailando: 3d dance generation by actor-critic gpt with choreographic memory", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 050–11 059.

[42] P. Starke, S. Starke, T. Komura, and F. Steinicke, "Motion in-betweening with phase manifolds", *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 6, no. 3, pp. 1–17, 2023.

[43] S. Starke, I. Mason, and T. Komura, "Deepphase: Periodic autoencoders for learning motion phase manifolds", *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–13, 2022.

[44] D. J. Stekhoven and P. Bühlmann, "Missforest—non-parametric missing value imputation for mixed-type data", *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 2012.

[45] Y. Sun, X. Yuan, W. Liu, and C. Sun, "Model-based reinforcement learning via proximal policy optimization", in *2019 Chinese Automation Congress (CAC)*, IEEE, 2019, pp. 4736–4740.

[46] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for dna microarrays", *Bioinformatics*, vol. 17, no. 6, pp. 520–525, 2001.

[47] S. Van Buuren and K. Groothuis-Oudshoorn, "Mice: Multivariate imputation by chained equations in r", *Journal of statistical software*, vol. 45, pp. 1–67, 2011.

[48] G. Van Rossum and F. L. Drake Jr, "Python tutorial", Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[49] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore,

J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020.

[50] Z. Yang, K. Yin, and L. Liu, "Learning to use chopsticks in diverse gripping styles", *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–17, 2022.

[51] J. Yoon, J. Jordon, and M. Schaar, "Gain: Missing data imputation using generative adversarial nets", in *International conference on machine learning*, PMLR, 2018, pp. 5689–5698.

[52] Y. Yuan and K. Kitani, "Residual force control for agile human behavior imitation and extended motion synthesis", *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 763–21 774, 2020.

[53] Y. Zhang, L. Ke, A. Deshpande, A. Gupta, and S. Srinivasa, "Cherry-picking with reinforcement learning", *arXiv preprint arXiv:2303.05508*, vol. 15, 2023.