High-Performance Data-Driven Control of Physically-Based Human Characters

by

Kevin Bergamin

Department of Mechanical Engineering
McGill University, Montreal
August 2020

A thesis submitted to McGill University
in partial fulfillment of the requirements of the degree of
Master of Engineering

Supervisor:

Professor James Richard Forbes

Kevin Bergamin
kevin.bergamin@mail.mcgill.ca

Abstract

This thesis describes a system to enable responsive user guided control of a physically simulated human character. The control system is meant to be robust to disturbances while also producing movements that are of a similar quality to the visuals produced by more high quality kinematic animation systems used in modern video games. This task is difficult because human characters must dynamically retain balance through contacts with an environment, and walking requires control of an underactuated dynamic system. Simulation also does not guarantee a character will move in a natural manner, so care has to be taken to ensure visually unusual behaviours do not occur as a result of control. Work in the field of reinforcement learning has demonstrated the possibility of generating physical character control policies that imitate human motions with a high degree of success. Many methods have focused selectively on generating controllers that produce high quality motion, while important factors such as responsiveness, user controllability, motion diversity, and runtime costs have been somewhat overlooked. The approach presented here focuses on improving performance with respect to all these factors. A data-driven kinematic character controller sequences and blends motion capture data in order to generate medium-term kinematic motion plans which fit user controlled high-level goals. This allows movement direction, heading direction, speed, and style of motion to be responsively altered in a real-time user controlled manner, while also capturing subtleties of human behaviour in the data. Reinforcement learning is then used to train a simulated character controller that is capable of imitating the motion of the kinematic character controlled by a user. This necessitates a training scheme that captures the full distribution of behaviours that a human is likely to use, and which enforces the learned behaviour to retain the stylistic characteristics of the generated motion while making it physically feasible. The design of this system is also made with runtime cost in mind, ensuring that the result is useful in the context of real world application in video games where performance budgets are strict.

Résumé

Cette thèse presente un système de contrôle responsif pour un personnage humain physiquement simulé. Le système de contrôle est conçu pour être résistant aux perturbations et à la même fois pour produire des mouvements d'une qualité similaire aux visuels produits par des systèmes d'animation cinématique de haute qualité utilisée dans les jeux vidéo modernes. Cette tâche est difficile car les personnages humains doivent maintenir dynamiquement l'équilibre grâce à des contacts avec un environnement, et la marche nécessite le contrôle d'un système dynamique qui est instable. La simulation ne garantit pas non plus qu'un personnage se déplacera d'une manière naturelle, il faut prendre soin d'éviter un comportement étrange. Les méthodes utilisant l'apprentissage par renforcement ont démontré la possibilité de générer des contrôleurs de caractères physiques qui imitent les mouvements humains avec un haut degré de succès. Mais, la réactivité, la contrôlabilité par l'utilisateur, la variété des mouvements et les coûts d'exécution ont été quelque peu négligés. L'approche présentée ici se concentre sur l'amélioration de tous ces facteurs. Un contrôleur de personnage pour l'animation basé sur mocap séquence et mélange les données afin de générer des plans de mouvement à moyen terme qui correspondent aux objectifs contrôlés par l'utilisateur. Cela permet de modifier en réponse la direction du mouvement, la direction du regard, la vitesse et le style de mouvement d'une manière contrôlée par l'utilisateur en temps réel, tout en capturant les subtilités du comportement humain. L'apprentissage par renforcement est ensuite utilisé pour entraîner un contrôleur de personnage simulé qui est capable d'imiter l'animation du personnage contrôlé par l'utilisateur. Cela nécessite un système d'entraînement qui capture la distribution complète des comportements qu'un humain est susceptible d'utiliser, et qui impose au contrôleur de personnage simulé de conserver les caractéristiques stylistiques du mouvement généré. La conception de ce système est faite avec le coût d'exécution à l'esprit, garantissant que le résultat est utile dans le contexte d'une application réelle, comme les jeux vidéo car ils ont des budgets de performance serrés.

Acknowledgements

This project would not have been possible without the support, opportunities, and advice provided by Professor James Richard Forbes. I am very grateful for his supervision and guidance throughout the course of my Masters studies and research. I have learned a great deal over the past few years in no small part due to his presence. This project would also have not been possible without the support of Ubisoft La Forge who provided funding, resources, data, and most of all, the amazing expertise of everyone there. Special thanks to Simon Clavet and Daniel Holden as this research has been a result of our combined efforts. Working together with them on this project over the past two years has been an unforgettable experience. Thanks also goes out to our paper's anonymous reviewers for their helpful feedback on our SIGGRAPH submission. It was a great honour to have a paper published at SIGGRAPH Asia 2019. I am also thankful for and acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and Mitacs through the Mitacs Accelerate program. Lastly, I would like to thank my family for supporting my efforts all these years.

Preface

The contributions of this thesis that are original to the author's knowledge are as follows,

- Chapter 3
  - An improved and simplified formulation of inertialization blending that uses lerp and slerp. An associated proof of relevant position, orientation, velocity, and angular velocity behaviour is given.
- Chapter 4
  - A method of optimizing geometry in order to fit the surface of a skinned mesh and emulate the mass properties of an isotropic deformable character using multiple rigid-bodies.
  - A method of anisotropically reducing compressive constraint drift in a maximal coordinate simulation using internal forces. This is done to model pressure like forces in the body, as well as to allow for more realistic compliant joints.
- Chapter 5
  - A method of using an artificial user in order to train policies using reinforcement learning that are robust to the conditions generated by worst case human users.
  - The method and details of combining motion matching with reinforcement learning such that medium-term planning is achieved in kinematic space using a database of trajectories. This enables training of a simulated character control policy which can robustly track kinematically planned motion, achieving high-level control objectives.

All text and illustrations in this thesis are produced by Kevin Bergamin. Plots as well as numerical and experimental results are produced by Kevin Bergamin and Simon Clavet. Some content and results presented here are material from the author's publication in ACM Transactions on Graphics November 2019 titled "DReCon: data-driven responsive control of physics-based characters" [1]. This Thesis serves as a more complete presentation of the content of that work, including various additions and improvements.

Table of Contents

Chapter 1

Introduction

The field of computer graphics can have its origin traced to computer programs that aimed to create interesting interactive visuals of engineered physical systems. It has been a goal of graphics researchers over time to generate visuals that are indistinguishable from reality, and one of the primary ways this has been pursued is by accurately reproducing the dynamics of physical processes that give rise to visual phenomena. There has been great success in this regard, with physically inspired light transport methods like path-tracing allowing the generation of photo-realistic images. Likewise, realistic animation of physical objects and substances has advanced through modeling of physical processes like fluid dynamics, deformable-body mechanics, and rigid-body dynamics.

While these progresses have allowed for phenomenal improvements in visuals, a large portion of computer animation remains largely driven by artistic decisions. This is particularly true in the case of the animation of characters. Artists generally have full kinematic control over character movements and interactions with the environment and as such, they perform a role similar to classical 2D animators. This can be desirable as it allows for animators to significantly stylize the animations they create, but can in other cases be problematic when the goal is to create physically accurate character movements. One possible remedy is for animation to be recorded by using motion capture technology, which utilizes spatial recordings of actors in the real world to create physically plausible character motions [2]. However, motion capture is not necessarily physically accurate if the target character and environment significantly differ from the motion-capture actor and the recording environment. Ultimately, to achieve physically accurate character motion an accurate physical simulation of a character and its environment are necessary.

Using simulation to animate characters and accurately model movement and interaction has been a topic of great interest and has given rise the study of physically-based animation. There is an underlying hope that methods can be found to produce character simulations indistinguishable from recordings of real humans and animals. Progress has been made towards this goal, but many

issues remain unsolved. Research progress in the field has been slow and difficult in no small part because of the necessity of solving complex nonlinear control problems, in particular the control of under-actuated multibody systems. Interestingly, this also means there is a large degree of overlap with the problems faced by those researching the control of legged robotic systems.

A key target application driving computer graphics has been video games. Characters must adapt continuously to their environment, user input, and unforeseen events. Current games in general animate characters using a set of animations that are chosen at runtime using heuristics. Character interaction with the environment and movement are faked using a proxy rigid-body with a simple shape such as a vertically aligned capsule. All interaction with the environment occurs through the proxy. Character movement is faked by applying forces to the proxy, and the visual character animation is purely kinematic with minimal or nonexistent interactions with the environment. Proxy based methods do not allow for realistic coupling between characters and the simulated environment, which can lead to implausible visuals. Proxy based methods do not generate novel behaviours since all animation is picked from a provided set. Proxy geometry fails to model accurate changes in center of mass position, inertia, and responses to perturbation, and proxy collision geometry is all around a poor fit for the character.

The motivation for this research has been to find a method that brings physically simulated characters closer to a state where their motion is indistinguishable from real human motion. The desire to find such a method is largely driven by the fact it could significantly improve the quality of many systems that must interact with complex physical environments, like characters in video games or legged robots. In particular, this research has focused on the goal of discovering a method that will be of practical use in current generation video games. Based on this goal, three main objectives were determined to be important aspects for the success of this research:

1. Create a method of controlling simulated characters that does not degrade stylistic qualities of character motion when compared to current state-of-the-art kinematically driven animation systems.

2. Ensure that the method retains a high degree of user directability and can respond to user control demands responsively.

3. Guarantee that the runtime performance cost of using the method is small enough to keep it practical in the context of a realistic video game performance budget.

The importance of these objectives is clarified by analyzing the limitations of prior work, and investigating why previous methods have not yet seen significant industrial adoption. These objectives are designed to ensure the results of this research meet demands that are realistically required of any system meant to replace those currently in use.

1.1 Related Work

The field of 3D kinematic character animation has a long history. For the sake of completeness, an overview of simple to complex character animation techniques will be given. Lasseter [3] gives a good summary of how traditional 2D animation techniques can be adapted to 3D animation. The classic method of an artist creating individual keys *pose-to-pose* and then filling in the gaps with in-betweens, or in the case of 3D animation spline interpolated poses, still sees significant use today for character animation [4].

In the context of video games, simply creating animations as standalone entities is insufficient. Interactivity necessitates that characters be able to *transition* between animations, for example a character that is walking may suddenly be required to jump or perform a different action. The most straight forward way is to simply end playback of the animation associated with the current action and begin playback of an animation associated with the next, however this introduces abrupt popping. A basic solution is to interpolate between multiple animations over a set blending time [2]. Blend trees can be used to combine motions, multiple simultaneously playing animations are interpolated to form mixtures of multiple similar behaviours [5]. The issue with such a method is that naive interpolations can easily lead to low quality results [2]. Large state machines must be constructed with many heuristics to prevent bad blends, limiting the generality and maintainability of techniques that rely on simple rule based blending operations [6].

A large amount of research has been devoted to the invention of new techniques to improve the quality of kinematic animations which meet user input requirements. A subset of these are procedural animation techniques. A commonly used method is *inverse kinematics* (IK), where a control rig is created and allows a character skeleton to be manipulated by solving for the kinematic configuration that fits constraints imposed by the rig [2]. This is used to modify animations in order to position end effectors, such as making a character hold an object or position its feet to prevent sliding and ground penetration [7]. Warping is another procedural method, and can be used to generate new animations from existing ones [7, 8].

Methods exist to generate animations which fit specified contexts. These methods usually require large amounts of data in order to produce high quality results, so they tend to rely on *motion capture* data rather than artist generated content. Motion capture is attractive because it provides large quantities of high quality physically plausible animation [2]. Kernel based methods are a simple way to allow new animations to be generated from data. For example Rose et al. [9] utilize radial basis functions (RBFs) to create a parameterization which allows the generation of animations which mix existing motion classes in the data. This is somewhat like an "intelligent" blend tree. However, this method has limitations. The source animations must not differ to significantly and require careful alignment. The alignment process can also be automated using classification

heuristics as shown by Kovar et al. [10]. More robust automation can be achieved by using Gaussian processes (GPs) instead of RBFs [11]. Similarly, animations can be parameterized into useful low dimensional embeddings using GPs, for example generation can be guided using end effector positions similar to IK, or through a low-dimensional vector [12, 13]. These simple kernel based methods nonetheless remain prone to issues either with noise and variance, or with memory and computational costs growing unfavourable as the number of examples increases. This limits their scalability, as mentioned by Holden et al. [14].

Newer deep learning based generative models allow for significant improvements in scalability when compared to kernel based methods. Holden et al. [14] demonstrate that a deep neural network can learn to encode a high-level control parameterization such as character trajectory or IK targets utilizing training data. This can be used to generate animations that fit control requirements. This method however can produce undesirable "averaged" behaviour if the high-level parameterization has ambiguity, for example if multiple motions in data are encoded to the same parameters. Solving the issue of ambiguity can be done by carefully augmenting data with information about motion phase, and parameterizing the generative network using the phase [15]. This has limitations for behaviours where phase is ill defined due to non-periodicity or multi-modal behaviours. Starke et al. [16] show this issue can be resolved if similar high-level features are disambiguated using a second network to parameterize the generative network into unambiguous weighted modes. The generality of deep learning has also allowed for unstructured contextual information about the surrounding environment to be added to the input, which enables generation of animations that have realistic interactions with environmental objects [17] or interactions with other characters in a goal oriented way [18].

Deep learning promises a high degree of generality, but it is not a panacea. There is a significant tendency for generative models to only generate interpolations of the data used for training. A common trick is to perform *data augmentation* to generate more varied animations for training from a base data set [15, 16, 17, 18]. Even with data augmentation, artifacts like foot sliding or improper contacts are common and must be corrected through post-processes like IK on the output of the network [15, 16, 17]. In addition, computational cost tends to be much higher than traditional animation strategies as large and complicated neural network designs are required to produce high quality results. Training neural networks also takes a significant amount of time and data, making iteration and fine tuning slow processes. The strength in deep learning based methods seems to be their potential to compress large amounts of animation data and learn complex encodings that facilitate choreography of existing animations.

There are also data-driven approaches to animation generation that do not use deep learning. *Motion graphs* [19] are directed graphs modeling connections between segments of compatible seamlessly transitioning animation in a database. Standard or modified graph traversal algo-

rithms can then be used to generate animations that follow along certain paths and complete different tasks at different times by visiting edges of the directed graph in particular orders. Motion graphs provide a high-level and explainable data driven method for animation synthesis, but can be unwieldy to use because they strictly enforce that animations and transitions must be part of a pre-computed graph. High degrees of connectivity can also make the graph searches computationally expensive.

Motion matching is a technique where database search is used for animation synthesis [6, 20, 21, 22, 23, 24, 25]. Animations and user input requirements are parameterized using manually designed encodings. The currently playing animation and user input form are encoded into a query and distances to encodings for all frames in a database are used to determine which data best matches the query. Best matches are blended with the currently playing animation, allowing the generation to be directed through query manipulation. Motion matching is one of the few generative methods that has seen widespread adoption in videogame production. It allows for generation of highly realistic and directable animations, similar to many of the other techniques mentioned, but with other attractive characteristics. Search over the database can be optimized to have a low and fixed performance cost, and the lack of long training times makes iteration by manipulation of the data set or design variables instant. The simplicity of the design also makes it straightforward to debug. The primary limitation is that performance and memory requirements scale with the size of the dataset.

1.1.1 Physically-Based Character Animation

Dynamic processes tend to govern all time varying phenomena. Scientists and engineers model the movements of objects using mathematics, capturing the rich behaviours of physical systems. In animation, characters and objects are usually made to move in a manner consistent with our physical expectations because large discrepancies are easily noticed by the viewer [2]. Even in stylistic workflows animators are encouraged to follow well known guidelines so that their animations look at least somewhat physically plausible [3].

Simulation-based methods of character animation are used to generate *physically-based* animation, that is, animation generated by respecting physical principles. These methods have sometimes directly been adapted from robotic control research, replacing the robot with a simulated one, and at other times been developed expressly for computer graphics. An overview of the most relevant strategies is covered here, the primary focus being on work related to character controllers for simulated bipeds or quadrupeds.

A large body of work is focused on the manual design of physical character controllers. Legged locomotion control is extremely complex, but simple heuristics have been discovered that perform quite well. Raibert and Hodgins [26] developed a straightforward technique of an-

imating legged characters using methods from robotics. Leg stiffness is modulated to control hopping oscillations and an inverted pendulum model is used to determine desirable foot placements and provide feedback. A simple state machine is utilized to coordinate timing of actions based on gait phase. *SIMBICON* [27] is a method that utilizes a pose based finite state machine. States determine angles for joints to target using PD control. Center of mass position and velocity is used as feedback to alter the targets in such a way that they help to achieve a balancing effect. A large portion of the control can be achieved by the innate stability of a feed forward component. Coros et al. [28] show this can be further improved by performing gravity compensation and predicting favorable foot placements using an inverted pendulum model. The Jacobian transpose is used to transform desired control forces in global space to joint space. Finite state machines have also been used to control more complicated characters with actuation provided by individual muscles [29]. Manually designed controllers such as these can often have quite robust performance but have the downside that they often do not have behaviour that will generalize to edge cases. The manual tuning process is also time consuming, and motion style is difficult to control. Some computer graphics research has focused on allowing for better stylistic control by using external forces to stabilize the character. One method is to artificially increase stance foot traction and support size to prevent toppling [30], allowing a controlled character to act similar to a fixed base robot. Another technique is to simply apply external forces to provide direct control of a character's center of mass, correcting for the potential failures of a manual designed controller [31]. The issue with external forces is that they create unnatural reactions, sometimes allowing characters to recover from unrealistic states and perturbations.

A popular alternative approach to heuristic based controllers are those based on online optimization. Each control action can be carefully selected through an optimization process, such that it attempts to maximizes some measure of performance. The high dimensionality of legged locomotion dynamics make long timescale exhaustive optimizations prohibitively slow in real-time applications. *Model Predictive Control* (MPC) is one of the methods of choice, as it still allows controller design to be underpinned by a high-level performance measure specification, but can be computationally efficient since control actions are only determined optimal over a short local time window. High quality, robust, and generalized controllers for bipedal characters can be generated, allowing recovery from complete falls, and stable dynamic tracking of locomotion by the optimization of per timestep joint torques [32, 33]. MPC has also been applied with great success for control of quadrupedal robots, such as the MIT Cheetah 3 [34]. Control of properties such as angular momentum, linear momentum, and center of pressure can also be achieved through online optimization by specifying tracking costs, allowing for robust balancing behaviours with high visual fidelity to be achieved [35]. In contrast to MPC, long horizon online optimization can also be performed if simplified models are used [36]. While online optimization is incredibly

powerful since it allows for realtime tuning of cost functions and very general controllers, it still tends to be impractically costly on current hardware.

Offline optimization discovers control actions or controller parameters which can be optimal over long time horizons through pre-computation, enabling much lower computational costs at runtime. One method is to optimize open-loop control actions that enable tracking of a trajectory [37, 38], however this does not allow for robust reactions to perturbations due to the lack of feedback. Various strategies can be used to improve controller robustness. Feedback corrections to perturbations can be combined with optimized open-loop trajectories [39, 40]. Linear feedback strategies can also be optimized offline which are associated with each time step of a trajectory [41]. Offline optimization has also been applied to learn feedback and behaviour transitions simultaneously [42, 43], allowing for choreographing of behaviours and higher robustness. Offline optimization is attractive because pre-computation complexity has no impact on runtime complexity. This makes it practical in performance constrained applications but time consuming to work with and difficult to tweak.

Recently, there has been a large surge in the popularity of control methods based on *Reinforcement Learning* (RL). RL can roughly be characterized as a method of optimal control, where controllers learn which actions are optimal (in the sense of maximizing sums of rewards) through sampling and exploration. The strength in RL lies in the ability to learn complex control behaviours across a variety of tasks using simple rewards, but currently this requires a significant number of samples which makes it difficult to apply in real world robotics applications [44]. Nonetheless, impressive robotic controllers with state-of-the-art performance on complex tasks such as dexterous manipulation with a human-like hand [45] or legged locomotion [46, 47] have been enabled using RL. Although the target platform of these controllers are real robots, a *sim-to-real* approach is common where controllers are first trained to optimize performance with respect to a simulation. The strengths of RL can be exploited most effectively in simulations since a large number of samples can be gathered for learning purposes in a short amount of time.

Control of physically-based characters fits well within the framework of RL, so there has been a strong focus on this research area in recent years. Physically-based characters can be trained to learn locomotion abilities from scratch using simple rewards for forward motion and stability [48, 49, 50]. However learning behaviours from scratch commonly leads to strange behavioural artifacts since global optimization is in general impossible, or otherwise requires careful initialization to guide behaviour. Another approach is to use RL to schedule transitions between multiple more traditionally optimized or manually designed controllers with associated motion fragments [51, 52], enabling higher level control and robustness by planning motion through the different stability regions. This scheduling approach requires complex architecture and reward design to handle the hierarchy of behaviours and ensure appropriate fragment ordering during transitions.

Discrete choices must also be made for planning, which can be problematic. *Motion imitation* allows for more natural looking behaviours to be learned, by rewarding controllers for imitating a target behaviour while preventing loss of balance. Such an approach is similar to the trajectory optimization approach, however RL enables simultaneous learning of non-linear feedback control, and additional rewards beyond those required for motion imitation make integration of tasks other than trajectory tracking straightforward. For example, basic locomotion control and navigation can be achieved while following a small set of reference motions [53], or basketball dribbling can be maintained through hand contacts while controlling overall movement [54]. DeepMimic [55] achieves impressive motion imitation results on a variety of complex tasks such as jump-kicks and back-flips, while also allowing specification of simple goals like a kick position. Multiple of these high-level controllers can be combined to create a selectable repertoire of robust control skills. The methods presented in DeepMimic can be expanded to imitate a large database of motion-capture skills by carefully designing the state and training procedure [56].

This thesis presents work which builds on the concept of RL for motion imitation, but maximizes responsiveness to user control changes for a variety of locomotion tasks through use of a kinematic character controller combined with a specialized training procedure. This method achieves high robustness, good motion quality, and low computational cost. The results of this work were published in ACM Transactions on Graphics in November 2019 and are currently considered state-of-the-art [1].

Chapter 2

Preliminaries

The following chapter gives an overview of methods used in character animation, kinematics, physical simulation, function approximation, and reinforcement learning which are relevant to this work. This chapter serves to both familiarize the reader with the necessary topics, and present the notation adopted throughout this work. For this reason it is suggested even those with experience in these various topics at least skim this section to familiarize themselves with the notation used in later chapters.

2.1 Quaternions & Rotations

Quaternions are an extension of complex numbers to a higher number of complex dimensions. Quaternions are defined using *three* imaginary values with the relationship,

$$\tilde{i}^2 = \tilde{j}^2 = \tilde{k}^2 = \tilde{i}\tilde{j}\tilde{k} = -1.$$

This has the consequence that multiplication of the different imaginary values is non-commutative, that is,

$$\begin{aligned}\tilde{i}\tilde{j} &= -\tilde{j}\tilde{i} = \tilde{k} \\ \tilde{j}\tilde{k} &= -\tilde{k}\tilde{j} = \tilde{i} \\ \tilde{k}\tilde{i} &= -\tilde{i}\tilde{k} = \tilde{j}.\end{aligned}$$

A quaternion $\mathbf{q}^a$ is the hypercomplex number,

$$\mathbf{q}^a = \eta^a + q_x^a\tilde{i} + q_y^a\tilde{j} + q_z^a\tilde{k}.$$

The superscript position will commonly be used for variable identifiers or indexes in this work in a manner similar to the typical usage of subscripts, and should not be confused for exponentiation. Exponents will always be contrasted from identifiers by context or by using brackets, for example, $(\mathbf{q}^a)^x$ or for quaternion valued functions $\mathbf{q}^a(t)^x$. This rule will be followed throughout this work, and is necessary due to the large number of variables needed.

The imaginary components associated with $\mathbf{q}^a$ can be written as a column matrix. This allows the quaternion to be written in the following more convenient form,

$$\boldsymbol{\epsilon}^a = \begin{bmatrix} q_x^a \\ q_y^a \\ q_z^a \end{bmatrix}$$
$$\mathbf{q}^a = \eta^a + \begin{bmatrix} \tilde{i} & \tilde{j} & \tilde{k} \end{bmatrix} \boldsymbol{\epsilon}^a.$$

Multiplying two quaternions results in another quaternion and is in general non-commutative [57],

$$\begin{aligned} \mathbf{q}^a\mathbf{q}^b &= \left(\eta^a + \begin{bmatrix} \tilde{i} & \tilde{j} & \tilde{k} \end{bmatrix} \boldsymbol{\epsilon}^a\right)\left(\eta^b + \begin{bmatrix} \tilde{i} & \tilde{j} & \tilde{k} \end{bmatrix} \boldsymbol{\epsilon}^b\right) \\ &= \eta^a\eta^b - (\boldsymbol{\epsilon}^a)^\mathsf{T}\boldsymbol{\epsilon}^b + \begin{bmatrix} \tilde{i} & \tilde{j} & \tilde{k} \end{bmatrix} \left(\eta^a\boldsymbol{\epsilon}^b + \eta^b\boldsymbol{\epsilon}^a + \boldsymbol{\epsilon}^a \times \boldsymbol{\epsilon}^b\right). \end{aligned}$$

Notably, the identity quaternion is simply the real number 1, that is $\mathbf{q}_I = 1$,

$$\mathbf{q}_I\mathbf{q}^a = \mathbf{q}^a\mathbf{q}_I = \mathbf{q}^a.$$

The multiplicative inverse of a quaternion is its conjugate quaternion,

$$(\mathbf{q}^a)^{-1} = \eta^a - \begin{bmatrix} \tilde{i} & \tilde{j} & \tilde{k} \end{bmatrix} \boldsymbol{\epsilon}^a$$
$$(\mathbf{q}^a)^{-1}\mathbf{q}^a = \mathbf{q}^a(\mathbf{q}^a)^{-1} = 1.$$

The norm of a quaternion is defined as follows,

$$\|\mathbf{q}^a\| = \sqrt{\eta^a\eta^a + \boldsymbol{\epsilon}^a \cdot \boldsymbol{\epsilon}^a}.$$

The quaternions with unit norm are of primary interest as they can be used to represent rotations. Each unit quaternion $\mathbf{q}^a$ and its sign flipped form $-\mathbf{q}^a$ are related to a rotation $\mathbf{R}^a$ (both unit

quaternions map to the same rotation),

$$\begin{aligned}\mathbf{R}^a &= \begin{bmatrix} 1-2(q_y^a)^2-2(q_z^a)^2 & 2q_x^aq_y^a-2q_z^a\eta^a & 2q_x^aq_z^a+2q_y^a\eta^a \\ 2q_x^aq_y^a+2q_z^a\eta^a & 1-2(q_x^a)^2-2(q_z^a)^2 & 2q_y^aq_z^a-2q_x^a\eta^a \\ 2q_x^aq_z^a-2q_y^a\eta^a & 2q_y^aq_z^a+2q_x^a\eta^a & 1-2(q_x^a)^2-2(q_y^a)^2 \end{bmatrix} \\ &= (1-(\boldsymbol{\epsilon}^a)^{\mathsf{T}}\boldsymbol{\epsilon}^a)\mathbf{1}+2\boldsymbol{\epsilon}^a(\boldsymbol{\epsilon}^a)^{\mathsf{T}}+2\eta^a(\boldsymbol{\epsilon}^a)^{\times},\end{aligned}$$

where the cross operator $(\cdot)^{\times} : \mathbb{R}^3 \to \mathfrak{so}(3), \quad \mathfrak{so}(3) = \left\{\mathbf{A} \in \mathbb{R}^{3\times 3} \mid \mathbf{A} = -\mathbf{A}^{\mathsf{T}}\right\}$ generates a skew symmetric matrix from a column matrix such that matrix multiplication is equivalent to taking a cross product, $\mathbf{a}^{\times}\mathbf{b} = \mathbf{a} \times \mathbf{b}$. The symbol $\mathbf{1}$ represent the identity matrix.

A unit quaternion can be used to rotate an arbitrary vector, $\mathbf{x} \in \mathbb{R}^3$ directly using quaternion algebra by converting $\mathbf{x}$ into a "pure quaternion" first,

$$\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\mathbf{R}^a\mathbf{x} = (\mathbf{q}^a)^{-1}\left(\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\mathbf{x}\right)\mathbf{q}^a.$$

This is a well known result. The $\circ$ operator is defined in this work to rotate vectors by the rotation associated with a unit quaternion,

$$\mathbf{R}^a\mathbf{x} = \mathbf{q}^a \circ \mathbf{x}.$$

Composing unit quaternions is equivalent to composing rotations,

$$\mathbf{R}^a\mathbf{R}^b\mathbf{x} = \mathbf{q}^a\mathbf{q}^b \circ \mathbf{x}.$$

Because all rotations can be represented as a rotation with an angle of $\theta \in \mathbb{R}$ around some unit axis $\hat{\mathbf{u}} \in \mathbb{S}^2$ all unit quaternions can be put in the following form,

$$\mathbf{q}(\hat{\mathbf{u}}, \theta) = \cos\frac{\theta}{2} + \begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\sin\frac{\theta}{2}.$$

Just as complex numbers are related to rotations in the plane, quaternions are related to 3D rotations. Euler's formula generalizes to quaternions, giving an equation for the exponential function of pure quaternions,

$$\exp\left(\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\frac{\theta}{2}\right) = \cos\frac{\theta}{2} + \begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\sin\frac{\theta}{2}.$$

The inverse operation defines the natural logarithm of a unit quaternion

$$\log\left(\cos\frac{\theta}{2}+\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\sin\frac{\theta}{2}\right)=\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\frac{\theta}{2}.$$

Exponentiation of a unit quaternion can also be defined by the generalization of de Moivre's formula,

$$\mathbf{q}(\hat{\mathbf{u}},\theta)^x=\exp\left(\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\hat{\mathbf{u}}\frac{\theta}{2}x\right).$$

The derivative of exponentiation for a unit quaternion can also be defined [58],

$$\frac{\mathrm{d}}{\mathrm{d}x}(\mathbf{q}^a)^x=(\mathbf{q}^a)^x\log(\mathbf{q}^a).$$

Up to now the promotion of a column matrix $\mathbf{x}\in\mathbb{R}^3$ to a pure quaternion has been shown using a pre-multiplication by a matrix in order to keep things clear. A tilde accent will be considered to represent the pure quaternion representation of a 3 element column matrix in order to simplify equations,

$$\tilde{\mathbf{x}}=\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\mathbf{x}.$$

The exponential function of a pure quaternion can then be written in a compact form,

$$e^{\tilde{\mathbf{x}}}=\cos\|\mathbf{x}\|+\frac{\tilde{\mathbf{x}}}{\|\mathbf{x}\|}\sin\|\mathbf{x}\|.$$

Division by zero may seem concerning, but using the small-angle approximation the exponential function for a pure quaternion with zero length can still be considered defined given the above definition, and will return identity,

$$\begin{aligned}\lim_{\|\mathbf{x}\|\to 0}e^{\tilde{\mathbf{x}}}&=\lim_{\|\mathbf{x}\|\to 0}\cos\|\mathbf{x}\|+\frac{\tilde{\mathbf{x}}}{\|\mathbf{x}\|}\sin\|\mathbf{x}\|.\\&=\lim_{\|\mathbf{x}\|\to 0}1+\frac{\tilde{\mathbf{x}}}{\|\mathbf{x}\|}\|\mathbf{x}\|\\&=1+\begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\mathbf{0}\\&=1.\end{aligned}$$

Alternatively (and more rigorously) this can be shown using the fact $\tilde{\mathbf{0}} = 0$,

$$\lim_{\|\mathbf{x}\| \to 0} e^{\tilde{\mathbf{x}}} = e^{\tilde{\mathbf{0}}} = e^0 = 1.$$

In this work a slightly overloaded notation will be used where quaternions are written as column matrices to allow for both matrix and quaternion algebra without introducing too many new symbols,

$$\mathbf{q}^a = \begin{bmatrix} \boldsymbol{\epsilon}^a \\ \eta^a \end{bmatrix}.$$

This is somewhat of an abuse of notation, especially since quaternion multiplication is written as $\mathbf{q}^a\mathbf{q}^b$. Nonetheless an operator symbol for quaternion multiplication (for example $\mathbf{q}^a * \mathbf{q}^b$) will not be used, multiplication of quaternions will instead be very obvious from the fact quaternions are adjacent as they will always be represented by bold letter $\mathbf{q}$. Quaternion operations like exponentiation, inversion, logarithm, etc. will also be implied by use of a quaternion argument.

Where legal matrix operations apply they will be allowed, for example multiplication with a matrix $\mathbf{A}\mathbf{q}^a$, transposing $(\mathbf{q}^a)^\mathsf{T}$, or dot products $\mathbf{q}^a \cdot \mathbf{q}^b$ will treat the quaternion like a normal column matrix. A unit quaternion in this notation is considered an element of the 3 sphere, $\mathbf{q} \in \mathbb{S}^3 = \{\mathbf{x} \in \mathbb{R}^4 : \|\mathbf{x}\| = 1\}$.

As unit quaternions represent rotations, it is possible to obtain angular velocities from time varying unit quaternion functions such as $\mathbf{q}^a(t) \in \mathbb{S}^3$. The angular velocity function resolved in the parent frame, $\boldsymbol{\omega}^a(t) \in \mathbb{R}^3$, can be obtained from the quaternion time derivative through pre-multiplication with a specially constructed matrix [57],

$$\begin{aligned} \mathbf{q}^a(t) &= \begin{bmatrix} \boldsymbol{\epsilon}^a(t) \\ \eta^a(t) \end{bmatrix} \\ \boldsymbol{\omega}^a(t) &= 2 \begin{bmatrix} (\eta^a(t)\mathbf{1} + \boldsymbol{\epsilon}^a(t)^\times) & -\boldsymbol{\epsilon}^a(t) \end{bmatrix} \dot{\mathbf{q}}^a(t). \end{aligned}$$

The quaternion time derivative can be obtained from the parent frame angular velocity if needed,

$$\dot{\mathbf{q}}^a(t) = \frac{1}{2} \begin{bmatrix} \eta^a(t)\mathbf{1} - \boldsymbol{\epsilon}^a(t)^\times \\ -\boldsymbol{\epsilon}^a(t)^\mathsf{T} \end{bmatrix} \boldsymbol{\omega}^a(t).$$

A time varying unit quaternion with constant angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$ can be created using the exponential function,

$$\begin{aligned}
\mathbf{q}^a(t) &= e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}t} \\
&= \cos\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right) + \frac{\tilde{\boldsymbol{\omega}}}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right) \\
&= \begin{bmatrix} \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|t}{2}\right) \\ \cos\left(\frac{\|\boldsymbol{\omega}\|t}{2}\right) \end{bmatrix}.
\end{aligned}$$

Proving this is a bit involved,

$$\begin{aligned}
\dot{\mathbf{q}}^a(t) &= \frac{\mathrm{d}}{\mathrm{d}t} e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}t} \\
&= e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}t}\log(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}}) = e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}t}\frac{1}{2}\tilde{\boldsymbol{\omega}} \\
&= \frac{1}{2} e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}t}\tilde{\boldsymbol{\omega}} \\
&= \frac{1}{2}\left(\left(-\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right)\cdot\boldsymbol{\omega}\right)\right. \\
&\left. + \begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\left(\cos\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right)\boldsymbol{\omega} + \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right)\times\boldsymbol{\omega}\right)\right) \\
&= \frac{1}{2}\left(-\|\boldsymbol{\omega}\|\sin\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right) + \begin{bmatrix}\tilde{i} & \tilde{j} & \tilde{k}\end{bmatrix}\cos\left(\frac{\|\boldsymbol{\omega}\|\, t}{2}\right)\boldsymbol{\omega}\right) \\
&= \frac{1}{2}\begin{bmatrix} \boldsymbol{\omega}\cos\left(\frac{\|\boldsymbol{\omega}\|t}{2}\right) \\ -\|\boldsymbol{\omega}\|\sin\left(\frac{\|\boldsymbol{\omega}\|t}{2}\right)\end{bmatrix} \\
\theta &= \frac{\|\boldsymbol{\omega}\|\, t}{2} \\
\boldsymbol{\omega}^a(t) &= 2\begin{bmatrix}(\eta^a(t)\mathbf{1} + \boldsymbol{\epsilon}^a(t)^\times) & -\boldsymbol{\epsilon}^a(t)\end{bmatrix}\dot{\mathbf{q}}^a(t) \\
&= 2\begin{bmatrix}\cos\theta\mathbf{1} + \frac{\boldsymbol{\omega}^\times}{\|\boldsymbol{\omega}\|}\sin\theta & -\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\theta\end{bmatrix}\frac{1}{2}\begin{bmatrix}\boldsymbol{\omega}\cos\theta \\ -\|\boldsymbol{\omega}\|\sin\theta\end{bmatrix} \\
&= \begin{bmatrix}\cos\theta\mathbf{1} + \frac{\boldsymbol{\omega}^\times}{\|\boldsymbol{\omega}\|}\sin\theta & -\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\theta\end{bmatrix}\begin{bmatrix}\boldsymbol{\omega}\cos\theta \\ -\|\boldsymbol{\omega}\|\sin\theta\end{bmatrix} \\
&= \left(\cos\theta\mathbf{1} + \frac{\boldsymbol{\omega}^\times}{\|\boldsymbol{\omega}\|}\sin\theta\right)\boldsymbol{\omega}\cos\theta + \left(\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}\sin\theta\right)\|\boldsymbol{\omega}\|\sin\theta \\
&= \boldsymbol{\omega}\left(\cos^2\theta + \sin^2\theta\right) \\
&= \boldsymbol{\omega} \quad \square.
\end{aligned}$$

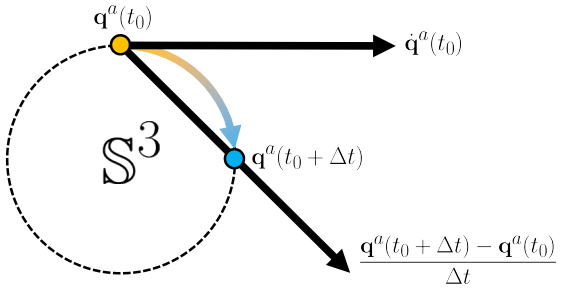


Figure 2.1: Finite difference using samples (blue and orange points) of a unit quaternion function will not produce an approximation of $\dot{\mathbf{q}}^a(t_0)$ that lies in the proper tangent space.

When performing finite difference to obtain angular velocities from unit quaternion samples of a time varying function, it is common to assume that the signal has constant angular velocity between samples. This allows for more robust calculation of angular velocities since finite difference using $\dot{\mathbf{q}}^a(t_0) \approx (\mathbf{q}^a(t_0 + \Delta t)) - \mathbf{q}^a(t_0)/\Delta t$ is likely to produce values which are not tangent to $\mathbb{S}^3$, that is not in the same space as true unit quaternion time derivatives would be, this is visualized in Figure 2.1 for a 2D cross section.

Instead a delta quaternion can be obtained using quaternion algebra, and the angular velocity can be estimated by assuming this quaternion is the result of applying a constant angular velocity over the timestep Δt. Forward difference can be performed,

$$
\begin{aligned}
\Delta\mathbf{q}^a_{t_0+\Delta t} &= \mathbf{q}^a(t_0)^{-1}\mathbf{q}^a(t_0 + \Delta t) \\
assume: & \\
\log(\Delta\mathbf{q}^a_{t_0+\Delta t}) &= \frac{1}{2}\tilde{\boldsymbol{\omega}}^a(t_0)\Delta t \\
\boldsymbol{\omega}^a(t_0) &= \frac{2}{\Delta t}\widetilde{\log}(\Delta\mathbf{q}^a_{t_0+\Delta t}),
\end{aligned}
$$

as well as backward difference,

$$
\begin{aligned}
\Delta\mathbf{q}^a_{t_0-\Delta t} &= \mathbf{q}^a(t_0)^{-1}\mathbf{q}^a(t_0 - \Delta t) \\
assume: & \\
\log(\Delta\mathbf{q}^a_{t_0-\Delta t}) &= -\frac{1}{2}\tilde{\boldsymbol{\omega}}^a(t_0)\Delta t \\
\boldsymbol{\omega}^a(t_0) &= -\frac{2}{\Delta t}\widetilde{\log}(\Delta\mathbf{q}^a_{t_0-\Delta t}),
\end{aligned}
$$

where $\widetilde{\log}(e^{\tilde{\mathbf{x}}}) = \mathbf{x}$ is defined for convenience to indicate converting the "pure quaternion" output of the unit quaternion logarithm to a normal column matrix.

2.2 Kinematic Character Animation

Animations consist of a set of poses that vary over time. Animations are generally represented by time varying bone position and orientation tracks. For most characters, a position track is only needed for the root bone, with other local bone positions being constant for all time and all animations.

The notation used in this work to describe character poses will now be explained. The column matrix $\mathbf{p}_i^a \in \mathbb{R}^3$ represents the local position of bone $i \in \mathbb{Z}^+$ relative to its parent, in a pose given identifier a. The column matrix $\mathbf{p}_{i,k}^a \in \mathbb{R}^3$ represents the local position of bone $i \in \mathbb{Z}^+$ relative to its parent at frame index $k \in \mathbb{Z}^+$ in an animation given identifier a. The quaternion $\mathbf{q}_i^a \in \mathbb{S}^3$ represents the local orientation of bone $i \in \mathbb{Z}^+$ relative to its parent, in a pose given identifier a. The quaternion $\mathbf{q}_{i,k}^a \in \mathbb{S}^3$ represents the local orientation of bone $i \in \mathbb{Z}^+$ relative to its parent at frame index $k \in \mathbb{Z}^+$, in an animation given identifier a. Letters in a superscript position should be understood to be either pose or animation identifiers, unless otherwise stated.

The pose of a character can be represented as an ordered set of its bone positions and orientations. For the purposes of this work the pose configurations of a character with N rigid bones will be represented by ordered sets $\mathcal{P}^a$, where a is the identifier for the pose,

$$\mathcal{P}^a = (\mathbf{p}_1^a, \mathbf{q}_1^a, \ldots, \mathbf{q}_N^a)\,.$$

The character root is the only bone with a defined position in the set. The value of $\mathbf{p}_1^a$ controls the rigid translation of the character. The characters defined in this work have non-root bones which only have rotational degrees of freedom, so the ordered sets representing poses will always take the above form. A particular frame $k \in \mathbb{Z}^+$ of an animation with identifier a will be represented by using a numbered subscript which denotes which frame the pose is referring to,

$$\mathcal{P}_k^a = \left(\mathbf{p}_{1,k}^a, \mathbf{q}_{1,k}^a, \ldots, \mathbf{q}_{N,k}^a\right),$$

The ordered set $\mathcal{P}_k^a$ should be understood to represent the character pose at frame k of animation with identifier a.

Animations made of discrete frames are assumed samples of a continuous signal. Interpolation is used to convert the discrete values back to continuous signals. In this work, frames per second is denoted by $\mathrm{FPS} \in \mathbb{R}^+$. To convert continuous time values to discrete time, the greatest integer function (commonly known as floor) is required,

$$\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{Z}, \quad \lfloor x \rfloor \leq x < \lfloor x \rfloor + 1.$$

Animations generally have dense enough time sampling that simple interpolation methods are appropriate. A piece-wise continuous function of time for positions can be defined using linear interpolation,

$$\text{lerp}(\mathbf{p}_a, \mathbf{p}_b, t) = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t,$$

$$\mathbf{p}_i(t) = \text{lerp}(\mathbf{p}_{i,\lfloor t \cdot \text{FPS} \rfloor}, \mathbf{p}_{i,\lfloor t \cdot \text{FPS} \rfloor + 1}, t \cdot \text{FPS} - \lfloor t \cdot \text{FPS} \rfloor).$$

For orientation spherical linear interpolation along the shortest geodesic can be used to obtain a piece-wise continuous function of time,

$$\text{slerp}(\mathbf{q}_a, \mathbf{q}_b, t) = \begin{cases} \mathbf{q}_a(\mathbf{q}_a^{-1}\mathbf{q}_b)^t, & \text{if } \mathbf{q}_a \cdot \mathbf{q}_b \geq 0 \\ \mathbf{q}_a(\mathbf{q}_a^{-1}(-\mathbf{q}_b))^t, & \text{otherwise} \end{cases}$$

$$\mathbf{q}_j(t) = \text{slerp}(\mathbf{q}_{j,\lfloor t \cdot \text{FPS} \rfloor}, \mathbf{q}_{j,\lfloor t \cdot \text{FPS} \rfloor + 1}, t \cdot \text{FPS} - \lfloor t \cdot \text{FPS} \rfloor).$$

The poses of a character associated with animation a at time t can be defined by a time-valued pose function,

$$\mathcal{P}^a(t) = \left(\mathbf{p}_1^a(t), \mathbf{q}_1^a(t), \ldots, \mathbf{q}_N^a(t)\right),$$

where the previous definitions for time interpolated position and orientation tracks are used.

Interpolation between two *poses* will be defined as follows,

$$\text{interp}(\mathcal{P}^a, \mathcal{P}^b, t) = \left(\text{lerp}(\mathbf{p}_1^a, \mathbf{p}_1^b, t), \text{slerp}(\mathbf{q}_1^a, \mathbf{q}_1^b, t), \ldots, \text{slerp}(\mathbf{q}_N^a, \mathbf{q}_N^b, t)\right).$$

The various positions and orientations of the bones are interpolated independently using the appropriate interpolation type. Note that using this definition $\mathcal{P}^a(t)$ can also be defined as follows,

$$\mathcal{P}^a(t) = \text{interp}(\mathcal{P}^a_{\lfloor t \cdot \text{FPS} \rfloor}, \mathcal{P}^a_{\lfloor t \cdot \text{FPS} \rfloor + 1}, t \cdot \text{FPS} - \lfloor t \cdot \text{FPS} \rfloor).$$

The notation $\mathcal{P}^a(\cdot) \in \mathscr{P}$ will be used to refer to the animation with identifier a as a mathematical object. This will be necessary when a function requires an animation itself as an input argument rather than a pose. The symbol $\mathscr{P}$ denotes the set of all time valued pose functions.

Pose-velocity $\mathcal{V}^a(t)$ is associated with the poses of a particular pose function $\mathcal{P}^a(t)$, and will be defined as,

$$\mathcal{V}^a(t) = (\dot{\mathbf{p}}_1^a(t), \boldsymbol{\omega}_1^a(t), \ldots, \boldsymbol{\omega}_N^a(t)) .$$

For elements of the ordered set $\mathcal{P}^a(t)$ which are quaternions, the pose-velocity $\mathcal{V}^a(t)$ contains corresponding angular velocities rather than time derivatives of quaternions.

Since the animation pose functions are usually constructed using linear interpolation they are only piece-wise continuous. In such cases the pose velocity is calculated using finite difference. A pose of the form described thus far is adequate to fully describe the state of a character, however it will also be necessary to calculate global positions and orientations of bones for a given pose. It will also be necessary to calculate global velocities and angular velocities for a given pose-function or animation.

The notation that will be adopted is as follows, for a pose $\mathcal{P}^a(t)$, the corresponding global pose will be denoted by an ordered set $\overline{\mathcal{P}}^a(t)$ which contains the global positions $\overline{\mathbf{p}}_i^a(t)$ and orientations $\overline{\mathbf{q}}_i^a(t)$ of each bone,

$$\overline{\mathcal{P}}^a(t) = (\overline{\mathbf{p}}_1^a(t), \ldots, \overline{\mathbf{p}}_N^a(t), \overline{\mathbf{q}}_1^a(t), \ldots, \overline{\mathbf{q}}_N^a(t)) .$$

Overbars are used to clearly indicated a variable represents the associated object with components given in reference to the global inertial reference frame. Similarly, the global pose-velocity will be denoted by $\overline{\mathcal{V}}^a(t)$ which contains the global velocities $\dot{\overline{\mathbf{p}}}_i^a(t)$ and angular velocities $\overline{\boldsymbol{\omega}}_i^a(t)$ of each bone,

$$\overline{\mathcal{V}}^a(t) = \left(\dot{\overline{\mathbf{p}}}_1^a(t), \ldots, \dot{\overline{\mathbf{p}}}_N^a(t), \overline{\boldsymbol{\omega}}_1^a(t), \ldots, \overline{\boldsymbol{\omega}}_N^a(t)\right) .$$

It should be noted that because the root bone has no bone as a parent, it is considered a "child" of the global frame, and thus,

$$\begin{aligned}
\overline{\mathbf{p}}_1^a(t) &= \mathbf{p}_1^a(t) \\
\overline{\mathbf{q}}_1^a(t) &= \mathbf{q}_1^a(t) \\
\dot{\overline{\mathbf{p}}}_1^a(t) &= \dot{\mathbf{p}}_1^a(t) \\
\overline{\boldsymbol{\omega}}_1^a(t) &= \boldsymbol{\omega}_1^a(t).
\end{aligned}$$

The overbar free form will be used in most cases to indicate this can be copied directly from the pose.

To calculate the global pose from a pose it is first necessary to define the bone offsets. Each non root bone has an offset $\boldsymbol{\ell}_i \in \mathbb{R}^3, i \in \{2, \ldots, N\}$ in its parent frame from the parent. In this work the bone offsets are considered constants associated with the character definition. For a non-root bone $i \in \{2, \ldots, N\}$ the parent index $k \in \{1, \ldots, N\}, k \neq i$ can also be defined. The notation $\mathfrak{p}(i)$ will be defined to return the parent bone index k of bone i, that is $k = \mathfrak{p}(i)$. To reduce clutter, additional functions that return the parent's parent, and so forth, will be given by repeating $\mathfrak{p}$ the appropriate number of times, for example $\mathfrak{pp}(i) = \mathfrak{p}(\mathfrak{p}(i))$ and $\mathfrak{ppp}(i) = \mathfrak{p}(\mathfrak{p}(\mathfrak{p}(i)))$. Additionally the set of bone indexes corresponding to the children of bone index i will be denoted $\mathfrak{c}(i) \subset \{2, \ldots, N\}$. If $j \in \mathfrak{c}(i)$, then it must be true $\mathfrak{p}(j) = i$.

As the characters in this work are assumed to be structured as a connected acyclic graph of bones (each non-root bone has only one parent), global values can be found through forward kinematics in a single well ordered pass starting from the root and using the results of previous calculations. Assuming the indexes have been arranged in an order such that $\mathfrak{p}(i) < i \ \forall \ i \in \{2, \ldots, N\}$, then the global pose elements can be computed from $\mathcal{P}^a(t)$ using the following recurrence relations,

$$
\begin{aligned}
\overline{\mathbf{p}}_1^a(t) &= \mathbf{p}_1^a(t) \\
\overline{\mathbf{q}}_1^a(t) &= \mathbf{q}_1^a(t) \\
\overline{\boldsymbol{\ell}}_i^a(t) &= \overline{\mathbf{q}}_{\mathfrak{p}(i)}^a(t) \circ \boldsymbol{\ell}_i \\
\overline{\mathbf{p}}_i^a(t) &= \overline{\mathbf{p}}_{\mathfrak{p}(i)}^a(t) + \overline{\boldsymbol{\ell}}_i^a(t) \\
\overline{\mathbf{q}}_i^a(t) &= \overline{\mathbf{q}}_{\mathfrak{p}(i)}^a(t) \mathbf{q}_i^a(t).
\end{aligned}
$$

Using the transport theorem the global pose-velocity elements can be computed from $\overline{\mathcal{P}}^a(t)$ and $\mathcal{V}^a(t)$ using another recurrence relation,

$$
\begin{aligned}
\dot{\overline{\mathbf{p}}}_1^a(t) &= \dot{\mathbf{p}}_1^a(t) \\
\overline{\boldsymbol{\omega}}_1^a(t) &= \boldsymbol{\omega}_1^a(t) \\
\dot{\overline{\mathbf{p}}}_i^a(t) &= \dot{\overline{\mathbf{p}}}_{\mathfrak{p}(i)}^a(t) + \overline{\boldsymbol{\omega}}_{\mathfrak{p}(i)}^a(t) \times \left(\overline{\mathbf{q}}_{\mathfrak{p}(i)}^a(t) \circ \boldsymbol{\ell}_i \right) \\
\overline{\boldsymbol{\omega}}_i^a(t) &= \overline{\boldsymbol{\omega}}_{\mathfrak{p}(i)}^a(t) + \overline{\mathbf{q}}_{\mathfrak{p}(i)}^a(t) \circ \boldsymbol{\omega}_i^a(t).
\end{aligned}
$$

2.2.1 Crossfade Blending

Interpolating poses is the basis of the simplest animation blending strategy, called *crossfade* blending [59]. Crossfade blending blending involves transitioning between source animation poses $\mathcal{P}^S(t_S)$ and target animation poses $\mathcal{P}^T(t_T)$ over a fixed period of time, while both animations play. The source and target animations progress through playback independently, hence the

use of associated time indexes t_S and t_T. The blend itself is parameterized by a third parameter t_B which can either be controlled by time in the case of transitions or may be an independently controlled parameter in the case of blend-tree like applications. When a transition begins, t_B starts at $t_B = 0$, and the target animation begins playback. The transition is completed when it reaches a value of $t_B = 1$, at which point the target animation assumes full control of the character and blending is no longer needed. The character pose during blending $\mathcal{P}^B$ is given by,

$$\mathcal{P}^B(t_B) = \text{interp}(\mathcal{P}^S(t_S), \mathcal{P}^T(t_T), t_B).$$

It is also straightforward to blend between multiple animations simultaneously using crossfade blending by performing bilinear or higher interpolation. This can commonly occur if a transition begins before another has finished. In these cases the source for the second blend is the result of the first. Herein lies one of the major shortcomings of crossfade blending, blending becomes increasingly expensive as more animations must be simultaneously evaluated and transitioned the more frequently transitions overlap.

2.3 Physics Simulation

It is very common for the movement of inanimate objects in computer graphics to be completely simulation driven. The behaviour of objects with high stiffness can be accurately simulated if it is modeled as a *rigid-body*.

Definition 2.1. A rigid-body $\mathcal{B}$ is a solid object that cannot undergo deformation. Rigid bodies are composed of a collection of rigidly constrained point masses such that the distance between any two points forming the body is constant across time, and the collection has a total of 6 degrees of freedom.

The following sections give an overview of the parameters and notation used in this work for features of rigid-bodies and describes the basics of multibody simulation.

2.3.1 Rigid-Body Simulation

Rigid bodies must contain at least three rigidly connected point masses, any fewer and it is not possible to assign a unique rotation which rotates the rigid-body from one orientation to another due to lacking degrees of freedom. In general a continuous set of points with variable infinitesimal mass form a volume $\mathbb{V}_\mathcal{B} \subset \mathbb{R}^3$ describing the solid geometry of the rigid-body. The total mass of the rigid-body can be calculated through integration of infinitesimal mass elements

$d_{\mathcal{B}}$ with positions $\overline{\mathbf{r}}^{d_{\mathcal{B}}} \in \mathbb{V}_{\mathcal{B}}$,

$$m_{\mathcal{B}} = \int_{\mathbb{V}_{\mathcal{B}}} \rho(\overline{\mathbf{r}}^{d_{\mathcal{B}}}) dV,$$

where $\rho(\overline{\mathbf{r}}^{d_{\mathcal{B}}})$ gives the mass density at the position $\overline{\mathbf{r}}^{d_{\mathcal{B}}}$ associated with an infinitesimal volume dV.

Given a point $\overline{\mathbf{r}}^{c} \in \mathbb{R}^3$, it is possible to calculate the mass-moment of inertia $\overline{\mathbf{I}}^{c}_{\mathcal{B}} \in \mathbb{R}^{3\times 3}$ for the rigid-body relative to the point with resepect to the global frame,

$$\overline{\mathbf{I}}^{c}_{\mathcal{B}} = \int_{\mathbb{V}_{\mathcal{B}}} \left(\left\| \overline{\mathbf{r}}^{d_{\mathcal{B}}} - \overline{\mathbf{r}}^{c} \right\|^2 \mathbf{1} - \left(\overline{\mathbf{r}}^{d_{\mathcal{B}}} - \overline{\mathbf{r}}^{c} \right) \left(\overline{\mathbf{r}}^{d_{\mathcal{B}}} - \overline{\mathbf{r}}^{c} \right)^{\mathsf{T}} \right) \rho(\overline{\mathbf{r}}^{d_{\mathcal{B}}}) dV.$$

The usual choice for this point is the center of mass $\overline{\mathbf{r}}^{c} = \overline{\mathbf{p}}_{\mathcal{B}} \in \mathbb{R}^3$,

$$\overline{\mathbf{p}}_{\mathcal{B}} = \frac{1}{m_{\mathcal{B}}} \int_{\mathbb{V}_{\mathcal{B}}} \overline{\mathbf{r}}^{d_{\mathcal{B}}} \rho(\overline{\mathbf{r}}^{d_{\mathcal{B}}}) dV.$$

Since a rigid-body is expected to rotate and translate freely, the choice of reference frame and the point c used to derive $\overline{\mathbf{I}}^{c}_{\mathcal{B}}$ are of particular importance. If a body-fixed frame which rotates with $\mathcal{B}$ is used and if c is a point which moves rigidly with the body, such as the center of mass, then $\overline{\mathbf{I}}^{c}_{\mathcal{B}}$ will be constant, and much more convenient to work with. The center of mass is the best choice as it simplifies the equations of motion that follow. Due to the nature of its construction as a sum of scaled symmetric tensors, the body-fixed frame can be chosen such that $\mathbf{I}_{\mathcal{B}}$ is diagonal with entries equal to the principal mass-moments of inertia. For any arbitrary global mass-moment of inertia matrix, eigendecomposition can be used to find an orientation in terms of a rotation matrix $\overline{\mathbf{R}}_{\mathcal{B}}$ giving the frame defined by the principle axes of $\mathcal{B}$, and also to find the principal axis aligned body frame moment of inertia matrix $\mathbf{I}_{\mathcal{B}}$ itself,

$$\overline{\mathbf{R}}_{\mathcal{B}} = \overline{\mathbf{R}}_{\mathcal{B}} \mathbf{I}_{\mathcal{B}} \overline{\mathbf{R}}^{\mathsf{T}}_{\mathcal{B}}.$$

With this form the equations of motion of a rigid-body influenced by arbitrary forces and moments are greatly simplified and given by the well known Newton-Euler equations. In an inertial frame these are written in terms of position, rotation, velocity, and angular velocity of $\mathcal{B}$, as well as external forces $\bar{\mathbf{f}}_{\mathcal{B}}$ and torques $\overline{\mathbf{m}}_{\mathcal{B}}$ acting on $\mathcal{B}$, as follows,

$$\begin{aligned} m_{\mathcal{B}} \ddot{\overline{\mathbf{p}}}_{\mathcal{B}} &= \bar{\mathbf{f}}_{\mathcal{B}} \\ \overline{\mathbf{R}}_{\mathcal{B}} \mathbf{I}_{\mathcal{B}} \overline{\mathbf{R}}^{\mathsf{T}}_{\mathcal{B}} \dot{\overline{\boldsymbol{\omega}}}_{\mathcal{B}} + \overline{\boldsymbol{\omega}}^{\times}_{\mathcal{B}} \overline{\mathbf{R}}_{\mathcal{B}} \mathbf{I}_{\mathcal{B}} \overline{\mathbf{R}}^{\mathsf{T}}_{\mathcal{B}} \overline{\boldsymbol{\omega}}_{\mathcal{B}} &= \overline{\mathbf{m}}_{\mathcal{B}}. \end{aligned}$$

Poisson's equation is needed to relate angular velocities to the rotation matrices,

$$\dot{\overline{\mathbf{R}}}_{\mathcal{B}} = \overline{\boldsymbol{\omega}}_{\mathcal{B}}^{\times}\overline{\mathbf{R}}_{\mathcal{B}}.$$

Unit quaternions are usually preferred as a rotation parameterization in graphics due to ease of re-normalization, easy extraction of rotation axis/angle, and potential to minimize floating point operations [60]. Rotation $\overline{\mathbf{R}}_{\mathcal{B}}$ can be represented by unit quaternion $\overline{\mathbf{q}}_{\mathcal{B}} \in \mathbb{S}^3$. A matrix can be associated to the quaternion which will be needed later to relate angular velocities to orientation quaternions in a manner analogous to Poisson's equation,

$$\begin{aligned}
\overline{\mathbf{q}}_{\mathcal{B}} &= \begin{bmatrix} \overline{\boldsymbol{\epsilon}}_{\mathcal{B}} \\ \overline{\eta}_{\mathcal{B}} \end{bmatrix} \\
\overline{\mathbf{Q}}_{\mathcal{B}} &= \frac{1}{2}\begin{bmatrix} \overline{\eta}_{\mathcal{B}}\mathbf{1} - \overline{\boldsymbol{\epsilon}}_{\mathcal{B}}^{\times} \\ -\overline{\boldsymbol{\epsilon}}_{\mathcal{B}}^{\mathsf{T}} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \overline{\eta}_{\mathcal{B}}\mathbf{1} + \overline{\boldsymbol{\epsilon}}_{\mathcal{B}}^{\times} \\ -\overline{\boldsymbol{\epsilon}}_{\mathcal{B}}^{\mathsf{T}} \end{bmatrix}\overline{\mathbf{R}}_{\mathcal{B}}^{\mathsf{T}}.
\end{aligned}$$

Equations of motion can be rewritten in a form which makes integration more straightforward [61]. First some relevant terms are defined to keep things neat. Some of these terms have a dependence on the state of the rigid-body indicated through parentheses in their definition,

$$\bar{\mathbf{s}}_{\mathcal{B}} = \begin{bmatrix} \overline{\mathbf{p}}_{\mathcal{B}} \\ \overline{\mathbf{q}}_{\mathcal{B}} \end{bmatrix} \tag{2.1}$$

$$\overline{\mathbf{u}}_{\mathcal{B}} = \begin{bmatrix} \dot{\overline{\mathbf{p}}}_{\mathcal{B}} \\ \overline{\boldsymbol{\omega}}_{\mathcal{B}} \end{bmatrix} \tag{2.2}$$

$$\overline{\mathbf{M}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}}) = \begin{bmatrix} m_{\mathcal{B}}\mathbf{1} & \mathbf{0} \\ \mathbf{0} & \overline{\mathbf{R}}_{\mathcal{B}}\mathbf{I}_{\mathcal{B}}\overline{\mathbf{R}}_{\mathcal{B}}^{\mathsf{T}} \end{bmatrix} \tag{2.3}$$

$$\overline{\mathbf{F}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}}, \overline{\mathbf{u}}_{\mathcal{B}}) = \begin{bmatrix} \bar{\mathbf{f}}_{\mathcal{B}} \\ \overline{\mathbf{m}}_{\mathcal{B}} - \overline{\boldsymbol{\omega}}_{\mathcal{B}}^{\times}\overline{\mathbf{R}}_{\mathcal{B}}\mathbf{I}_{\mathcal{B}}\overline{\mathbf{R}}_{\mathcal{B}}^{\mathsf{T}}\overline{\boldsymbol{\omega}}_{\mathcal{B}} \end{bmatrix} \tag{2.4}$$

$$\overline{\mathbf{G}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}}) = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \overline{\mathbf{Q}}_{\mathcal{B}} \end{bmatrix}. \tag{2.5}$$

These are used to write the equations of motion for a rigid-body concisely as,

$$\overline{\mathbf{M}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}})\dot{\overline{\mathbf{u}}}_{\mathcal{B}} = \overline{\mathbf{F}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}}, \overline{\mathbf{u}}_{\mathcal{B}}) \tag{2.6}$$

$$\dot{\bar{\mathbf{s}}}_{\mathcal{B}} = \overline{\mathbf{G}}_{\mathcal{B}}(\bar{\mathbf{s}}_{\mathcal{B}})\overline{\mathbf{u}}_{\mathcal{B}}. \tag{2.7}$$

2.3.2 Multibody Simulation

The strength of simulation driven methods is that complicated chaotic interactions can be generated based on physical principles. In the preceding section methods for accurately simulating the physical behaviour of a rigid-body in isolation were shown. A *multibody* simulation simulates multiple objects. Constraints can be used to simulate many physical elements and effects, the most relevant being: sliding and rotating joints connecting bodies, friction between bodies, non-penetration, and motors connecting bodies.

The equations of motion for individual bodies can be grouped into a single expression. Given N bodies $\mathcal{B}_1, \ldots, \mathcal{B}_N$ equations (2.1)-(2.7) can be generalized as follows,

$$
\begin{aligned}
\overline{\mathbf{s}}_{\mathcal{M}} &= \begin{bmatrix} \overline{\mathbf{s}}_{\mathcal{B}_1} \\ \vdots \\ \overline{\mathbf{s}}_{\mathcal{B}_N} \end{bmatrix} \\
\overline{\mathbf{u}}_{\mathcal{M}} &= \begin{bmatrix} \overline{\mathbf{u}}_{\mathcal{B}_1} \\ \vdots \\ \overline{\mathbf{u}}_{\mathcal{B}_N} \end{bmatrix} \\
\overline{\mathbf{M}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}}) &= \begin{bmatrix} \overline{\mathbf{M}}_{\mathcal{B}_1}(\overline{\mathbf{s}}_{\mathcal{B}_1}) & & \\ & \ddots & \\ & & \overline{\mathbf{M}}_{\mathcal{B}_N}(\overline{\mathbf{s}}_{\mathcal{B}_N}) \end{bmatrix} \\
\overline{\mathbf{F}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}}, \overline{\mathbf{u}}_{\mathcal{M}}) &= \begin{bmatrix} \overline{\mathbf{F}}_{\mathcal{B}_1}(\overline{\mathbf{s}}_{\mathcal{B}_1}, \overline{\mathbf{u}}_{\mathcal{B}_1}) \\ \vdots \\ \overline{\mathbf{F}}_{\mathcal{B}_N}(\overline{\mathbf{s}}_{\mathcal{B}_N}, \overline{\mathbf{u}}_{\mathcal{B}_N}) \end{bmatrix} \\
\overline{\mathbf{G}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}}) &= \begin{bmatrix} \overline{\mathbf{G}}_{\mathcal{B}_1}(\overline{\mathbf{s}}_{\mathcal{B}_1}) & & \\ & \ddots & \\ & & \overline{\mathbf{G}}_{\mathcal{B}_N}(\overline{\mathbf{s}}_{\mathcal{B}_N}) \end{bmatrix}.
\end{aligned}
$$

Then the equations of motion for the system of bodies are given by,

$$
\begin{aligned}
\overline{\mathbf{M}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}})\dot{\overline{\mathbf{u}}}_{\mathcal{M}} &= \overline{\mathbf{F}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}}, \overline{\mathbf{u}}_{\mathcal{M}}) \\
\dot{\overline{\mathbf{s}}}_{\mathcal{M}} &= \overline{\mathbf{G}}_{\mathcal{M}}(\overline{\mathbf{s}}_{\mathcal{M}})\overline{\mathbf{u}}_{\mathcal{M}}.
\end{aligned}
$$

Constraints between bodies can be supported through the method of Lagrange multipliers [62]. Friction and contact constraints can be implemented by solving linear complementarity problems [61]. Refer to other work for details on these methods, they are outside the scope of this work.

2.4 Function Approximation

Machine learning has been become increasingly prevalent in modern computer graphics research. Neural networks are well known for their advantages in approximating high-dimensional functions, but simpler methods are also relevant depending on the type of data and the desired application. In general the goal is to approximate some real matrix valued function with matrix valued inputs,

$$\begin{aligned} \boldsymbol{f}(\mathbf{x}) &= \mathbf{y} \\ \mathbf{x} &\in \mathbb{R}^m \\ \mathbf{y} &\in \mathbb{R}^n. \end{aligned}$$

Function approximation itself has a few important applications. Functions can sometimes be prohibitively expensive to evaluate so function approximation is applied to reduce the need for expensive computations. Another key application is to approximate functions that can be sampled but have no known closed form which can be determined and evaluated. Typically a set of N input and output data pairs are sampled from the function as *training data*, and approximating a function from such input-output data is commonly referred to as *supervised learning*,

$$\begin{aligned} X &= (\mathbf{x}_1, \ldots, \mathbf{x}_N) \\ Y &= (\mathbf{y}_1, \ldots, \mathbf{y}_N)\,. \end{aligned}$$

One of the absolute simplest methods of approximating a function is to store a *lookup table*. A lookup table stores a mapping of known function inputs to known function outputs. Approximating the function for a given input $\mathbf{x}$ then consists of searching X to find the index i of the best matching data element $\mathbf{x}_i$, then returning $\mathbf{y}_i$. A lookup table is usually used for simple datasets where a "search" for the best matching input can be replaced with a simple mapping of inputs to indexes $\mathfrak{i}(\mathbf{x})$ in a data structure. For example if $\mathbf{x}_i = i$ then,

$$\begin{aligned} \operatorname{clamp}(x, a, b) &= \max(\min(x, b), a) \\ \mathfrak{i}(\mathbf{x}) &= \operatorname{clamp}(\lfloor \mathbf{x} + 0.5 \rfloor, 1, N) \approx \arg\min_i \|\mathbf{x} - \mathbf{x}_i\| \\ \boldsymbol{f}(\mathbf{x}) &\approx \mathbf{y}_{\mathfrak{i}(\mathbf{x})}. \end{aligned}$$

In such a case it is trivial to determine the optimal output in the range covered by the data. For more complicated sets of input-output data the indexing function $\mathfrak{i}(\mathbf{x})$ will necessarily have to be more complicated, and also some sacrifices to accuracy will probably be made.

2.4.1 Nearest Neighbor Search

If input-ouput data is sparsely distributed and/or models higher dimensional data, *nearest neighbor search* can be used to find the best matching data index for a given input and error metric. Typically the squared Euclidean distance between the data and the input is a good error choice,

$$\mathfrak{i}(\mathbf{x}) = \arg\min_i \|\mathbf{x} - \mathbf{x}_i\|^2$$
$$\boldsymbol{f}(\mathbf{x}) \approx \mathbf{y}_{\mathfrak{i}(\mathbf{x})}.$$

Unlike the lookup table example, no special structure or ordering is required in the input data. The optimization can be performed by brute force search over all indexes, or accelerated through efficient nearest neighbor search optimizations like a kd-tree [63].

2.4.2 Deep Neural Networks

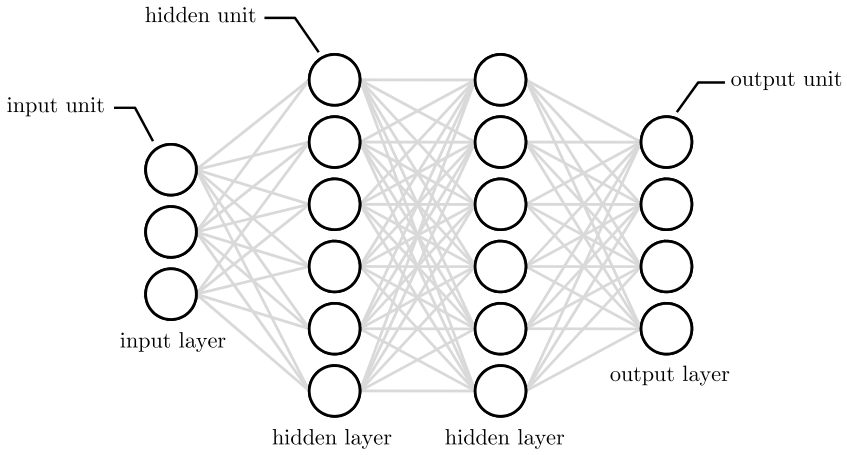


Figure 2.2: Typical nomenclature used to describe deep neural networks.

Deep Neural Networks (DNN) are computational graphs that can be used for function approximation. Nodes in the graph represent functions and directed connections define input-output relationships between nodes. The computation graphs are equivalent to a set of nested function compositions, where at least some portion of these functions are non-polynomial, and typically multiple parameters are available to modify the result of a computation in a non-linear manner. This is a very general definition but a wide variety of neural network architectures exist, most sharing this description.

While many complicated DNN architectures exist, a common and simple form is that of the *feed-forward network* as shown in Figure 2.2. The computational graph for feed-forward networks consist of multiple distinct *layers* of nodes with indexes $\ell \in \{0, \dots, h\}$. All nodes in layers where $\ell < h$ are connected to all nodes in the subsequent layer $\ell + 1$. The first layer $\ell = 0$ is the *input* layer and has no incoming connections, while the final layer $\ell = h$ is the *output* layer and has no outgoing connections. Nodes represent functions which take multi-dimensional input and have single dimensional output,

$$\alpha_{(\ell,k)}(\cdot) : \mathbb{R}^{m_{(\ell-1)}} \to \mathbb{R}.$$

where $\alpha_{(\ell,k)}(\cdot)$ refers to the function associated with the node in layer ℓ at index $k \in \{1, \dots, m_\ell\}$, and $\alpha_{(\ell,k)}$ will be used to refer to the node itself. The layer-indexed values m_ℓ refer to the *width* of the associated layers $\ell \in \{0, \dots, h\}$. The directed edges connecting nodes can be said to each have an associated *weight* value, $w_{(\ell,j,k)} \in \mathbb{R}$, referring to the weight of the edge connecting $\alpha_{(\ell-1,j)}$ to $\alpha_{(\ell,k)}$. Nodes $\alpha_{(\ell,k)}$ also have an associated *bias* value $b_{(\ell,k)} \in \mathbb{R}$.

In the context of function approximation, input nodes are used to represent the elements $x_k \in \mathbb{R},\ k \in \{1, \dots, m_0\}$ of a function input value $\mathbf{x} \in \mathbb{R}^{m_0}$,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m_0} \end{bmatrix}$$

$$\alpha_{(0,k)}(\mathbf{x}) = x_k.$$

The values in non-input nodes are usually calculated using some continuous non-linear *activation function* which is non-polynomial,

$$\psi(\cdot) : \mathbb{R} \to \mathbb{R}.$$

Multi-layer feed-forward neural networks are proven to be capable of approximating *any* continuous function to *any* degree of accuracy desired on a compact subset of $\mathbb{R}^n$ if and only if the activation functions used are non locally bounded piecewise continuous and non-polynomial [64]. This is referred to as the *Universal Approximation Theorem*. For this reason there are many reasonable activation function choices, each having properties which may in practice alter approximation performance but are theoretically capable of achieving similar degrees of accuracy.

Two common activation function choices are [65],

$$\psi_{\text{ReLU}}(x) = \max(x, 0)$$
$$\psi_{\text{tanh}}(x) = \tanh x.$$

The ouput of non-input nodes $\ell' \in \{1, \ldots, h\}$ are calculated in the same way regardless of activation function choice,

$$\mathbf{w}_{(\ell',k)} = \begin{bmatrix} w_{(\ell',1,k)} \\ \vdots \\ w_{(\ell',m_{(\ell'-1)},k)} \end{bmatrix}$$
$$\alpha_{(\ell',k)}(\mathbf{a}) = \psi(\mathbf{w}_{(\ell',k)}^{\mathsf{T}}\mathbf{a} + b_{(\ell',k)}).$$

Each layer has an associated *activation* $\mathbf{a}_\ell(\mathbf{x})$ which can be defined through a recurrence relation,

$$\mathbf{a}_0(\mathbf{x}) = \mathbf{x}$$
$$\mathbf{a}_{\ell'}(\mathbf{x}) = \begin{bmatrix} \alpha_{(\ell',1)}(\mathbf{a}_{(\ell'-1)}(\mathbf{x})) \\ \vdots \\ \alpha_{(\ell',m_{(\ell'-1)})}(\mathbf{a}_{(\ell'-1)}(\mathbf{x})) \end{bmatrix}.$$

The output vector of the network is given by the output of the final layer $\mathbf{a}_h(\mathbf{x})$. Output is dependent on the input $\mathbf{x} \in \mathbb{R}^{m_0}$ and parameters $\boldsymbol{\theta} = \left\{\mathbf{w}_{(1,1)}, \cdots, \mathbf{w}_{(h,m_h)}, b_{(1,1)}, \cdots, b_{(h,m_h)}\right\}$, so a function $\mathcal{F}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{a}_h(\mathbf{x})$ is defined. Model *hyperparameters* like the activation function type, layer count h, and layer widths $m_0, \ldots, m_h$ are considered fixed choices made by the model designer, and not included as arguments to the output function.

The feed-forward neural network can be trained to be a predictive model of some function $\boldsymbol{f}(\mathbf{x}) = \mathbf{y}$ given a set of input-output data (X, Y) by optimizing the model parameters. This process is similar to that of non-linear regression. A minimization objective function, commonly called a *loss function*, is defined to measure the prediction error of the network on data samples. Loss functions can take many forms, but for continuous function approximation *Mean Squared Error* (MSE) is a common choice. MSE is nearly identical to the sum of squared residuals used in least squares regression, with the sole modification being a normalization factor $\frac{1}{N}$ that makes evaluated losses more easily compared between different size datasets.

The MSE loss function given an input dataset X with corresponding output data Y and size N is given by,

$$L_{MSE}(\boldsymbol{\theta}, X, Y) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \mathcal{F}(\mathbf{x}_i, \boldsymbol{\theta})\|^2 .$$

Optimal neural network weights and biases are found to try and minimize the loss with respect to the training data,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, X, Y).$$

The non-linearity of the loss necessitates that this optimization be carried out using a numerical method such as gradient descent. The directed graph structure of neural networks allows gradients to be efficiently computed using *backpropagation*, a method which takes advantage of the chain rule of calculus to calculate derivatives through iterations that efficiently reuse previous computation. Refer to an introductory text [65] on machine learning for the details of the back-propagation algorithm.

Memory use can become an issue with large datasets and models, so *mini-batch stochastic gradient descent* is usually preferred over standard gradient descent. Typical loss functions (such as L_{MSE}) over a set of data can be expressed as a sum of individual sample losses,

$$L(\boldsymbol{\theta}, X, Y) = \sum_{i=1}^{N} L(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i).$$

In such cases the gradient with respect to the loss can then be expressed as sum of the gradients of individual sample losses,

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, X, Y) = \sum_{i=1}^{N} \frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i).$$

For large datasets [65] it is typical that the direction of the gradient of the loss for the entire dataset is approximated by the average direction of the gradient from randomized subsets of the data,

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, X, Y) \approx E_{D_k}\left[\sum_{i \in D_k} \frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i)\right].$$

Where $E_{D_k}[\cdot]$ refers to the expected value of the argument with respect to randomly chosen ordered sets of indices $D_k \subset \{1, \ldots, N\}$. The size of the subsets $|D_k| = N_D$ can be arbitrarily chosen, and the ordering of indices can be varied as well. If the loss is normalized by the number the data inputs, such as L_{MSE}, then the magnitudes of the gradients remain similar regardless of the subset size. Mini-batch stochastic gradient descent takes advantage of this approximation to optimize neural network parameters with increased performance. This also makes training on large data-sets which would otherwise not fit in memory possible. Given a learning rate ϵ, initial set of parameters $\boldsymbol{\theta}^0$, and *mini-batch* size N_D, iterations are carried out as follows,

$$\boldsymbol{\theta}^{k+1} \leftarrow \boldsymbol{\theta}^k + \epsilon \sum_{i \in D_k} \frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{x}_i, \mathbf{y}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}$$
$$\boldsymbol{\theta}^* \approx \lim_{k \to \infty} \boldsymbol{\theta}^k,$$

in general a new D_k is generated each iteration. The indices in D_k refer to a subset of data in X and Y and are referred to as a mini-batch. Iterations are carried out until losses reach an acceptably low level, eventually yielding the optimized set of parameters $\boldsymbol{\theta}^*$. The function used to sample data can then be approximated by the trained neural network,

$$\boldsymbol{f}(\mathbf{x}) \approx \mathcal{F}(\mathbf{x}, \boldsymbol{\theta}^*).$$

Algorithms which improve on mini-batch stochastic gradient descent are typically used over the standard formulation presented here. The *Adam* algorithm is a common choice, it automatically tunes per parameter learning rates as well as running averages of gradients to help overcome local variations in the optimization landscape that would cause convergence to unfavorable local minima [65]. This is somewhat analogous to treating the optimization problem like a simulation of the dynamics of a ball that will roll downhill and jump over small obstacles due to its inertia, while scaling the friction in various directions of travel.

2.5 Reinforcement Learning

Broadly speaking, a control system is a function that takes the state of a dynamic system as input, and generates a control action that can effect the dynamic system as output. Reinforcement learning (RL) is an area of machine learning concerned with training control systems to improve by rewarding desired behaviour and learning from interactions with the dynamic system being controlled. RL can be likened to formalizing the process of learning through trial and error [66] to maximize rewards. Trained controllers are capable of taking actions that are part of long time

horizon plans into account, with the implication that the controller may take an action which looks unfavorable in the short term if it has the ultimate effect of resulting in a more favorable outcome later on. The following section covers some preliminary topics about RL, and borrows notations, definitions, and conventions from "Reinforcement Learning: An Introduction" [66] which the reader is encouraged to read for further details.

A common framework to represent RL problems is that the *agent-environment interface*, which is shown in Figure 2.3. The *agent* represents a learner/decision maker, and everything which the

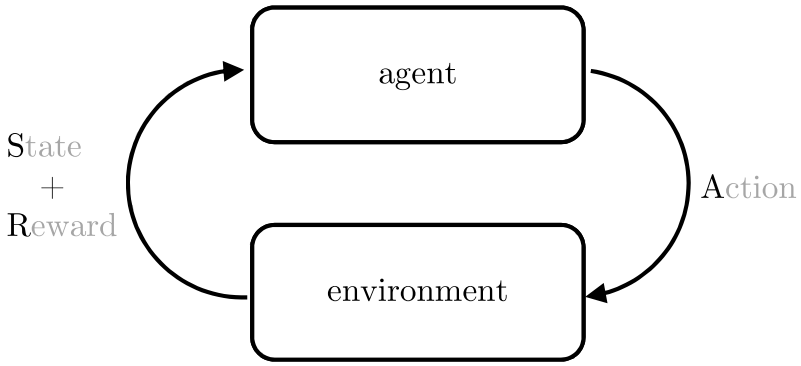


Figure 2.3: A Markov Decision Process can be represented by an agent interacting with an environment which responds to its actions.

agent interacts with at discrete time-steps $t = 0, 1, 2, \ldots$ comprises the *environment*. Each time-step, the agent receives information from the environment about its current state $S_t \in \mathcal{S}$, and takes some action $A_t \in \mathcal{A}$ based on this information. As time advances one step, the action as well as the internal dynamics of the environment result in the environment returning a new state for the agent $S_{t+1} \in \mathcal{S}$ and an associated reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. These are both forwarded to the agent, and the cycle continues indefinitely or until some termination condition is met. The resulting sequence of states, actions, and rewards form an ordered set commonly called a *trajectory*,

$$(S_0, A_0, R_1, S_1, R_2, A_1, S_2, A_2, \ldots),$$

Typically, the agent-environment interface is assumed to be a stochastic process which possesses the *Markov Property*,

Definition 2.2. If the conditional probability distribution of future states for a stochastic process depend only on the present state, the process is said to possess the *Markov Property* [66].

If the Markov property holds true the system is a *Markov Decision Process* (MDP), where the outcome resulting from the agent's decision making can be in part random and in part due to the influence of the agent. An MDP can be represented as a directed graph where some nodes represent states an agent can take in the environment, edges leaving state nodes connect to nodes representing actions which can be taken from the connected states, and edges connecting actions to new states represent the probabilistic transitions possible in the environment for the connected action. Transitions have associated probabilities and rewards. A simple two state MDP with two available action choices per state is shown in Figure 2.4 as an example. The dynamics of the MDP

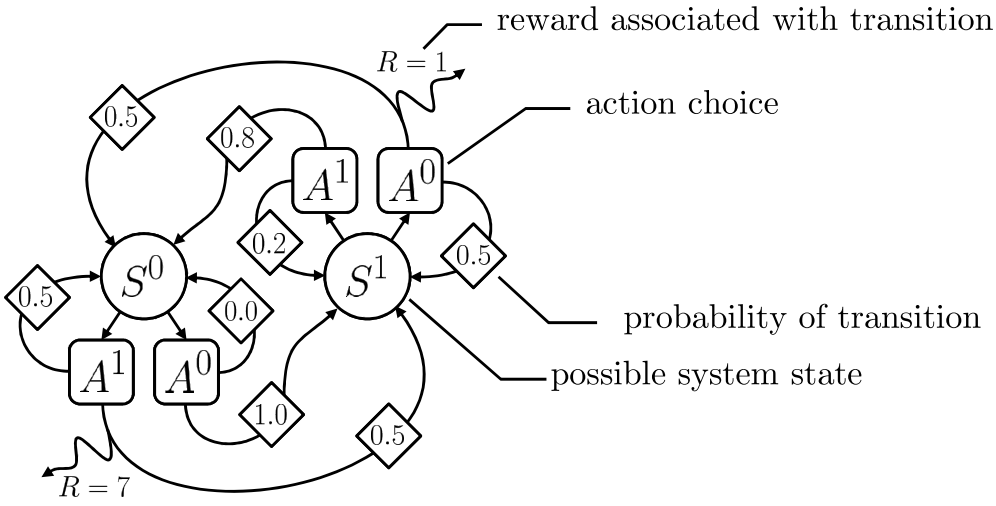


Figure 2.4: A simple markov decision process with two states, two action choices per state, transition probabilities, and transition associated rewards.

can be summarized by writing the conditional probability distributions of receiving all specifiable future states and rewards from the environment given all possible present states and actions,

$$
\begin{aligned}
S_t, R_t &\sim p(S_{t-1}, A_{t-1}) \\
p(s', r \mid s, a) &= Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\} \\
\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) &= 1, \quad \forall\, s \in \mathcal{S}, a \in \mathcal{A}(s),
\end{aligned}
$$

where the notation $Pr\{Y = y | X = x\}$ defines the probability of random variable Y taking on value y if random variable X takes on value x.

The following are also defined for convenience to assign symbols to the conditional probability of a future state, irrespective of reward, and to the distribution and probability of starting in a state,

$$\begin{aligned} p(s' \mid s, a) &= Pr\left\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right\} \\ \sum_{s' \in \mathcal{S}} p(s' \mid s, a) &= 1, \quad \forall\, s \in \mathcal{S}, a \in \mathcal{A}(s) \\ S_0 &\sim \mathcal{S}^0 \\ p(s) &= Pr\left\{S_0 = s\right\} \\ \sum_{s \in \mathcal{S}} p(s) &= 1. \end{aligned}$$

The agent has a control system that outputs action choices which is commonly referred to as a *policy*. The policy is typically a stochastic function of the state, giving rise to a conditional probability for actions,

$$\begin{aligned} A_t &\sim \pi(S_t) \\ \pi(a \mid s) &= Pr\left\{A_t = a \mid S_t = s\right\} \\ \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) &= 1, \quad \forall\, s \in \mathcal{S}. \end{aligned}$$

In most RL applications the policy is optimized such that the probability of taking actions which maximize long term sums of rewards along trajectories is increased relative to actions which offer low rewards. Long term sums of rewards are referred to as *returns*, and a random variable G_t is defined to represent the return associated with each time step,

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$$

where $T \in [0, \infty)$ is a random variable determined by the stopping condition of the problem and $\gamma \in [0, 1]$ represents a *discount factor* which can be used to adjust the weighting of present rewards compared to future rewards. The *state-value function* for any specified policy π is the expected return from starting in a specified state and following the policy,

$$v_\pi(s) = E_\pi\left[G_t \mid S_t = s\right].$$

The state-value function can also be defined using a recursive relationship [66],

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a)(r + \gamma v_\pi(s')).$$

An *action-value function* can also be defined, which is the expected return of taking a specified action in a specified state and then following an associated policy thereafter,

$$\begin{aligned} q_\pi(s, a) &= E_\pi \left[G_t \mid S_t = s,\ A_t = a \right] \\ &= E_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a \right]. \end{aligned}$$

Loosely speaking, optimal policies π^* are defined as the policies which maximize the state-value function in a given RL problem,

$$\pi^* = \arg\max_\pi v_\pi(s), \quad \forall\ s \in \mathcal{S}.$$

In practice it is virtually impossible to perform this optimization except in the most contrived examples. Numerical methods and approximation procedures must be employed to yield useful results.

2.5.1 Tabular Methods

In a limited class of problems where number of states and actions are small finite sets, RL methods can be straightforward. Understanding these builds useful intuition about more complex problems. One of the major difficulties in finding an optimal policy is that the value functions do not have a form which can be easily evaluated, making the optimization of a policy difficult. In general this is solved by using function approximation to approximate value functions. For simple problems where the number of possible states and actions are small, a lookup table can be improved iteratively by sampling trajectories, then used to converge on optimal policies.

The approximations maintained in the lookup tables are termed value function *estimates* $V_\pi(s)$, each entry in the table is associated uniquely to a state in $\mathcal{S}$. The lookup table can have all entries initialized arbitrarily, so in general the estimate will begin completely wrong. Likewise, a table can be used to represent the policy associated conditional probability distributions $\pi(a \mid s)$, with entries in the table for each state in $\mathcal{S}$ representing the probability of taking each of the actions in $\mathcal{A}(s)$. In general, $\pi(a \mid s)$ can also be initialized arbitrarily, but usually the probability of actions is set to make all actions have at least some non-zero level of probability.

Policy iteration is a method which can be used to improve the performance of the policy as well as the accuracy of the value function estimate [66]. The estimate will be improved such that

Algorithm 2.1 τ sampling procedure

$s_0 \leftsquigarrow \mathcal{S}^0$
while $t < T$ **do**
    $a_t \leftsquigarrow \pi(\cdot \mid s_t)$
    $s_{t+1}, r_{t+1} \leftsquigarrow p(s_t, a_t)$
    if $s_{t+1} \in \mathcal{S}^+$ **then**
        $T = t + 1$
    end if
    $t \leftarrow t + 1$
end while
then define
$\tau = (s_0, a_0, r_1, s_1, a_1, \ldots, r_T, s_T)$
$g_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \ \forall\, t \in \{0, \ldots, T\}$

it approximates the true value function of the current policy, and then the policy is improved such that it has a larger state-value in all states compared to the previous iteration's policy. These improvement iterations are carried out until there is convergence on an optimal policy.

In each iteration, multiple trajectories τ are sampled from the MDP. Trajectories are terminated after a fixed number of steps T or when a terminal state in $\mathcal{S}^+ \subset \mathcal{S}$ is reached. Sampling a trajectory from the MDP, $\tau \leftsquigarrow$ MDP, is defined using Algorithm 2.1. The symbol $\leftsquigarrow$ is used to represent a sampling operation and the notation will be somewhat abused to indicate sampling from more abstract objects, for example $\tau \leftsquigarrow$ MDP. It is advisable that π and p allow for *exploration* such that all visitable states have a non-zero probability of being sampled along a trajectory.

Algorithm 2.2 n-step TD

repeat
    $\tau \leftsquigarrow$ MDP
    for $t = 0, 1, \ldots, T$ **do**
        $\hat{t} \leftarrow t - n + 1$
        if $\hat{t} \geq 0$ **then**
            $g \leftarrow \sum_{i=\hat{t}+1}^{\min(\hat{t}+n, T)} \gamma^{i-\hat{t}-1} r_i, \quad r_i \in \tau$
            if $\hat{t} + n < T$ **then**
                $g \leftarrow g + \gamma^n V_\pi(s_{\hat{t}+n}), \quad s_{\hat{t}+n} \in \tau$
            end if
            $V_\pi(s_{\hat{t}}) \leftarrow V_\pi(s_{\hat{t}}) + alpha\,(g - V_\pi(s_{\hat{t}}))\,, \quad s_{\hat{t}} \in \tau$
        end if
    end for
until satisfactorily converged

The state value function estimate can be improved at each time-step sampled using an algorithm called the *n-step temporal-difference method*, which has $n \in \{1, \ldots, T\}$ and $\alpha \in (0, 1]$ as tunable parameters, see Algorithm 2.2.

The n-step TD method improves the state-value estimate by averaging sampled returns g for each associated state at all sampled steps along trajectories. When a state has been visited many times the averaged returns eventually converge on the expected value of the return for the state, which is by definition the state-value function. Sampled returns are generated for each step from the rewards experienced in the next n steps and added with the state-value estimate at the state which occurs n steps later. Using the value estimate in calculation of g is referred to as *bootstrapping* and increases sample efficiency by approximating multiple branching outcomes. The parameter n controls the influence of bootstrapping.

When the state-value estimate has converged to an acceptable threshold the policy can be modified to act greedily and prefer the estimated optimal actions a^* which lead to the highest average returns. In order to maintain some level of exploration in future trajectories, action selection can be ε*-greedy*, taking the estimated optimal action with a probability $1-\varepsilon,\ \varepsilon \in (0, 1)$, and a completely random action otherwise.

Algorithm 2.3 ε-greedy policy optimization

for each $s \in \mathcal{S}$ **do**
  $a^* \leftarrow \arg\max_a \sum_{s' \in \mathcal{S},\, r \in \mathcal{R}} p(s', r \mid s, a)\,(r + \gamma V_\pi(s'))$
  $\pi(a^* \mid s) \leftarrow 1 - \varepsilon$
  for each $\hat{a} \in \mathcal{A}(s) - \{a^*\}$ **do**
    $\pi(\hat{a} \mid s) \leftarrow \varepsilon / |\mathcal{A}(s) - \{a^*\}|$
  end for
end for

In the Algorithm 2.3, a^* is found by exhaustively searching all possible outcomes of every possible action in every state, and finding the actions which maximize the probability weighted sum of estimated returns for all outcomes. The table representing the conditional probabilities of each action are then appropriately modified to make the a^* associated with each state the most probable action in that state. During each iteration, Algorithm 2.2 then Algorithm 2.3 are ran in sequence and the policy performance improves. By performing many iterations the policy converges on something akin to a fixed point representing the locally optimal policy [66]. The value of ε can also be reduced to smaller and smaller values as iterations progress. This forces the policy to converge on optimal deterministic behaviour which focuses on *exploitation* of locally optimal actions rather than *exploration* of new strategies, improving performance further if a plateau has been reached.

This method works well enough in discrete state and action spaces where it is feasible to store everything in lookup tables and exhaustively search every possible outcome to optimize the policy. While not directly applicable to cases with continuous or infinitely large state and action spaces, policy iteration is similar to *policy gradient methods* which are designed to handle these more complicated cases.

2.5.2 Policy Gradient Methods

Function approximation in the form of lookup tables played a clear role in the policy iteration method described in the previous section. Policy gradient methods share similarities with the policy iteration method shown, but allow for more elaborate function approximation techniques to be used. Typically the state-value function estimate and policy probability distribution function are modeled using deep neural networks, which necessitates that states $\mathbf{s} \in \mathbb{R}^{n_\mathcal{S}}$ and actions $\mathbf{a} \in \mathbb{R}^{n_\mathcal{A}}$ be real valued matrices rather than elements of a small set,

$$\begin{aligned}\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) &= \mathcal{F}_\pi(\mathbf{s}, \boldsymbol{\theta}) \\ \pi_{\boldsymbol{\theta}}(\mathbf{s}) &= \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}) \\ V_{\boldsymbol{\theta}}(\mathbf{s}) &= \mathcal{F}_V(\mathbf{s}, \boldsymbol{\phi}_{\boldsymbol{\theta}}),\end{aligned}$$

this allows continuous state and action sets to be used. Note that the probability density function forming the policy in a given state is defined by a multivariate normal distribution $\mathcal{N}(\cdot, \cdot)$ with mean controlled by the output of a neural network $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\cdot) : \mathbb{R}^{n_\mathcal{S}} \to \mathbb{R}^{n_\mathcal{A}}$ with parameters $\boldsymbol{\theta}$. The covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} \in \mathbb{R}^{n_\mathcal{A} \times n_\mathcal{A}}$ is usually independent of the state and used to control the degree of exploration during sampling, similar to ε in Algorithm 2.3. The elements of $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ could also be parameters in $\boldsymbol{\theta}$ if optimization of variances is desirable. The neural network controlling the mean is used to modulate normal distributions rather than directly model a custom distribution because this greatly simplifies random sampling and alleviates the complexity of ensuring the probability density over all actions integrates to one, however this is not a prerequisite for policy gradient methods to function [66]. The state-value function for any given policy $\pi_{\boldsymbol{\theta}}$ is approximated by a different neural network $V_{\boldsymbol{\theta}}(\cdot) : \mathbb{R}^{n_\mathcal{S}} \to \mathbb{R}$, with its own set of parameters $\boldsymbol{\phi}_{\boldsymbol{\theta}}$.

In policy gradient methods the goal remains the same: find an optimal policy which maximizes the returns in all states. Policy gradient methods achieve this by finding optimal policy parameters, $\boldsymbol{\theta}^*$. This requires numerically solving the following optimization,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} v_{\pi_{\boldsymbol{\theta}}}(\mathbf{s}), \ \forall \ \mathbf{s} \in \mathbb{R}^{n_\mathcal{S}}.$$

This is usually written in terms of an expected return with respect to random trajectories from the distribution created by the policy, dynamics, and initialization,

$$\begin{aligned} r(\tau) &= g_0 \\ \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} E_{\tau\sim\text{MDP}}\left[r(\tau)\right] \\ &= \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \end{aligned}$$

Of course computing this is in general not possible, the true state-value function is never known. Also only local optima can be guaranteed from numerical methods. This is the source of the name policy *gradient*, as gradient based numerical optimization is applied to find $\boldsymbol{\theta}^*$.

As in policy iteration methods, it is first necessary to obtain a state-value function estimate through sampling of trajectories. A large number of trajectories are sampled and combined into an *experience buffer* which is used to train the various neural networks, defined here as,

$$\mathcal{T} = (\tau^1, \tau^2, \ldots, \tau^{N_\mathcal{T}}).$$

For a given set of experiences, the loss used to optimize the state-value function network is the mean squared error of all time-steps along all trajectories in the experience buffer,

$$L_{V_{\boldsymbol{\theta}}}(\boldsymbol{\phi}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{N_\mathcal{T}} \sum_{i=1}^{N_\mathcal{T}} \left(\frac{1}{T^i} \sum_{t=0}^{T^i-1} \left\| g_t^i - \mathcal{F}_V(\mathbf{s}_t^i, \boldsymbol{\phi}_{\boldsymbol{\theta}}) \right\|^2 \right).$$

Superscript i are used to indicate that states, actions, rewards, returns, and terminal time-step indexes T correspond to particular trajectories τ^i, since subscripts are already used to denote time-steps. The algorithm for learning the state-value approximation can then be defined as sampling a set of trajectories and then performing gradient descent (or some variation described in section 2.4.2) to improve the approximation.

Algorithm 2.4 gather $\mathcal{T}$, optimize $V_{\boldsymbol{\theta}}$

1: **for** $i = 1, 2, \ldots, N_\mathcal{T}$ **do**
2: $\quad \tau^i \rightsquigarrow$ MDP
3: **end for**
4: $\mathcal{T} \leftarrow (\tau^1, \tau^2, \ldots, \tau^{N_\mathcal{T}})$
5: **for** N_V **iterations do**
6: $\quad \boldsymbol{\phi}_{\boldsymbol{\theta}} \leftarrow \boldsymbol{\phi}_{\boldsymbol{\theta}} + \epsilon \frac{\partial}{\partial \boldsymbol{\phi}_{\boldsymbol{\theta}}^{opt}} L_{V_{\boldsymbol{\theta}}}(\boldsymbol{\phi}_{\boldsymbol{\theta}}^{opt}, \mathcal{T})\Big|_{\boldsymbol{\phi}_{\boldsymbol{\theta}}^{opt}=\boldsymbol{\phi}_{\boldsymbol{\theta}}}$
7: **end for**

While a gradient with respect to the value function estimate could be used directly to improve the policy, this tends to be unstable [67] so it is not the preferred method. Better methods exist, many of which make use of an estimated *advantage*. The advantage function for an arbitrary policy π is defined as,

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s).$$

Advantage gives the difference for any given state between the expected return which would have been achieved by taking the specified action then following the policy, and, the return which would have been achieved by following the policy all along. Advantage values make it straightforward to determine the improvements particular actions have over those taken by the policy in a quantifiable manner. Optimal policies will always take the most advantageous action (this was leveraged in Algorithm 2.3), and a loss can be created with this in mind.

Algorithm 2.5 Generalized Advantage Estimator

for $t = 0, 1, \ldots, (T-1)$ **do**
    $\delta_t \leftarrow r_t + \gamma V_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1}) - V_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$
end for
for $t = 0, 1, \ldots, (T-1)$ **do**
    $d_t \leftarrow \sum_{t'=t}^{T-1} (\gamma\lambda)^{t'-t} \delta_{t'}$
end for

Advantages, denoted here d_t, can be estimated at each step along a sampled trajectory except the terminal states where there are no associated actions. The common method is to use a *Generalized Advantage Estimator* (GAE) [68] where a parameter $\lambda \in [0, 1]$ controls the level of bias vs. variance in the advantage estimates by influencing the amount of bootstrapping which occurs using the state-value estimates.

Proximal Policy Optimization (PPO) [67] offers a method of utilizing advantage estimates to create a loss function which ensures more stable policy improvements. The PPO loss prevents large policy changes from occurring during a single iteration, ensuring optimized policy parameters remain within a small trusted region around the previous iteration's policy. This is necessary because the state-value estimate, advantages, etc. used for the optimization are functions of the policy parameters which are actively being optimized. It is assumed these are approximating the new policy well enough to be useful even though they were sampled using the old one. PPO requires the old policy parameters from the beginning of the iteration to be stored in $\boldsymbol{\theta}_{\text{old}}$.

A PPO like clipped loss can then be defined,

$$L_{CLIP}(\boldsymbol{\theta}, \tau) = \frac{1}{T} \sum_{t=0}^{T-1} \min \left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)} d_t, \operatorname{clamp} \left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)}, 1-\rho, 1+\rho \right) d_t \right).$$

The factor ρ is usually some small positive number, $\rho = 0.2$ being cited in the original paper [67]. This parameter is proportional to how large the trusted region around the old policy should be. The PPO loss usually includes an additional term which acts as a constraint on the Kullback–Leibler (KL) divergence between the old and new policy. KL divergence measures the difference between probability distributions, and can give an indication of when the policy has changed significantly. The constraint can be replaced with an early stopping criteria triggered by the KL divergence surpassing a threshold $\varepsilon_{\text{KL}} \approx 0.015$ [69]. This ends the iteration and initiates the next, preventing the KL divergence from growing too large. A PPO like reinforcement learning algorithm combining everything so far can be defined as follows,

Algorithm 2.6 PPO

Initialize elements of $\boldsymbol{\theta}, \boldsymbol{\phi}_{\boldsymbol{\theta}}$ as desired
for N_{epoch} **iterations do**
    Gather $\mathcal{T}$, optimize $V_{\boldsymbol{\theta}}$
    GAE on trajectories in $\mathcal{T}$
    $\boldsymbol{\theta}_{\text{old}} \leftarrow \boldsymbol{\theta}$
    while $k < N_\pi$ **do**
        $k \leftarrow k + 1$
        $\Delta_{\text{KL}} \leftarrow 0$ ▷ initialize KL divergence accumulator
        for each $\tau \in \mathcal{T}$ **do**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{\epsilon}{N_{\mathcal{T}}} \frac{\partial}{\partial \boldsymbol{\theta}^{opt}} L_{CLIP}(\boldsymbol{\theta}^{opt}, \tau)\big|_{\boldsymbol{\theta}^{opt}=\boldsymbol{\theta}}$
            for $t = 0, 1, \ldots, (T-1)$ **do** ▷ average KL divergences over all samples
                $\Delta_{\text{KL}} \leftarrow \Delta_{\text{KL}} + \frac{1}{T \cdot N_{\mathcal{T}}} KL\left[\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{s}_t), \pi_{\boldsymbol{\theta}}(\mathbf{s}_t)\right]$
            end for
        end for
        if $\Delta_{\text{KL}} > \varepsilon_{\text{KL}}$ **then** ▷ trigger early stopping if threshold exceeded
            $k \leftarrow N_\pi$
        end if
    end while
end for

Algorithm 2.6 was based on the design of the PPO implementation in OpenAI's open source "Spinning Up" project [69], but omits some of the details about synchronizing multiple parallel learning processes. It should also be noted that a better performing optimizer like Adam [65] is usually used instead of vanilla gradient descent to improve training times.

After many samples have been gathered and the system has performed many iterations, a locally optimal policy is obtained with parameters $\boldsymbol{\theta}^*$. A deterministic policy can be created by setting $\Sigma_{\boldsymbol{\theta}^*} = \mathbf{0}$, which will usually improve average performance and make actions vary smoothly rather than erratically (this is equivalent to always picking the mean of the distribution as an action). The algorithm can also be modified to decrease the variance matrix values slowly as training progresses to obtain a better converged deterministic policy.

Chapter 3

Motion Matching

Finite state machines (FSMs) have seen pervasive use throughout the history of computer graphics in controlling animations for games and interactive applications. Use of FSMs can be effective and straightforward, but the shortcomings become apparent when realistic animation systems must be designed [6]. Realistic animation requires use of many animations with subtle differences, for example walking forward and turning 10 degrees mid-step versus turning 15 degrees at the end of a step will require different animations which vary only slightly in foot placement and timing. This poses an issue for FSMs since the number of states that must be tuned and maintained by the FSM designers grows quickly when many subtly different animations are desired.

Data-driven generative approaches to animation provide excellent scalability compared to FSMs and have recently begun to have more widespread usage in industry [6, 21, 22, 23, 24, 25]. Motion matching seems to be the most popular method in industry, probably due to its potential for high performance, ease of maintainability, and proven high quality results [6, 23]. This chapter provides an in-depth overview of the design and inner workings of the motion matching system utilized by this work.

Motion Matching generates animation through frequent animation database searches and continuous animation blending. Animation databases generally consist of a large number of varied animations. Animations are selected through this search process in a sequence that best achieves high-level goals. For example, the high-level goals for locomotion can be obtaining a desired movement speed, heading direction, turning radius, and style. When goals change, a search is triggered and a new animation that better meets the updated goals is selected. If the old animation and the new animation differ significantly enough there will be jumps in the animation that are undesirable. The solution to this is to blend the animations to hide inconsistencies, but also to also provide goals that will minimize the magnitude of discontinuities. With enough data and an appropriately designed set of goals it is possible to generate realistic and easily controlled

animations by piecing together disjointed segments from the database. A properly designed system will gracefully transition from old goals to new goals, while providing a visual result which is often indistinguishable from raw motion capture.

At its core motion matching is a database search with cleverly constructed queries. It should therefore be unsurprising that the data itself is an extremely important aspect of the design of a motion matching system and can significantly influence the quality of the output. While motion matching can be referred to as a "generative" approach to animation, it is a bit more accurate to consider it as a system that can automate the scheduling and stitching of animation clips in a manner that maintains high-level consistency. A motion matching system will not extrapolate to create novel animations which are not present in the manifold of animations formed by the database, so the expressiveness of the output will be directly related to the coverage of behaviours in the database.

While the straightforward solution may seem to be capturing as much data as possible, there are usually practical limitations such as cost and time constraints which prevent this. It is all too easy to capture a lot of data of the *wrong* type unless a well made plan is put in place to understand what coverage is desired beforehand. If a system is being designed with locomotion in mind, it is important to consider a few key elements.

- How will different motion styles be differentiated from one another after capture? Will it be manually or automatically?
- What motions are the target character likely to make, and what are the most significant motion parameters? For example they could be movement speed, stride length, turn radius, facing direction, movement direction, etc.
- How similar are different motion styles, and how do transitions occur?
- What can be generated without extra takes by mirroring/transforming existing takes?

Significant thought should be put into answering these questions in order to come up with a motion capture plan and associated *dance cards*. The plan should contain a scheduled breakdown of takes required to capture all desired motion styles and transitions. The dance cards are a set of actor movement patterns designed to get an adequately dense sampling of animations by continuously varying motion parameters. Some example dance cards are shows in Figure 3.1. The primary concern during data acquisition should be to optimize use of time. There is little utility in capturing the exact same motion twice because a motion matching system cannot and will not differentiate between virtually identical animations. Symmetric behaviours can be produced by mirroring a single animation, for example a left turn can be produced from a right turn, so there is low utility in capturing both. Similarly capturing rigid transformations of behaviours in the

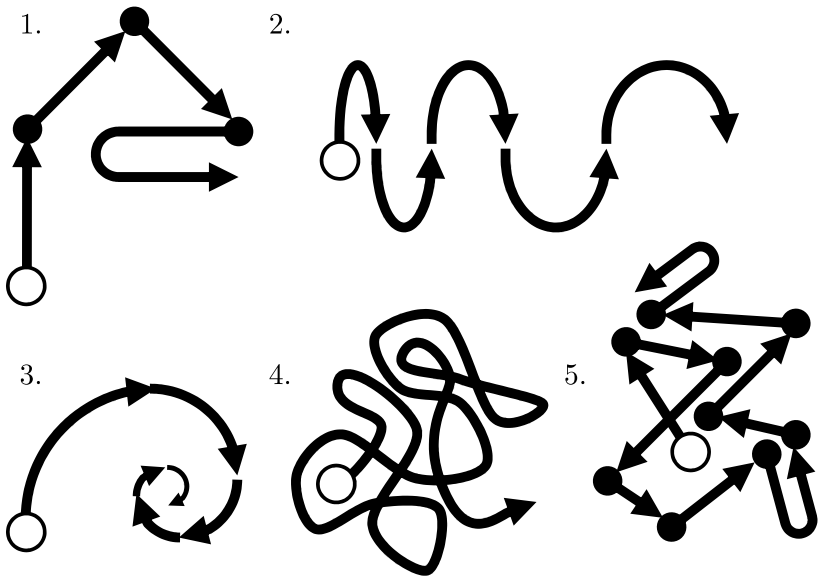


Figure 3.1: Various maneuvers that are useful for motion matching dance cards.

horizontal plane is unnecessary. For example walking north on a flat plane can be produced by rigidly transforming an animation with westward walking. A variety of numbered dance cards are shown in Figure 3.1. Referencing the number labels in the figure these have the following useful attributes for locomotion,

1. Straight line motion and right hand sharp turns at $45°$, $90°$, $135°$, and $180°$. Mirroring gives left hand turn versions of these.

2. Varying radius of curvature $180°$ turns from straight line motion.

3. Varying radius of curvature right hand turns. mirroring gives left hand turns.

4. Random unstructured meandering motion, if performed appropriately gives good coverage of different sequences of varied radius turns.

5. Random unstructured abrupt turns, if performed appropriately gives good coverage of different sequences of varied angle sharp turns.

It is a good idea to mix structured takes and unstructured takes for each style. Structured takes ensure there is good coverage of commonly encountered motions (circular arcs, 8 directional steering, plants, pivots, etc.) and ensures a set of animations in common between different styles,

while unstructured takes help fill gaps like missing turning angles, radii, and transitions between these, all of which are difficult to choreograph and describe to actors. The same dance card can be used for different styles, and for each style and each dance card pattern multiple takes at different walk and run speeds, and different facing directions relative to the tangent of the path should be performed. The number of combinations can be quite large. Capturing transitions between motion parameters is also important for behaviours like turning facing direction independent of motion direction, starting and stopping at different speeds, changes in speeds, etc. It is easiest to ask actors to perform transitional behaviours at random in unstructured takes so natural behaviour patterns are captured.

Post-processing data involves style tagging, retargeting, and mirroring. Styles can be matched to sets of time ranges, allowing the style(s) at a particular frame to be easily identified. This process is usually manual and time consuming, however simple heuristics can be developed to tag things like running, or stopped behaviours based on speed and other animation features. retargeting and mirroring of animations can usually be performed using professional software which is part of a typical motion capture production pipeline, in the case of this work Autodesk MotionBuilder [70] was used.

3.1 Motion Matching Queries

Search queries are the heart of motion matching. They provide the primary mechanism of setting and controlling high-level goals. In this section an overview of the design of the queries which were utilized for the generation of locomotion animation using motion matching are described.

The search process in motion matching functions by finding a mapping of data and queries to points in a high dimensional space. This mapping is constructed such that nearest neighbor search is appropriate to find the frame of animation in the database which is closest to meeting the high-level goals associated with the query. Ultimately, motion matching consists of searching points associated with each frame to find the one closest to the point associated with a query.

Each dimension of the space is associated with a *feature*. Feature values are matrices of real numbers which can be generated for each frame of animation. Features come in two forms, hard features and soft features. Hard features are used for binary classification, such as specifying if a particular frame is or is not a particular style, and only take on two possible values which are so far apart we consider them to have a distance of infinity in feature space. Soft features are for continuously variable properties, for example the individual velocity components of the character for the frame. Hard features are handled specially due to their unique properties, but for now they are considered numeric values without loss of generality.

Queries consist of the desired feature values the end user wants the resulting animation to take on. For example a query could specify which styles the end user wants as well as the desired velocity components of the character motion. Performing a query will find which data point is most similar to the query point, generating an animation which has the right style and the closest velocity. By stringing together queries which ask for different velocities as time progresses this example motion matching system will generate animation following a user specified path.

Because of their nature, hard feature queries result in exact feature matches, but soft features only give a match with accuracy dependent on the sampling density. A rejection threshold can be set to prevent a transition if the closest matching point is unacceptably far away from the query, allowing the current animation to continue playing as if no search occurred.

3.1.1 Query design

If queries are too strict it is likely matches will be of poor quality. A strict query is one which specifies features which are too specific or too numerous. If there are too many features the search will be in a space with so many dimensions that the chances of a point being nearby becomes low. Features can be too specific if they separate the data too sparsely, for example if the feature takes on different values walking in a straight line north versus walking in a straight line east. The goal is to make queries as simple as possible, and as general as possible.

One of the best ways to increase generality of features is to make them translation and rotation invariant. All 2D rigid transformations of an animation should produce identical features. The invariance should be 2D rather than 3D so that for example lying face down in a T-pose remains different to standing upright in a T-pose. For spatial features this can be achieved by creating a character reference frame. A character reference frame can be generated from the facing direction of the character and the direction opposite of gravity. Facing direction is estimated from the pointing direction of the root. Components of positions, orientations, velocities, and angular velocities are translation and rotation invariant features if they are taken with respect to the character reference frame relative to a character reference point.

To find the character reference frame (CRF) for a time varying pose $\mathcal{P}^a(t)$, a unit vector $\hat{\mathbf{f}}$ resolved in the root frame which faces forward while standing is first picked. The vector $\hat{\mathbf{f}}$ is usually associated with the character definition and does not change. A unit vector in global space aligned opposite gravity $-\hat{\mathbf{g}}$ is also needed. The orientation $\overline{\mathbf{R}}^a_{\text{CRF}}(t)$ of the character reference frame and the position of associated character reference point $\overline{\mathbf{p}}^a_{\text{CRF}}(t)$ are defined as follows,

$$\overline{\mathbf{R}}^a_{\text{CRF}}(t) = \left[\hat{\mathbf{g}} \times \frac{(\mathbf{q}^a_1(t) \circ \hat{\mathbf{f}}) \times \hat{\mathbf{g}}}{\left\|(\mathbf{q}^a_1(t) \circ \hat{\mathbf{f}}) \times \hat{\mathbf{g}}\right\|} \quad \frac{(\mathbf{q}^a_1(t) \circ \hat{\mathbf{f}}) \times \hat{\mathbf{g}}}{\left\|(\mathbf{q}^a_1(t) \circ \hat{\mathbf{f}}) \times \hat{\mathbf{g}}\right\|} \quad -\hat{\mathbf{g}}\right] \rightarrow \overline{\mathbf{q}}^a_{\text{CRF}}(t)$$
$$\overline{\mathbf{p}}^a_{\text{CRF}}(t) = \left(\mathbf{I} - \hat{\mathbf{g}}\hat{\mathbf{g}}^{\mathsf{T}}\right) \mathbf{p}^a_1(t).$$

Rotations are given relative to the global frame. The rotation matrix $\overline{\mathbf{R}}^{a}_{\text{CRF}}(t)$ has the associated unit quaternion representation $\overline{\mathbf{q}}^{a}_{\text{CRF}}(t)$. The character reference point $\overline{\mathbf{p}}^{a}_{\text{CRF}}(t)$ is found by projecting the root position on the ground which is assumed flat. If a pose or animation involves more complex terrain it should be projected vertically onto the terrain surface instead of a flat plane.

Queries should be composed of features which are useful for steering animations, but also features which are useful for ensuring consistency during transitions. For example, take the case of a character running in a straight line then turning. If features which control velocity are the only aspect of the query the resulting animation may perform the maneuver, but at the transition from straight line motion to turning motion pick animations where the stance foot has instantaneously changed. This will result in an instant unrealistic stance foot change. The better alternative is to provide features which ensure the feet which are currently planted remain planted, limiting the turning animations to those with the correct foot on the ground. The features and query need to be designed with the implications of various physical effects in mind,

- Inertia $\implies$ Fast velocity changes are unrealistic.
- Friction $\implies$ Sliding looks unnatural.
- Gravity $\implies$ Support contacts cannot change arbitrarily.

3.1.2 Locomotion Features

Motion Matching is well suited to the task of generating animations for locomotion. Locomotion can be controlled primarily through choice of appropriate features derived from spatial characteristics. Figure 3.2 summarizes the spatial features which were found to be most appropriate for locomotion.

Figure 3.2.a. shows *3D character root velocity*. The global 3D velocity of the character root can be used to create an inertia like constraint on the generated motion. Features can be generated from the components of this velocity resolved in the current character frame. These features are primarily used to enforce consistency between animations during transitions, preventing instantaneous velocity changes.

Figure 3.2.b. shows *foot positions and global velocities*. Maintaining consistency of foot placement and motion is important to prevent sliding during transitions and unrealistic changes in contact state. The components of foot positions and global velocities resolved in the character reference frame are used as a set of features which are rotation and translation invariant, but simultaneously can be used to enforce consistency during transitions.

Figure 3.2.c. shows *2D future character positions and headings*. Steering and velocity control can be achieved simultaneously by creating features which give the position of the character center of mass projected on the floor at points in the future relative to the current projected center

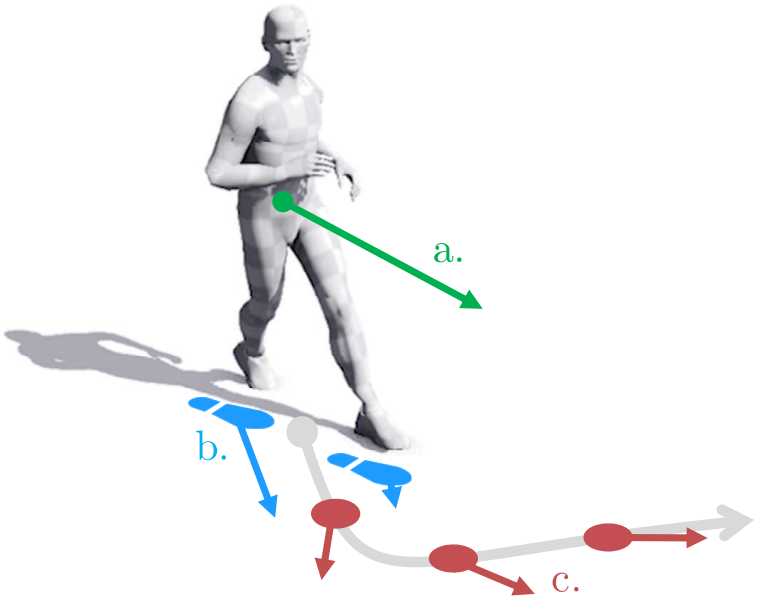


Figure 3.2: Diagram of spatial features used for applying motion matching to generation of locomotion.

of mass position. These are provided for positions 0.33 seconds, 0.66 seconds, and 1 second in the future. The relative placements of points allow both trajectory shape and velocity along a trajectory to be matched. The components of these positions resolved in the character reference frame at the current time provide rotation and translation invariant features which differentiate turns of different curvatures and transitions between velocities. It is also desirable to control character heading independently of motion to allow for behaviours like side-stepping and walking backwards. This is achieved by providing components of normalized vectors representing the heading direction of the character at 0.33 seconds, 0.66 seconds, and 1 second in the future. These vector components are also resolved in the character reference frame at the current time, providing rotation and translation invariant features which allow transitions in heading direction to be captured and controlled. All quantities are projected onto the ground, so vertical components can be omitted from the set of features.

3.2 Methods & Optimizations

Motion matching relies heavily on nearest neighbor searches. Nearest neighbor search allows for many performance optimizations to be applied in order to create a computationally efficient algorithm [63]. Soft features typically vary in a smooth manner during any particular animation,

which has the implication that an animation's frames form curves through the high dimensional feature space. The data being searched can therefore be thought of as a bundle of curves rather than a completely unstructured cloud of points, as shown in Figure 3.3a. The underlying regularity of this structure can be leveraged for optimization. A reasonable assumption is that subsequent frames of animation will be close together in feature space, which is visualized in Figure 3.3b. as an example. The first step in optimizing performance is to precompute the feature space repre-

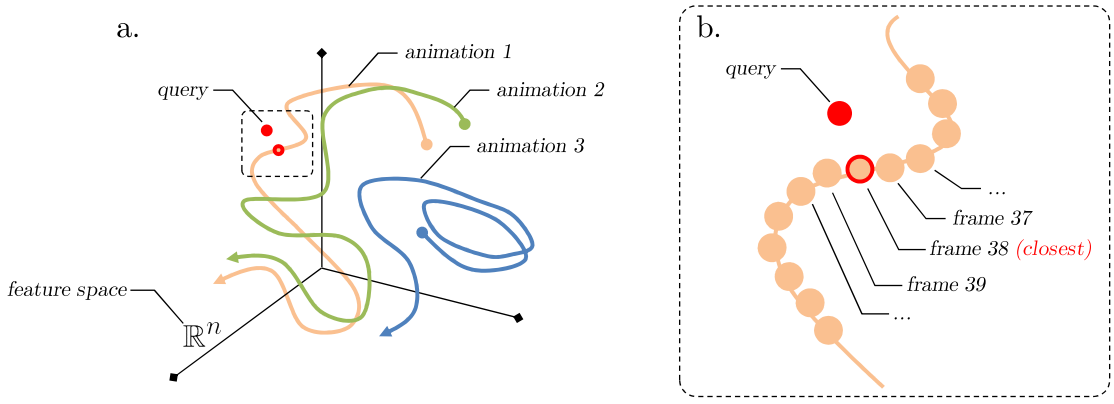


Figure 3.3: Motion matching finds the frame of animation in feature space closest to a supplied query point. Animation frames are mapped to feature space and form discretized curves.

sentation of the data. Precomputation is possible because the database contents do not typically change at runtime. Using the notation from section 2.2, a database will be described as containing a set of m animations $a_1, a_2, \ldots, a_m$ with associated frame counts $T_1, T_2, \ldots, T_m \in \mathbb{Z}^+$. Arrays of boolean values $\mathbf{b}^{a_1}, \ldots, \mathbf{b}^{a_m} \in \mathbb{B}^h$ are associated with each animation to represent whether or not each particular tag applies, with h representing the number of tags being used.

For every frame of animation in the database the features must be calculated using whatever feature mapping has been defined by the system designer. The mapping of an animation at a particular time to feature space will be given by the following function,

$$\mathfrak{P}(\cdot,\cdot,\cdot) : (\mathscr{P}, \mathbb{Z}^+, \mathbb{B}^h) \to \mathbb{R}^n,$$

That is $\mathfrak{P}$ maps a specified animation, an integer representing a frame in that animation, and a matrix of boolean values representing tags associated with the animation to a point in the feature space $\mathbb{R}^n$. Using this operator a large data matrix is constructed,

$$\begin{aligned}
\Sigma_T &= \sum_{i=1}^{m} T_i \\
\underline{\mathbf{D}}_k &= [\mathfrak{P}(\mathcal{P}^{a_k}(\cdot), 0, \mathbf{b}^{a_k})\ \mathfrak{P}(\mathcal{P}^{a_k}(\cdot), 1, \mathbf{b}^{a_k})\ \cdots\ \mathfrak{P}(\mathcal{P}^{a_k}(\cdot), T_k - 1, \mathbf{b}^{a_k})] \in \mathbb{R}^{n \times T_k} \\
\underline{\mathbf{D}} &= [\underline{\mathbf{D}}_1\ \underline{\mathbf{D}}_2\ \cdots\ \underline{\mathbf{D}}_m] \\
&= [\underline{\mathbf{d}}_1\ \underline{\mathbf{d}}_2\ \cdots\ \underline{\mathbf{d}}_{\Sigma_T}] \\
&= \begin{bmatrix} \underline{d}_{1,1} & \cdots & \underline{d}_{1,\Sigma_T} \\ \vdots & \ddots & \vdots \\ \underline{d}_{n,1} & \cdots & \underline{d}_{n,\Sigma_T} \end{bmatrix} \in \mathbb{R}^{n \times \Sigma_T}.
\end{aligned}$$

Each row in $\underline{\mathbf{D}}$ corresponds to a different feature. The mean and standard deviation for each row $k \in \{1, \ldots, n\}$ is calculated from the data,

$$\begin{aligned}
\mu_k^{\mathbf{D}} &= \frac{1}{\Sigma_T} \sum_{i=1}^{\Sigma_T} d_{k,i} \\
\sigma_k^{\mathbf{D}} &= \sqrt{\frac{1}{\Sigma_T} \sum_{i=1}^{\Sigma_T} (d_{k,i} - \mu_k^{\mathbf{D}})^2}.
\end{aligned}$$

The standard deviation is used to normalize the features, making their relative importance more evenly distributed during matching. Features are given unique weights $w_k \in \mathbb{R}^+$ to adjust their importance. Hard Features are considered to have an infinitely large weighting. This normalization and weighting produces a data matrix $\mathbf{D}$ which is defined as,

$$\begin{aligned}
\mathbf{D} &= [\mathbf{D}_1\ \mathbf{D}_2\ \cdots\ \mathbf{D}_m] \\
&= [\mathbf{d}_1\ \mathbf{d}_2\ \cdots\ \mathbf{d}_{\Sigma_T}] \\
&= \begin{bmatrix} w_1 \underline{d}_{1,1} / \sigma_1^{\mathbf{D}} & \cdots & w_1 \underline{d}_{1,\Sigma_T} / \sigma_1^{\mathbf{D}} \\ \vdots & \ddots & \vdots \\ w_n \underline{d}_{n,1} / \sigma_n^{\mathbf{D}} & \cdots & w_n \underline{d}_{n,\Sigma_T} / \sigma_n^{\mathbf{D}} \end{bmatrix} = \begin{bmatrix} d_{1,1} & \cdots & d_{1,\Sigma_T} \\ \vdots & \ddots & \vdots \\ d_{n,1} & \cdots & d_{n,\Sigma_T} \end{bmatrix} \in \mathbb{R}^{n \times \Sigma_T}.
\end{aligned}$$

The columns $\mathbf{d}_i \in \mathbb{R}^n, i \in [1, \Sigma_T]$ of $\mathbf{D}$ correspond to the feature mapping of each animation frame in the database to a point in the weighted feature space.

Given a query $\underline{\mathbf{Q}} \in \mathbb{R}^n$, a normalized and weighted query point $\mathbf{Q}$ is produced for matching against data in $\mathbf{D}$,

$$\underline{\mathbf{Q}} = \begin{bmatrix} \underline{Q}_1 \\ \vdots \\ \underline{Q}_n \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} \underline{w}_1 Q_1 / \sigma_1^{\mathbf{D}} \\ \vdots \\ \underline{w}_n Q_n / \sigma_n^{\mathbf{D}} \end{bmatrix} = \begin{bmatrix} Q_1 \\ \vdots \\ Q_n \end{bmatrix}.$$

A matching cost for the query point, C_i, can be calculated for each frame as the squared distance between the query point $\mathbf{Q}$ and the point representing the frame $\mathbf{d}_i$,

$$\mathbf{c}_i = \begin{bmatrix} c_{1,i} \\ c_{2,i} \\ \vdots \\ c_{n,i} \end{bmatrix} = \mathbf{d}_i - \mathbf{Q}$$

$$C_i = \|\mathbf{c}_i\|^2 = c_{1,i}^2 + c_{2,i}^2 + \ldots + c_{n,i}^2.$$

The best matching frame index i^* is that which has minimum cost,

$$i^* = \arg\min_i C_i.$$

Finding the minimum cost can be done in a multitude of ways. The simplest is a brute force search of every index as in Algorithm 3.7. This is unfavorable as it requires every cost to be calculated.

Algorithm 3.7 brute force matching

```
given Q
C* ← ‖c_1‖^2
i* ← 1
for i = 2, 3, ..., Σ_T do
    C_test ← ‖c_i‖^2
    if C_test < C* then
        C* ← C_test
        i* ← i
    end if
end for
return i*
```

Reducing the number of cost computations will improve performance. To find i^* every frame cost is checked against the currently known lowest value, $C_{\text{test}} < C^*$. The calculation of a cost C_i requires the summing of the squared components of $\mathbf{c}_i$. Since these values are squared the sums can only grow or remain the same size as the terms are added together,

$$c_{1,i}^2 \leq \sum_{k=1}^{2} c_{k,i}^2 \leq \sum_{k=1}^{3} c_{k,i}^2 \leq \ldots \leq \sum_{k=1}^{n-1} c_{k,i}^2 \leq \sum_{k=1}^{n} c_{k,i}^2 = C_i. \tag{3.1}$$

This can be leveraged to create an early stopping criteria. If one of the partial sums of squares is greater than the currently known lowest value, there is no sense in performing more calculations for this data point. This gives Algorithm 3.8, which will perform fewer computations than a brute force search. Even better performance can be achieved by rearranging the order of the features so that those with higher variances appear earlier in the sum, speeding up the triggering of early outs by adding larger terms earlier in the summation. Further calculations can be eliminated by group-

Algorithm 3.8 early out matching

```
given Q
C* ← ||c_0||^2
i* ← 0
for i = 1, 2, ..., Σ_T do
    C_test ← 0
    for k = 0, 1, ..., n do
        C_test ← C_test + (d_{k,i} − Q_k)^2          ▷ note: c_{k,i} = d_{k,i} − Q_k
        if C_test ≥ C* then
            skip to next i iteration
        end if
    end for
    if C_test < C* then
        C* ← C_test
        i* ← i
    end if
end for
return i*
```

ing points in the feature space into a bounding volume hierarchy. Distances to bounding volumes containing specific points give a minimum bound on distances to contained points. While any bounding volume shape can be used, axis aligned n dimensional bounding boxes (AABBs) are chosen here due to their simplicity. Bounding boxes are given identifier $B_j \subset \mathbb{R}^n$, with indexes j used to differentiate between different boxes. Each box has associated range Z_j of indexes, composed of the indexes of points contained in the box such that $\mathbf{d}_z \in B_j, \forall z \in Z_j, Z_j \subset \{1, \ldots, \Sigma_T\}$.

The minimum bounds for each feature index k, $B^{\min}_{k,j}$, and maximum bounds, $B^{\max}_{k,j}$, describe B_j,

$$B^{\min}_{k,j} = \min_{z \in Z_j} d_{k,z}$$
$$B^{\max}_{k,j} = \max_{z \in Z_j} d_{k,z}.$$

These values can be precomputed. The minimum bound on cost for points contained in the axis aligned bounding box can be calculated as follows,

$$\mathbf{c}^{B_j} = \begin{bmatrix} c_1^{B_j} \\ \vdots \\ c_n^{B_j} \end{bmatrix} = \begin{bmatrix} \text{clamp}(Q_1, B^{\min}_{1,j}, B^{\max}_{1,j}) - Q_1 \\ \vdots \\ \text{clamp}(Q_n, B^{\min}_{n,j}, B^{\max}_{n,j}) - Q_n \end{bmatrix}$$
$$C^{B_j} = \left\| c^{B_j} \right\|^2 = (c_1^{B_j})^2 + (c_2^{B_j})^2 + \ldots + (c_n^{B_j})^2.$$

The rule for partial sums in inequality (3.1) also applies for the calculation of C^{B_j}, allowing an early out on the bounding box lower cost bound calculation. The bounding volume itself provides a lower bound on the cost of points it contains. This can be described by the following inequality,

$$(c_1^{B_j})^2 \leq \sum_{k=1}^{2} (c_k^{B_j})^2 \leq \ldots \leq \sum_{k=1}^{n} (c_k^{B_j})^2 = C^{B_j} \leq C_z, \quad \forall z \in Z_j$$

Using this inequality and a set of bounding boxes, Algorithm 3.9 can be created and has the potential of skipping all calculations for points in a particular box if a calculated lower bound surpasses the currently known lowest cost. It is up to the system designer to decide how many bounding boxes to create, and which points to associate with them. If boxes are too large because they contain too many points the lower bound will almost always be useless, while if they are too small because they contain too few points they may result in a slowdown versus a brute force search since *extra* computations will be needed. Attention should be paid to the memory locality of points in the boxes. If any point can be contained in a box it is likely that Z_j will not be a contiguous range of indexes and memory access will be random, leading to cache misses which incur a significant performance penalty. Because of the fact each animation is a curve through space and frames are likely to be stored contiguously, a good option is to only allow Z_j to be a contiguous range of indexes. This makes the generation of bounding boxes simple as the range $\{1, \ldots, \Sigma_T\}$ can be broken up into a predetermined number of boxes containing an equal number of points. The points associated with contiguous ranges of indexes will also be close together if they are from the same animation, since soft feature variations are generally smooth (as visualized in Figure 3.3). The range size can be tweaked to find a value which results in optimal performance.

Algorithm 3.9 AABB matching

given Q
$C^* \leftarrow \infty$ ▷ initialize with worst case
$i^* \leftarrow -1$ ▷ dummy index
for $j = 1, 2, 3, \ldots$ **do** ▷ loop over all AABBs in order
    $C^B \leftarrow 0$
    for $k = 1, 2, \ldots, n$ **do**
        $C^B \leftarrow C^B + (\text{clamp}(Q_k, B^{\min}_{k,j}, B^{\max}_{k,j}) - Q_k)^2$
        if $C^B \geq C^*$ **then**
            skip to next j **iteration**
        end if
    end for
    for each $z \in Z_j$ **do**
        $C_{\text{test}} \leftarrow 0$
        for $k = 1, 2, \ldots, n$ **do**
            $C_{\text{test}} \leftarrow C_{\text{test}} + (d_{k,z} - Q_k)^2$ ▷ note: $c_{k,z} = d_{k,z} - Q_k$
            if $C_{\text{test}} \geq C^*$ **then**
                skip to next z **iteration**
            end if
        end for
        if $C_{\text{test}} < C^*$ **then**
            $C^* \leftarrow C_{\text{test}}$
            $i^* \leftarrow z$
        end if
    end for
end for
return i^*

Hierarchies of boxes can also be used to recursively obtain lower bound calculations. An outer box can compute lower bounds on the costs associated with the boxes it contains, and those on the boxes they contain, and so forth. This may or may not reduce the required number of calculations to perform a search depending on the data distribution. Larger boxes tend to be pretty ineffective at causing an early out to trigger, and the box size will grow with each additional hierarchy of boxes.

3.2.1 Hard Feature Searches

Up to now hard features have been treated the same as soft features in all derivations. While this is not *wrong*, it adds needless inefficiency to algorithms since the cost component associated with these features will only ever take on one of two values, 0 or ∞. If the query is asking for a particular set of hard feature values alongside the soft feature values, then for the cost to be non-

infinite it must come from a data point which has the exact same hard feature values as the query. From an implementation perspective it is also tricky to make hard feature values represented by numbers take on these infinitely distant positions, so a different representation is superior. Hard features make it so that only the data ranges associated with matching hard features need to be searched. There is no need to actually calculate the distances to hard features to find the best match. Matrices $\mathbf{D}$ and $\mathbf{Q}$ can omit the rows associated with hard features if ranges of data points are filtered from searching by using a comparison of hard features against the query. The binary valued tag matrices $\mathbf{b}_1, \ldots, \mathbf{b}_m$ for each animation or frame range can be tested for equivalence with the binary valued hard features in the query. Only those ranges of data associated with the hard features in the query need to be searched using one of the mentioned algorithms. If bounding volumes are used then they should only contain points with the same hard feature values so that they too can be filtered from the search based on the query.

3.2.2 Improved Inertialization

Motion matching systems search continuously to produce a continuous stream of animation. Each search finds a new animation segment to start playing, however this cannot be done instantaneously or else a visible pop between poses will occur. Simple strategies for animation blending like crossfade blending could be applied to smooth out this popping, however transitions occur so frequently that blends are very likely to stack up with one starting before another has ended. Stacking blends require multiple animations to be evaluated simultaneously and can have an unfavorable performance impact. A more favorable strategy is to use a blending method which is better equipped to handle frequent transitions. One method that does not require simultaneous playback of multiple animation clips is called *inertialization* [59]. An alternative method derived from the design principles of inertialization but with a simplified implementation is derived here.

The basis of inertialization is that blending can be considered as a post-process which occurs after a transition. At the time of transition t_0, the pose $\mathcal{P}^S(t_0)$ and pose-velocity $\mathcal{V}^S(t_0)$ of the source animation are stored. The goal is to then initialize a dynamic system which defines the blend pose $\mathcal{P}^B(t)$ and pose-velocity $\mathcal{V}^B(t)$ in a state which ensures,

$$\mathcal{P}^B(t_0) = \mathcal{P}^S(t_0)$$
$$\mathcal{V}^B(t_0) = \mathcal{V}^S(t_0).$$

This is meant to mimic the velocity preserving effects of inertia. The dynamic system controlling the blend is defined in such a manner that it will ultimately stabilize the pose and pose-velocity to those of the target animation at an offset time $t_T = t + \Delta t_T$ for the transition.

Stabilization can either occur after a fixed time $t_f \geq t_0$ or asymptotically if $t_f \to \infty$,

$$\mathcal{P}^B(t) = \mathcal{P}^T(t_T) \; \forall \; t \geq t_f$$
$$\mathcal{V}^B(t) = \mathcal{V}^T(t_T) \; \forall \; t \geq t_f.$$

Inertialization as described in the original presentation of the method [59] uses pose-velocity and acceleration related information with the goal of finding coeffecients of a quintic polynomial which approximate the solution of a dynamic system. However, this was found to be needlessly complex so an improved simplified method is proposed here. The choice of dynamic system is arbitrary so long as it is stable and has the desired blending properties. It is preferable if the pose and pose-velocity can be easily controlled at transition start and end. A single dimension critically damped spring damper system provides good inspiration for an asymptotically stable dynamic system,

$$\begin{aligned} \ddot{x}(t) + 2\alpha\dot{x}(t) + \alpha^2 x(t) &= 0 \\ x(t) &= (c_1 + c_2 t)e^{-\alpha t} \\ \dot{x}(t) &= (c_2(1 - \alpha t) - \alpha c_1)e^{-\alpha t}. \end{aligned}$$

Specifying initial conditions on position and velocity allows the constants to be solved for,

$$\begin{aligned} x(0) &= (c_1 + c_2 \cdot 0)e^{-\alpha \cdot 0} && \to c_1 = x(0) \\ \dot{x}(0) &= (c_2(1 - \alpha \cdot 0) - \alpha c_1)e^{-\alpha \cdot 0} && \to c_2 = \dot{x}(0) + \alpha x(0). \end{aligned}$$

However it is not so straightforward to obtain an analogous solution for a spring damper controlling 3D orientation. Instead of doing this, a method with similar dynamics which is built from interpolation functions was devised using specifically designed interpolation factor controlling functions. The following equations blend animations with a result similar to inertialization,

$$\begin{aligned} \mathbf{p}_1^B(t) &= \text{lerp}\left(\mathbf{p}_1^T(t_T), \mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t), \mathfrak{a}(t)\right) \\ \mathbf{q}_i^B(t) &= \text{slerp}\left(\mathbf{q}_i^T(t_T), e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0), \mathfrak{a}(t)\right), \quad i = 1, 2, \ldots, N \end{aligned}$$

The interpolation factors are designed to allow pose-velocity derivatives during the blend to be easily controlled,

$$\begin{aligned} \mathfrak{a}(t) &= (1 + \alpha(t - t_0))e^{-\alpha(t-t_0)} \\ \mathfrak{b}(t) &= \frac{1}{\beta}\left(1 - e^{-\beta(t-t_0)}\right). \end{aligned}$$

Exponential decay constants controlling the duration of the blend, $\alpha \in \mathbb{R}^+$, and factors controlling decay of initial velocity, $\beta \in \mathbb{R}^+$, are chosen. Values are picked by a designer to achieve different looks based on needs. It is advisable to at least select different values for blending rotation vs. blending position since they behave differently. For now these are assumed to be shared for all tracks to keep the derivations which follow simple, but per track values can be used and all derivations here will not be affected. Using the animation poses $\mathcal{P}^B(t)$ resulting from a blend, a pose-function is defined to concisely describe blending between two animations through time,

$$\text{inertialize}(\mathcal{P}^S(t_0), \mathcal{P}^T(t_T), \mathcal{V}^S(t_0), \mathcal{V}^T(t_T), t) = \mathcal{P}^B(t).$$

The inertialize function ensures that pose and pose-velocity have continuity with the source animation at the start of the transition, and asymptotically approach target animation values. This is guaranteed because of the design of $\mathfrak{a}(t)$ and $\mathfrak{b}(t)$. In particular note that,

$$\begin{aligned}
\mathfrak{a}(t_0) &= 1 && (3.2)\\
\dot{\mathfrak{a}}(t_0) &= 0 && (3.3)\\
\lim_{t\to\infty} \mathfrak{a}(t) &= 0 && (3.4)\\
\lim_{t\to\infty} \dot{\mathfrak{a}}(t) &= 0 && (3.5)\\
\mathfrak{b}(t_0) &= 0 && (3.6)\\
\dot{\mathfrak{b}}(t_0) &= 1 && (3.7)\\
\lim_{t\to\infty} \dot{\mathfrak{b}}(t) &= 0. && (3.8)
\end{aligned}$$

The start of the blend has pose continuity with the source animation. The remainder of this section is devoted to proving this method has the desired behaviour. Pose continuity with the source is shown by proving $\mathcal{P}^B(t_0) = \mathcal{P}^S(t_0)$ with $t_{T0} = t_0 + \Delta t_T$ defined to reduce clutter,

$$\begin{aligned}
\mathbf{p}_1^B(t_0) &= \text{lerp}\left(\mathbf{p}_1^T(t_{T0}), \mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t_0), \mathfrak{a}(t_0)\right)\\
&= \mathbf{p}_1^T(t_{T0}) + \left(\mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t_0) - \mathbf{p}_1^T(t_{T0})\right)\mathfrak{a}(t_0)\\
&= \mathbf{p}_1^T(t_{T0}) + \left(\mathbf{p}_1^S(t_0) + (0) - \mathbf{p}_1^T(t_{T0})\right)(1)\\
&= \mathbf{p}_1^S(t_0)\\
\mathbf{q}_i^B(t_0) &= \text{slerp}\left(\mathbf{q}_i^T(t_{T0}), e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t_0)}\mathbf{q}_i^S(t_0), \mathfrak{a}(t_0)\right), \quad i = 1, 2, \ldots, N\\
&= \mathbf{q}_i^T(t_{T0})\left((\mathbf{q}_i^T(t_{T0}))^{-1} e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t_0)}\mathbf{q}_i^S(t_0)\right)^{\mathfrak{a}(t_0)}\\
&= \mathbf{q}_i^T(t_{T0})\left((\mathbf{q}_i^T(t_{T0}))^{-1} e^{(0)}\mathbf{q}_i^S(t_0)\right)^{(1)}\\
&= \mathbf{q}_i^S(t_0).
\end{aligned}$$

Similarly it can be shown as the blend progresses that the blended pose asymptotically approaches the target animation. This is done by proving $\lim_{t\to\infty}\mathcal{P}^B(t) = \lim_{t\to\infty}\mathcal{P}^T(t)$,

$$
\begin{aligned}
\lim_{t\to\infty}\mathbf{p}_1^B(t) &= \lim_{t\to\infty}\operatorname{lerp}\left(\mathbf{p}_1^T(t_T), \mathbf{p}_1^S(t) + \dot{\mathbf{p}}_1^S(t)\mathfrak{b}(t), \mathfrak{a}(t)\right)\\
&= \lim_{t\to\infty}\mathbf{p}_1^T(t_T) + \left(\mathbf{p}_1^S(t) + \dot{\mathbf{p}}_1^S(t)\mathfrak{b}(t) - \mathbf{p}_1^T(t_T)\right)\mathfrak{a}(t)\\
&= \lim_{t\to\infty}\mathbf{p}_1^T(t_T) + (0)\\
&= \lim_{t\to\infty}\mathbf{p}_1^T(t_T)\\
\lim_{t\to\infty}\mathbf{q}_i^B(t) &= \lim_{t\to\infty}\operatorname{slerp}\left(\mathbf{q}_i^T(t_T), e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0), \mathfrak{a}(t)\right), \quad i = 1, 2, \ldots, N\\
&= \lim_{t\to\infty}\mathbf{q}_i^T(t_T)\left((\mathbf{q}_i^T(t_T))^{-1}e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0)\right)^{\mathfrak{a}(t)}\\
&= \lim_{t\to\infty}\mathbf{q}_i^T(t_T)\left((\mathbf{q}_i^T(t_T))^{-1}e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)(1)}\mathbf{q}_i^S(t_0)\right)^{(0)}\\
&= \lim_{t\to\infty}\mathbf{q}_i^T(t_T).
\end{aligned}
$$

Showing $\mathcal{V}^B(t_0) = \mathcal{V}^S(t_0)$ and $\lim_{t\to\infty}\mathcal{V}^B(t) = \lim_{t\to\infty}\mathcal{V}^T(t)$ is more involved and requires some derivatives to be calculated first,

$$
\begin{aligned}
\dot{\mathbf{p}}_1^B(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\operatorname{lerp}\left(\mathbf{p}_1^T(t_T), \mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t), \mathfrak{a}(t)\right)\\
&= \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{p}_1^T(t_T) + \left(\mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t) - \mathbf{p}_1^T(t_T)\right)\mathfrak{a}(t)\right)\\
&= \dot{\mathbf{p}}_1^T(t_T) + \left(\dot{\mathbf{p}}_1^S(t_0)\dot{\mathfrak{b}}(t) - \dot{\mathbf{p}}_1^T(t_T)\right)\mathfrak{a}(t) + \left(\mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t) - \mathbf{p}_1^T(t_T)\right)\dot{\mathfrak{a}}(t).
\end{aligned}
$$

The proof for orientation related continuity contains derivatives which quickly become unwieldy so the chain rule is applied to slerp with arbitrary parameters first,

$$
\begin{aligned}
\mathbf{f}(\mathbf{q}_a, \mathbf{q}_b, c) &= \operatorname{slerp}(\mathbf{q}_a, \mathbf{q}_b, c) = \mathbf{q}_a(\mathbf{q}_a^{-1}\mathbf{q}_b)^c\\
\dot{\mathbf{f}}(\mathbf{q}_a, \mathbf{q}_b, c) &= \frac{\partial\mathbf{f}}{\partial\mathbf{q}_a}\dot{\mathbf{q}}_a + \frac{\partial\mathbf{f}}{\partial\mathbf{q}_b}\dot{\mathbf{q}}_b + \frac{\partial\mathbf{f}}{\partial c}\dot{c}\\
&= \left((\mathbf{q}_a^{-1}\mathbf{q}_b)^c + \mathbf{q}_a c(\mathbf{q}_a^{-1}\mathbf{q}_b)^{c-1}(-\mathbf{q}_a^{-2}\mathbf{q}_b)\right)\dot{\mathbf{q}}_a\\
&\quad + \left(\mathbf{q}_a c(\mathbf{q}_a^{-1}\mathbf{q}_b)^{c-1}\mathbf{q}_a^{-1}\right)\dot{\mathbf{q}}_b\\
&\quad + \left(\mathbf{q}_a(\mathbf{q}_a^{-1}\mathbf{q}_b)^c\log(\mathbf{q}_a^{-1}\mathbf{q}_b)\right)\dot{c}.
\end{aligned}
$$

It is then helpful to evaluate what occurs if $c(t_f) = 0$ and $c(t_0) = 1$,

$$\begin{aligned}\dot{\mathbf{f}}\left(\mathbf{q}_a, \mathbf{q}_b, 0\right) &= \dot{\mathbf{q}}_a + \left(\mathbf{q}_a \log\left(\mathbf{q}_a^{-1}\mathbf{q}_b\right)\right)\dot{c} \\ \dot{\mathbf{f}}\left(\mathbf{q}_a, \mathbf{q}_b, 1\right) &= \dot{\mathbf{q}}_b + \left(\mathbf{q}_b \log\left(\mathbf{q}_a^{-1}\mathbf{q}_b\right)\right)\dot{c}.\end{aligned}$$

The pose-velocity behaviour can now be shown at t_0,

$$\begin{aligned}\dot{\mathbf{p}}_1^B(t_0) &= \dot{\mathbf{p}}_1^T(t_{T0}) + \left(\dot{\mathbf{p}}_1^S(t_0)\dot{\mathfrak{b}}(t_0) - \dot{\mathbf{p}}_1^T(t_{T0})\right)\mathfrak{a}(t_0) \\ &\quad + \left(\mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t_0) - \mathbf{p}_1^T(t_{T0})\right)\dot{\mathfrak{a}}(t_0) \\ &= \dot{\mathbf{p}}_1^T(t_{T0}) + \left(\dot{\mathbf{p}}_1^S(t_0)(1) - \dot{\mathbf{p}}_1^T(t_{T0})\right)(1) + (0) \\ &= \dot{\mathbf{p}}_1^S(t_0) \\ \dot{\mathbf{q}}_i^B(t_0) &= \dot{\mathbf{f}}\left(\mathbf{q}_i^T(t_{T0}), \left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0)\right)_{t=t_0}, \mathfrak{a}(t_0)\right), \quad i = 1, 2, \ldots, N \\ &= \dot{\mathbf{f}}\left(\mathbf{q}_i^T(t_{T0}), \left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0)\right)_{t=t_0}, (1)\right) \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0)\right)_{t=t_0} + \left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t_0)}\mathbf{q}_i^S(t_0)\log\left(\ldots\right)\right)\dot{\mathfrak{a}}(t_0) \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0)\right)_{t=t_0} + (0) \\ &= \dot{\mathfrak{b}}(t_0)e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t_0)}\log\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)}\right)\mathbf{q}_i^S(t_0) \\ &= (1)e^{(0)}\log\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)}\right)\mathbf{q}_i^S(t_0) \\ &= \log\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)}\right)\mathbf{q}_i^S(t_0) \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\left(e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)(t-t_0)}\mathbf{q}_i^S(t_0)\right)_{t=t_0} \implies \boldsymbol{\omega}_i^B(t_0) = \boldsymbol{\omega}_i^S(t_0).\end{aligned}$$

The quaternion derivatives at t_0 imply that the angular velocity terms of the blend $\boldsymbol{\omega}_i^B(t_0)$ are identical to those of quaternion functions with constant angular velocity $\boldsymbol{\omega}_i^S(t_0)$.

A similar approach follows for the case of the limiting pose-velocity,

$$\begin{aligned}
\lim_{t\to\infty} \dot{\mathbf{p}}_1^B(t) &= \lim_{t\to\infty} \dot{\mathbf{p}}_1^T(t_T) + \left(\dot{\mathbf{p}}_1^S(t_0)\dot{\mathfrak{b}}(t) - \dot{\mathbf{p}}_1^T(t_T)\right) \mathfrak{a}(t) \\
&\quad + \left(\mathbf{p}_1^S(t_0) + \dot{\mathbf{p}}_1^S(t_0)\mathfrak{b}(t) - \mathbf{p}_1^T(t_T)\right) \dot{\mathfrak{a}}(t) \\
&= \lim_{t\to\infty} \dot{\mathbf{p}}_1^T(t_T) + (0) + (0) \\
&= \lim_{t\to\infty} \dot{\mathbf{p}}_1^T(t_T) \\
\lim_{t\to\infty} \dot{\mathbf{q}}_i^B(t) &= \lim_{t\to\infty} \dot{\mathbf{f}}\left(\mathbf{q}_i^T(t_T), e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0), \mathfrak{a}(t)\right), \quad i = 1, 2, \ldots, N \\
&= \lim_{t\to\infty} \dot{\mathbf{f}}\left(\mathbf{q}_i^T(t_T), e^{\frac{1}{2}\tilde{\boldsymbol{\omega}}_i^S(t_0)\mathfrak{b}(t)}\mathbf{q}_i^S(t_0), (0)\right) \\
&= \lim_{t\to\infty} \dot{\mathbf{q}}_i^T(t_T) + \left(\mathbf{q}_i^T(t_T)\log(\ldots)\right) \dot{\mathfrak{a}}(t) \\
&= \lim_{t\to\infty} \dot{\mathbf{q}}_i^T(t_T) + (0) \\
&= \lim_{t\to\infty} \dot{\mathbf{q}}_i^T(t_T) \implies \lim_{t\to\infty} \boldsymbol{\omega}_i^B(t) = \lim_{t\to\infty} \boldsymbol{\omega}_i^T(t_T).
\end{aligned}$$

Since the quaternion derivatives of the blend pose approach those of the target animation, the angular velocities must also be equal in the limit.

Asymptotic behaviour has the implication that any blend effectively never ends. However it is possible to define $\mathfrak{a}(t)$ and $\mathfrak{b}(t)$ with a fixed blend time t_f in mind, so long as (3.2) to (3.8) are respected. This requires the additional constraints,

$$\begin{aligned}
\mathfrak{a}(t) &= 0 \;\forall\; t \geq t_f \\
\dot{\mathfrak{a}}(t) &= 0 \;\forall\; t \geq t_f \\
\dot{\mathfrak{b}}(t) &= 0 \;\forall\; t \geq t_f.
\end{aligned}$$

The main motivation behind inertialization is however to efficiently trigger transitions before previous blends have completed, so asymptotic stability is usually sufficient and does not pose any issues. When a transition is triggered, the current blend immediately ends and requires no further evaluation. The new blend source animation states are simply set using the pose and pose-velocity at the last evaluated time,

$$\begin{aligned}
\mathcal{P}^S(t_0) &\leftarrow \mathcal{P}^B(t_0) \\
\mathcal{V}^S(t_0) &\leftarrow \mathcal{V}^B(t_0).
\end{aligned}$$

Pose-velocity can be determined using finite difference with values from the previous time-step. Inertialization only requires updating $\mathcal{P}^S(t_0)$, $\mathcal{V}^S(t_0)$, and blend start time t_0 whenever a new

transition occurs. The function $\text{inertialize}(\mathcal{P}^S(t_0), \mathcal{P}^T(t_T), \mathcal{V}^S(t_0), \mathcal{V}^T(t_T), t)$ can be continuously evaluated every time step to generate animation, without regard to when or how transitions are being triggered.

3.2.3 Motion Matching Algorithm

Because motion matching functions by generating poses at discrete timesteps, output will be considered in terms of a sequence poses sequence rather than as a pose-function. The current timestep's pose and previous timestep's pose will be denoted $\mathcal{P}^{\text{MM}}$ and $\mathcal{P}^{\text{MM}-}$ respectively. The length of time between these poses is some fixed value Δt which is dependent on the target framerate of the application. At startup the previous pose and current pose are considered initialized in some way usually based on some animation in the database.

The pose-velocity at the current time is calculated using backward difference of the two pose samples and given the identifier $\mathcal{V}^{\text{MM}}$,

$$\begin{aligned}\mathcal{V}^{\text{MM}} &= \left(\frac{\mathbf{p}_1^{\text{MM}} - \mathbf{p}_1^{\text{MM}-}}{\Delta t}, -\frac{2}{\Delta t}\widetilde{\log}\left((\mathbf{q}_1^{\text{MM}})^{-1}\mathbf{q}_1^{\text{MM}-}\right), \ldots, -\frac{2}{\Delta t}\widetilde{\log}\left((\mathbf{q}_N^{\text{MM}})^{-1}\mathbf{q}_N^{\text{MM}-}\right)\right) \\ &= \left(\dot{\mathbf{p}}_1^{\text{MM}}, \boldsymbol{\omega}_1^{\text{MM}}, \ldots, \boldsymbol{\omega}_N^{\text{MM}}\right).\end{aligned}$$

When an animation is selected through the motion matching search process, it is important to consider how it should be played back to introduce the minimal amount of popping possible. First, the best matching index i^* returned by the search must be converted into an appropriate animation reference and playback time. An operation is defined which takes the matched index i^* in the data matrix $\mathbf{D}$ and returns the animation $\mathcal{P}^{a^*}(\cdot)$ and time in that animation t^* which it is uniquely associated to,

$$\mathcal{P}^{a^*}(\cdot), t^* \leftarrow \text{index\_to\_anim}(i^*).$$

However, this output usually requires processing before it can be used. If the character reference frame is used the animations in the database that are matched will not necessarily have the correct transformation due to the rotation and translation invariance of the features. The animation from the database must first be aligned through an appropriate transformation so that it begins playback in a sensible place which minimizes popping. This is done by ensuring the character reference frame of the current pose and the matched pose $\mathcal{P}^{a^*}(t^*)$ are aligned first.

The identifier $\mathcal{P}^{\text{MM}_0}$ is given to the pose when the match occurred. The best matching animation a^* is rigidly transformed in the horizontal plane to have the proper alignment at time t^* with the character reference frame of $\mathcal{P}^{\text{MM}_0}$,

$$\begin{aligned}
\Delta\mathbf{q}_{\text{align}^*} &= \overline{\mathbf{q}}^{\text{MM}_0}_{\text{CRF}}(\overline{\mathbf{q}}^{a^*}_{\text{CRF}}(t^*))^{-1} \\
\mathcal{V}^{\text{MM}_0:a^*}(t) &= \left(\Delta\mathbf{q}_{\text{align}^*} \circ \dot{\mathbf{p}}^{a^*}_1(t), \Delta\mathbf{q}_{\text{align}^*}\boldsymbol{\omega}^{a^*}_1(t), \boldsymbol{\omega}^{a^*}_2(t), \ldots, \boldsymbol{\omega}^{a^*}_N(t)\right) \\
\mathcal{P}^{\text{MM}_0:a^*}(t) &= \left(\overline{\mathbf{p}}^{\text{MM}_0}_{\text{CRF}} + \Delta\mathbf{q}_{\text{align}^*} \circ \left(\mathbf{p}^{a^*}_1(t) - \overline{\mathbf{p}}^{a^*}_{\text{CRF}}(t^*)\right), \Delta\mathbf{q}_{\text{align}^*}\mathbf{q}^{a^*}_1(t), \mathbf{q}^{a^*}_2(t), \ldots, \mathbf{q}^{a^*}_N(t)\right).
\end{aligned}$$

Algorithm 3.10 gives an overview of the motion matching algorithm used in this work, defined with methods described throughout this chapter. Searches can be triggered either by a timer or some event like an abrupt control request from a user. The search interval is usually quite short (≈ 10 timesteps) to maintain responsiveness, although this can vary depending on the type of character being animated or the type of animations being used. Longer timescale effects in motion will necessitate a longer interval.

Algorithm 3.10 motion matching

initialize $\mathcal{P}^{a^*}(\cdot), t^*$
$\mathcal{P}^{\text{MM}-} \leftarrow \mathcal{P}^{a^*}(t^* - \Delta t)$
$\mathcal{P}^{\text{MM}} \leftarrow \mathcal{P}^{a^*}(t^*)$
$\mathcal{V}^{\text{MM}} \leftarrow \mathcal{V}^{a^*}(t^*)$
$\mathcal{P}^{\text{MM}_0} \leftarrow \mathcal{P}^{\text{MM}}$
$\mathcal{V}^{\text{MM}_0} \leftarrow \mathcal{V}^{\text{MM}}$
$t \leftarrow 0$
for each timestep do
    $t \leftarrow t + \Delta t$
    $\mathcal{P}^{\text{MM}-} \leftarrow \mathcal{P}^{\text{MM}}$
    $\mathcal{P}^{\text{MM}} \leftarrow \text{inertialize}(\mathcal{P}^{\text{MM}_0}, \mathcal{P}^{\text{MM}_0:a^*}(t^* + t), \mathcal{V}^{\text{MM}_0}, \mathcal{V}^{\text{MM}_0:a^*}(t^* + t), t)$
    compute $\mathcal{V}^{\text{MM}}$ **using** $\mathcal{P}^{\text{MM}}, \mathcal{P}^{\text{MM}-}$
    if ($t \geq$ **search timer**) **or search triggered then**
        construct query Q
        search for i^* **of frame best matching Q** ▷ Algorithm 3.9
        $\mathcal{P}^{a^*}(\cdot), t^* \leftarrow \text{index\_to\_anim}(i^*)$
        $\mathcal{P}^{\text{MM}_0} \leftarrow \mathcal{P}^{\text{MM}}$
        $\mathcal{V}^{\text{MM}_0} \leftarrow \mathcal{V}^{\text{MM}}$
        $t \leftarrow 0$
    end if
    output $\mathcal{P}^{\text{MM}}$
end for

3.3 Procedural Animation Touch-ups

The output of the motion matching algorithm is usually of an appreciable quality, but in some cases transitions may lead to a small amount of noticeable sliding at contacts. Because the sliding is usually over a short distance, a simple procedure using inverse kinematics can be used to eliminate it.

The details of this correction will be dependent on the character definition and the type of contacts which are expected in the animations. For the purposes of this work human characters in locomotion are considered. Given an animation $\mathcal{P}^a(t)$, a function returning the states of *contacts* is needed. This function will take the form of an ordered set $\mathcal{C}^a(t)$,

$$\mathcal{C}^a(t) = \left(b_1^a(t), \ldots, b_{N_\mathcal{C}}^a(t)\right).$$

These contact sets contain time varying indexed boolean tracks $b_k^a(t) \in \mathbb{B}, k \in \{1, \ldots, N_\mathcal{C}\}$ denoting the binary state of each contact. A state is considered active when a contact without any slipping is occurring, and inactive otherwise. The number of tracked contact states is $N_\mathcal{C}$, and is considered fixed across all animations. Contact states can be gathered from sensors during the capture, or estimated using a heuristic.

The heuristic used in this work to estimate contacts was to check the position and velocity of points of interest such as the feet and hands. A good indicator for contact being active was found to be when the point associated with it is near a known terrain feature (indicating touching) in the motion capture volume and its velocity is near zero (indicating lack of slip). The amount of contacts to track and fix is up to choice. For the purposes of locomotion tracking the contact state of the feet and hands was found to work well.

Blends may cause slipping during transitions, so a two bone inverse kinematic solutions can be used to prevent this. Two bone IK works well if the contacts are near end-effectors, as is the case for feet and hands. Each contact index $k \in \{1, \ldots, N_\mathcal{C}\}$ has associated bones for the two bone IK. The tip bone index is given by $\varsigma_k \in \{3, \ldots, N\}$, the hinge bone index is that of the tip's parent, $\mathfrak{p}(\varsigma_k)$, and the pivot bone index is that of the hinge bone's parent $\mathfrak{pp}(\varsigma_k)$. These should be picked so that no bones are associated with two IK solvers simultaneously. In this work $N_\mathcal{C} = 4$, and the indexes $\varsigma_1, \varsigma_2, \varsigma_3, \varsigma_4$ are associated with the feet and hand bones. The IK system then prevents slipping by by bending the knees and elbows, and rotating chains at the hip and shoulder joints. The rotation axis of the hinge joint connecting bone $\mathfrak{p}(\varsigma_k)$ to bone $\mathfrak{pp}(\varsigma_k)$ resolved in the frame of $\mathfrak{pp}(\varsigma_k)$ is given by unit vector $\hat{\mathbf{u}}_k \in \mathbb{S}^2$.

The axis direction in global space for pose $\mathcal{P}^a$ is then given by,

$$\hat{\overline{\mathbf{u}}}_k^a = \overline{\mathbf{q}}_{\mathtt{pp}(\varsigma_k)}^a \circ \hat{\mathbf{u}}_k.$$

A target position in global space $\overline{\mathbf{p}}_k^{\oplus}$ is also specified. The goal of two bone IK is to find the joint configuration which touches the tip to target, or gets as close as possible. Achieving this is broken into three tasks which are performed in order to obtain a solution:

1. Save the line which intersects the positions of the base and tip.
2. Rotate the hinged bone so that the distance between the base and tip of the IK chain is equal to the distance between the base and target.
3. Rotate the entire system rigidly at the base so that the line formed by the base and tip lies on the same axis it started on.
4. Rotate the entire system rigidly using the minimal rotation which moves the tip as close to the target as possible.

Items 1 and 3 are not always necessary, but can be useful to reduce some artifacts. The notation that will be adopted is that the initial pose is $\mathcal{P}^a$ and the IK adjusted pose is $\mathcal{P}^{a'}$. For the remainder of this section index k will always refer to the index of one of the IK chains. The line which initially exists between base and tip is defined as follows,

$$\begin{aligned}
\boldsymbol{r}_k^a &= \overline{\mathbf{p}}_{\varsigma_k}^a - \overline{\mathbf{p}}_{\mathtt{pp}(\varsigma_k)}^a \\
\hat{\boldsymbol{r}}_k^a &= \begin{cases} \frac{\boldsymbol{r}_k^a}{\|\boldsymbol{r}_k^a\|} & \text{if } \|\boldsymbol{r}_k^a\| > 0 \\ \mathbf{0} & \text{otherwise} \end{cases}.
\end{aligned}$$

The displacement of the target from the base is similarly defined,

$$\begin{aligned}
\boldsymbol{r}_k^{\oplus} &= \overline{\mathbf{p}}_k^{\oplus} - \overline{\mathbf{p}}_{\mathtt{pp}(\varsigma_k)}^a \\
\hat{\boldsymbol{r}}_k^{\oplus} &= \begin{cases} \frac{\boldsymbol{r}_k^{\oplus}}{\|\boldsymbol{r}_k^{\oplus}\|} & \text{if } \|\boldsymbol{r}_k^{\oplus}\| > 0 \\ \mathbf{0} & \text{otherwise} \end{cases}.
\end{aligned}$$

The projection operator and the orthogonal complement of the projection are also defined for later use,

$$\begin{aligned}\text{proj}_{\mathbf{u}}\mathbf{v} &= \frac{\mathbf{v}\cdot\mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u}\\ \text{proj}_{\mathbf{u}}^{\perp}\mathbf{v} &= \mathbf{v} - \text{proj}_{\mathbf{u}}\mathbf{v}\upsilon\\ \mathbf{v} &= \text{proj}_{\mathbf{u}}\mathbf{v} + \text{proj}_{\mathbf{u}}^{\perp}\mathbf{v}.\end{aligned}$$

The rotation of the hinge bone which puts the tip at the right distance from the base is found by considering the intersection of a sphere with a circular arc. The sphere is centered at the base and represents the distance to the target. To be general the case where the hinge bone is not perfectly orthogonal to the hinge axis is considered. The circular arc is created by all the positions of the tip rotating the hinge bone around the hinge axis through the the allowable range of motion. The various important geometric relationships, vectors, and lengths for the general two bone IK problem are summarized in Figure 3.4. With reference to the figure, various terms have the following relationships which allow them to be computed directly using constants as well as values from a global pose $\overline{\mathcal{P}}^a$,

$$\begin{aligned}l_k^{c_1} &= \left\|\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\varsigma_k}^a)\right\|\\ l_k^{c_2} &= \left\|\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\mathfrak{p}(\varsigma_k)}^a)\right\|\\ l_k^{\min} &= \left\|\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}(\overline{\boldsymbol{\ell}}_{\varsigma_k}^a) - \frac{\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\mathfrak{p}(\varsigma_k)}^a)}{l_k^{c_2}}l_k^{c_1}\right\|\\ l_k^{\max} &= \left\|\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}(\overline{\boldsymbol{\ell}}_{\varsigma_k}^a) + \frac{\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\mathfrak{p}(\varsigma_k)}^a)}{l_k^{c_2}}l_k^{c_1}\right\|\\ l_k^d &= \text{clamp}\left(\left\|\boldsymbol{r}_k^{\oplus}\right\|, l_k^{\min}, l_k^{\max}\right)\\ l_k^{c_0} &= \sqrt{(l_k^d)^2 - \left\|\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}(\overline{\boldsymbol{\ell}}_{\varsigma_k}^a) + \text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}(\overline{\boldsymbol{\ell}}_{\mathfrak{p}(\varsigma_k)}^a)\right\|^2}\\ \phi_k^{\varsigma} &= \arccos\left(\frac{\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\varsigma_k}^a)}{l_k^{c_1}}\cdot\frac{\text{proj}_{\hat{\overline{\mathbf{u}}}_k^a}^{\perp}(\overline{\boldsymbol{\ell}}_{\mathfrak{p}(\varsigma_k)}^a)}{l_k^{c_2}}\right)\\ \phi_k^{c_1} &= \arccos\left(\frac{(l_k^{c_0})^2 - (l_k^{c_1})^2 - (l_k^{c_2})^2}{-2l_k^{c_1}l_k^{c_2}}\right).\end{aligned}$$

One issue is that it may not always be feasible to find a configuration where the tip aligns with the target. In these cases the next best thing is to find the configuration which puts the tip as close as possible. This can be done by clamping the reaching distance l_k^d to stay within the distance limits which are achievable. The value $l_k^{\min}$ is the smallest distance achievable, and $l_k^{\max}$

the largest distance achievable. In the calculation of these quantities it has been assumed that the joint can only rotate within a range of $180°$. This is typical for knees and elbows which can only take on a straight or fully bent positions. This range of motion also simplifies the problem so that the number of solutions is one rather than two. The "side" of range limit is determined by choice of $\hat{\mathbf{u}}_k$ and be flipped by multiplying it by -1.

The local rotation of the hinge bone which brings the tip to the proper distance from the base is given by,

$$\mathbf{q}^{a'}_{\mathfrak{p}(\varsigma_k)} = e^{\frac{1}{2}\tilde{\hat{u}}_k(\phi^{\varsigma}_k - \phi^{c_1}_k)}\mathbf{q}^{a}_{\mathfrak{p}(\varsigma_k)}.$$

Finding the adjusted position of the tip relative to the base $\boldsymbol{d}^a_k$ then only requires forward kinematics with the adjusted values,

$$\boldsymbol{d}^a_k = \overline{\mathbf{p}}^a_{\mathfrak{p}(\varsigma_k)} + \overline{\mathbf{q}}^a_{\mathfrak{pp}(\varsigma_k)}\mathbf{q}^{a'}_{\mathfrak{p}(\varsigma_k)} \circ \overline{\boldsymbol{\ell}}^a_{\varsigma_k}.$$

The final steps require two separate rigid rotations pivoting at the base. Both are operations which involve finding the shortest rotations which align vectors. Given two vectors $\mathbf{u}$ and $\mathbf{v}$ a quaternion with favorable properties for this task can be constructed,

$$\mathbf{q}^{[\mathbf{v}\triangleleft\mathbf{u}]} = \begin{cases} \mathbf{q}_I = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^{\mathsf{T}} & \text{if } \mathbf{u} = \mathbf{0} \text{ or } \mathbf{v} = \mathbf{0} \text{ or } \hat{\mathbf{u}} \cdot \hat{\mathbf{v}} = 1, \\ \frac{1}{\sqrt{2(1+\hat{\mathbf{u}}\cdot\hat{\mathbf{v}})}} \begin{bmatrix} \hat{\mathbf{u}} \times \hat{\mathbf{v}} \\ 1 + \hat{\mathbf{u}} \cdot \hat{\mathbf{v}} \end{bmatrix} & \text{else if } |\hat{\mathbf{u}} \cdot \hat{\mathbf{v}}| < 1, \\ \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^{\mathsf{T}} & \text{else if } \left|\hat{\mathbf{u}} \cdot \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}}\right| < 1, \\ \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} & \text{otherwise.} \end{cases}$$

The proof this works as intended in the second case is not immediately obvious, but can be shown by proving the resulting quaternion is equivalent to that representing a rotation of the angle θ between the vectors around the axis orthogonal to the vectors $\hat{\mathbf{a}}$,

$$\int \frac{\sin\theta}{\sqrt{2(1+\cos\theta)}} - \sin\frac{\theta}{2}\,d\theta = 2\cos\frac{\theta}{2} - 2\sqrt{\cos^2\frac{\theta}{2}} + C$$

$$2\cos\frac{\theta}{2} - 2\sqrt{\cos^2\frac{\theta}{2}} = 0, \quad \text{if } -\pi < \theta < \pi$$

$$\implies \frac{\sin\theta}{\sqrt{2(1+\cos\theta)}} = \sin\frac{\theta}{2} \quad \text{if } -\pi < \theta < \pi$$

$$\int \frac{1+\cos\theta}{\sqrt{2(1+\cos\theta)}} - \cos\frac{\theta}{2}\,d\theta = -2\sin\frac{\theta}{2} + 2\sqrt{\cos^2\frac{\theta}{2}}\tan\frac{\theta}{2} + C$$

$$-2\sin\frac{\theta}{2} + 2\sqrt{\cos^2\frac{\theta}{2}}\tan\frac{\theta}{2} = 0 \quad \text{if } -\pi < \theta < \pi$$

$$\implies \frac{1+\cos\theta}{\sqrt{2(1+\cos\theta)}} = \cos\frac{\theta}{2} \quad \text{if } -\pi < \theta < \pi$$

$$\theta = \arccos\hat{\mathbf{u}}\cdot\hat{\mathbf{v}}$$

$$\hat{\mathbf{a}} = \frac{\hat{\mathbf{u}}\times\hat{\mathbf{v}}}{\|\hat{\mathbf{u}}\times\hat{\mathbf{v}}\|}$$

$$\begin{aligned}
\frac{1}{\sqrt{2(1+\hat{\mathbf{u}}\cdot\hat{\mathbf{v}})}}\begin{bmatrix}\hat{\mathbf{u}}\times\hat{\mathbf{v}}\\ 1+\hat{\mathbf{u}}\cdot\hat{\mathbf{v}}\end{bmatrix} &= \frac{1}{\sqrt{2(1+\cos\theta)}}\begin{bmatrix}\hat{\mathbf{a}}\sin\theta\\ 1+\cos\theta\end{bmatrix}\\
&= \begin{bmatrix}\hat{\mathbf{a}}\frac{\sin\theta}{\sqrt{2(1+\cos\theta)}}\\ \frac{1+\cos\theta}{\sqrt{2(1+\cos\theta)}}\end{bmatrix}\\
&= \begin{bmatrix}\hat{\mathbf{a}}\sin\frac{\theta}{2}\\ \cos\frac{\theta}{2}\end{bmatrix} \quad \text{if } -\pi < \theta < \pi\\
&= e^{\tilde{\hat{\mathbf{a}}}\frac{\theta}{2}} \quad \text{if } -\pi < \theta < \pi\\
&= e^{\tilde{\hat{\mathbf{a}}}\frac{\theta}{2}} \quad \text{if } |\hat{\mathbf{u}}\cdot\hat{\mathbf{v}}| < 1 \quad \square.
\end{aligned}$$

These vector alignment rotations either perform no rotation if there is no possible way to align the vectors, the unique shortest rotation if it exists, and a rotation around an arbitrary axis in cases where the vectors point opposite directions. The result is that,

$$\mathbf{q}^{[\mathbf{v}\triangleleft\mathbf{u}]}\circ\hat{\mathbf{u}} = \begin{cases}\hat{\mathbf{v}} & \text{if } (\|\mathbf{u}\| > 0 \text{ and } \|\mathbf{v}\| > 0) \text{ or } (\|\mathbf{u}\| = 0 \text{ and } \|\mathbf{v}\| = 0)\\ \hat{\mathbf{u}} & \text{if } \|\mathbf{u}\| = 0 \text{ or } \|\mathbf{v}\| = 0.\end{cases}$$

This can then be applied to complete the two bone IK correction by finding the new local and global quaternions related to the base and hinge bones,

$$\begin{aligned}
\overline{\mathbf{q}}^{a'}_{\mathfrak{pp}(\varsigma_k)} &= \mathbf{q}^{[\boldsymbol{r}^{\oplus}_k \lhd \boldsymbol{r}^a_k]} \mathbf{q}^{[\boldsymbol{r}^a_k \lhd \boldsymbol{d}^a_k]} \overline{\mathbf{q}}^{a}_{\mathfrak{pp}(\varsigma_k)} \\
\mathbf{q}^{a'}_{\mathfrak{pp}(\varsigma_k)} &= (\overline{\mathbf{q}}^{a'}_{\mathfrak{ppp}(\varsigma_k)})^{-1} \overline{\mathbf{q}}^{a'}_{\mathfrak{pp}(\varsigma_k)} \\
\overline{\mathbf{q}}^{a'}_{\mathfrak{p}(\varsigma_k)} &= \overline{\mathbf{q}}^{a'}_{\mathfrak{pp}(\varsigma_k)} \mathbf{q}^{a'}_{\mathfrak{p}(\varsigma_k)}.
\end{aligned}$$

Optionally the tip bone orientation can also be adjusted to align with some specified global target orientation $\overline{\mathbf{q}}^{\oplus}_k$,

$$\begin{aligned}
\overline{\mathbf{q}}^{a'}_{\varsigma_k} &= \overline{\mathbf{q}}^{\oplus}_k \\
\mathbf{q}^{a'}_{\varsigma_k} &= (\overline{\mathbf{q}}^{a'}_{\mathfrak{p}(\varsigma_k)})^{-1} \overline{\mathbf{q}}^{\oplus}_k.
\end{aligned}$$

All remaining bones retain their original local orientations and positions,

$$\begin{aligned}
\mathbf{q}^{a'}_i &= \mathbf{q}^a_i \quad \forall\, i \in \{1, \ldots, N\} - \{\varsigma_1, \ldots, \varsigma_{N_\mathcal{C}}\} \\
\mathbf{p}^{a'}_i &= \mathbf{p}^a_i.
\end{aligned}$$

Given a set of targets for all IK chains, the IK adjusted pose is then given by $\mathcal{P}^{a'}$. This process will be defined as a pose function which assigns the adjusted values to a new pose variable,

$$\mathcal{P}^b \leftarrow \text{TwoBoneIK}(\mathcal{P}^a, \overline{\mathbf{p}}^{\oplus}_1, \ldots, \overline{\mathbf{p}}^{\oplus}_{N_\mathcal{C}}, \overline{\mathbf{q}}^{\oplus}_1, \ldots, \overline{\mathbf{q}}^{\oplus}_{N_\mathcal{C}}) = \mathcal{P}^{a'}.$$

It is assumed the details of the hinge axis choice, number of IK groups, etc. are part of the character definition and need not be supplied as arguments.

3.3.1 Motion Matching With Foot Sliding Corrections

Contact sliding can be fixed by tracking contact state changes and using two bone IK to make adjustments gradually. The motion matching algorithm can be modified for this purpose as in Algorithm 3.11. Global IK tip positions and orientations are inertialized to and from the locked contact position after contact change events to smooth out adjustments over time. The inertialization parameters for IK need to be adjusted to achieve a blend which minimizes slipping but does not introduce significant discontinuity by occurring too quickly. This process usually requires character and style dependent adjustments. The algorithm presented also naively chooses the contact locations by projecting tip positions on planar terrain.

Algorithm 3.11 motion matching with IK

initialize $\mathcal{P}^{a^*}(\cdot), t^*$
$t \leftarrow 0$
$\mathcal{P}^{\text{MM}-} \leftarrow \mathcal{P}^{a^*}(t^* - \Delta t)$
$\mathcal{P}^{\text{MM}} \leftarrow \mathcal{P}^{a^*}(t^*)$
$\mathcal{V}^{\text{MM}} \leftarrow \mathcal{V}^{a^*}(t^*)$
$\mathcal{P}^{\text{MM}_0} \leftarrow \mathcal{P}^{\text{MM}}$
$\mathcal{V}^{\text{MM}_0} \leftarrow \mathcal{V}^{\text{MM}}$
for each $k \in \{1, \dots, N_{\mathcal{C}}\}$ **do**
    $b_k^{\mathcal{C}} \leftarrow b_k^{a^*}(t^*)$ ▷ save initial contact states
    $t_k^{\mathcal{C}} \leftarrow 0$
    $\overline{\mathbf{p}}_k^{\mathcal{C}} \leftarrow (\mathbf{1} - \hat{\mathbf{g}}\hat{\mathbf{g}}^{\mathsf{T}})\overline{\mathbf{p}}_{\varsigma_k}^{\text{MM}}$ ▷ project on terrain (plane assumed)
    $\overline{\mathbf{q}}_k^{\mathcal{C}} \leftarrow \overline{\mathbf{q}}_{\varsigma_k}^{\text{MM}}$
end for
for each timestep do
    $t \leftarrow t + \Delta t$
    $\mathcal{P}^{\text{MM}-} \leftarrow \mathcal{P}^{\text{MM}}$
    $\mathcal{P}^{\text{MM}} \leftarrow \text{inertialize}(\mathcal{P}^{\text{MM}_0}, \mathcal{P}^{\text{MM}_0:a^*}(t^* + t), \mathcal{V}^{\text{MM}_0}, \mathcal{V}^{\text{MM}_0:a^*}(t^* + t), t)$
    compute $\mathcal{V}^{\text{MM}}$ **using** $\mathcal{P}^{\text{MM}}, \mathcal{P}^{\text{MM}-}$
    for each $k \in \{1, \dots, N_{\mathcal{C}}\}$ **do**
        if $!b_k^{\mathcal{C}}$ **and** $b_k^{a^*}(t^* + t)$ **then** ▷ prevent slipping on new contact
            $b_k^{\mathcal{C}} \leftarrow$ true
            $t_k^{\mathcal{C}} \leftarrow 0$
            $\overline{\mathbf{p}}_k^{\mathcal{C}} \leftarrow (\mathbf{1} - \hat{\mathbf{g}}\hat{\mathbf{g}}^{\mathsf{T}})\overline{\mathbf{p}}_{\varsigma_k}^{\text{MM}}$ ▷ project on terrain (plane assumed)
            $\overline{\mathbf{q}}_k^{\mathcal{C}} \leftarrow \overline{\mathbf{q}}_{\varsigma_k}^{\text{MM}}$
        end if
        if $b_k^{\mathcal{C}}$ **and** $b_k^{a^*}(t^* + t)$ **then** ▷ blend/maintain contact adjustment
            $t_k^{\mathcal{C}} \leftarrow t_k^{\mathcal{C}} + \Delta t$
            $\overline{\mathbf{p}}_k^{\oplus} \leftarrow \text{inertialize}(\overline{\mathbf{p}}_{\varsigma_k}^{\text{MM}}, \overline{\mathbf{p}}_k^{\mathcal{C}}, \dot{\overline{\mathbf{p}}}_{\varsigma_k}^{\text{MM}}, \mathbf{0}, t_k^{\mathcal{C}})$
            $\overline{\mathbf{q}}_k^{\oplus} \leftarrow \text{inertialize}(\overline{\mathbf{q}}_{\varsigma_k}^{\text{MM}}, \overline{\mathbf{q}}_k^{\mathcal{C}}, \overline{\boldsymbol{\omega}}_{\varsigma_k}^{\text{MM}}, \mathbf{0}, t_k^{\mathcal{C}})$
        end if
        if $b_k^{\mathcal{C}}$ **and** $!b_k^{a^*}(t^* + t)$ **then** ▷ contact broken
            $t_k^{\mathcal{C}} \leftarrow 0$
            $b_k^{\mathcal{C}} \leftarrow$ false
        end if
        if $!b_k^{\mathcal{C}}$ **and** $!b_k^{a^*}(t^* + t)$ **then** ▷ blend away contact adjustment
            $t_k^{\mathcal{C}} \leftarrow t_k^{\mathcal{C}} + \Delta t$
            $\overline{\mathbf{p}}_k^{\oplus} \leftarrow \text{inertialize}(\overline{\mathbf{p}}_k^{\mathcal{C}}, \overline{\mathbf{p}}_{\varsigma_k}^{\text{MM}}, \mathbf{0}, \dot{\overline{\mathbf{p}}}_{\varsigma_k}^{\text{MM}}, t_k^{\mathcal{C}})$
            $\overline{\mathbf{q}}_k^{\oplus} \leftarrow \text{inertialize}(\overline{\mathbf{q}}_k^{\mathcal{C}}, \overline{\mathbf{q}}_{\varsigma_k}^{\text{MM}}, \mathbf{0}, \overline{\boldsymbol{\omega}}_{\varsigma_k}^{\text{MM}}, t_k^{\mathcal{C}})$
        end if
    end for

$\mathcal{P}^{\text{MM}'} \leftarrow \text{TwoBoneIK}(\mathcal{P}^{\text{MM}}, \overline{\mathbf{p}}^{\oplus}_0, \ldots, \overline{\mathbf{p}}^{\oplus}_{N_\mathcal{C}}, \overline{\mathbf{q}}^{\oplus}_0, \ldots, \overline{\mathbf{q}}^{\oplus}_{N_\mathcal{C}})$
if ($t \geq$ **search timer**) **or search triggered then**
  construct query Q
  search for i^* **of frame best matching Q** ▷ Algorithm 3.9
  $\mathcal{P}^{a^*}(\cdot), t^* \leftarrow \text{index\_to\_anim}(i^*)$
  $\mathcal{P}^{\text{MM}_0} \leftarrow \mathcal{P}^{\text{MM}}$
  $\mathcal{V}^{\text{MM}_0} \leftarrow \mathcal{V}^{\text{MM}}$
  $t \leftarrow 0$
end if
output $\mathcal{P}^{\text{MM}'}$
end for

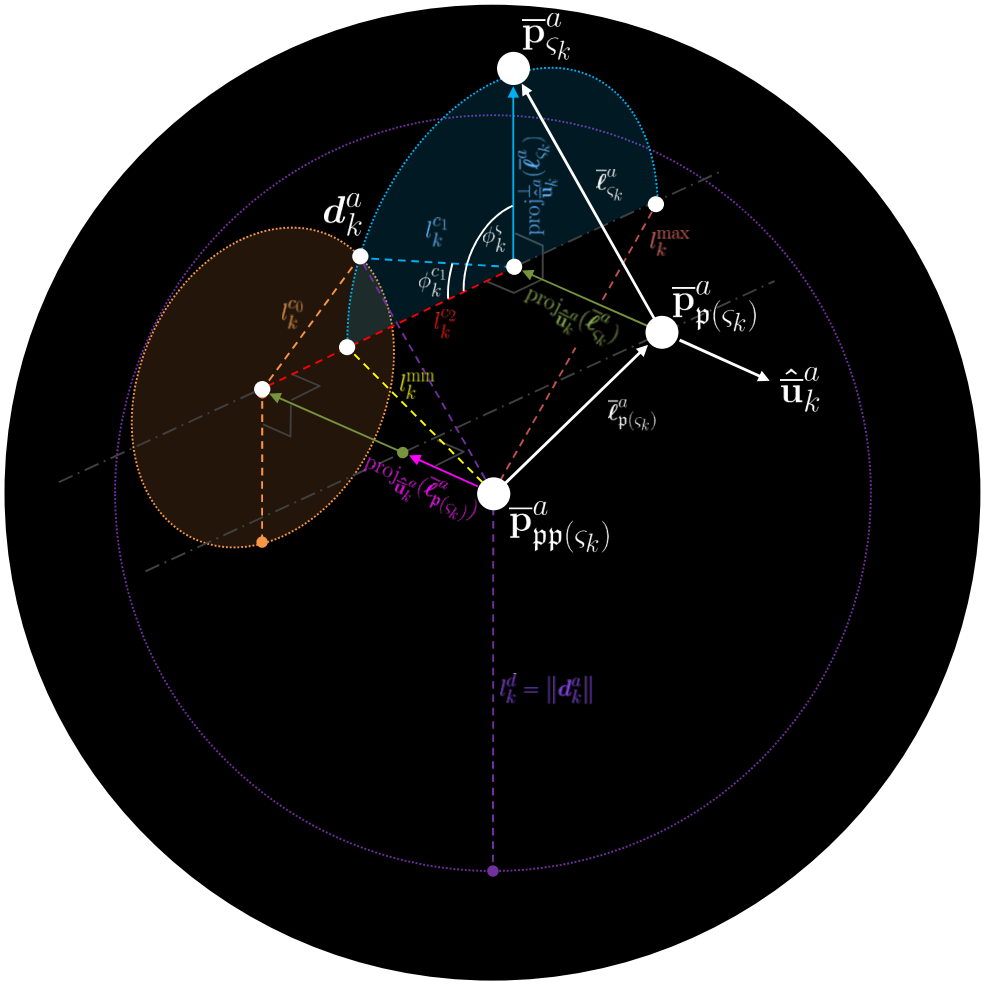


Figure 3.4: Orthographic view of geometric relationships involved in two bone IK.

Chapter 4

Physical System Modeling

Approximations are inevitable when modeling a complicated physical system. Real world physical phenomena can seldom be perfectly reproduced through computation. To make matters worse, increased accuracy usually comes at the cost of increased computational requirements. For offline simulation computational requirements are not of primary importance, but in realtime applications where computation budgets are razor thin, a delicate balance between accuracy and performance always exists. This section is devoted to developing a physical simulation of a human for use in real time simulation. The goal is to make the best use out of finite resources, making approximations where necessary.

The first big approximation made in this work is the assumption that a human can be represented by connected rigid-bodies. Humans are of course made of many interconnected stiff elements like bones, but most structure comes from deformable interconnected anisotropic soft tissues. Accurately modeling the musculoskeletal system generally requires finite element models, but these are usually far too slow for real time applications and generally reserved for scientific calculations [71]. Rigid-body physics simulations are already a well optimized and common feature of many game engines used for real time computer graphics applications, with multiple high performance open source libraries available such as NVIDIA's PhysX [72] and Bullet Physics [73]. Using an existing well maintained rigid-body physics library to model a human is the path of least resistance and prevents "reinventing the wheel" unnecessarily, so it is the approach adopted in this work.

Interconnected rigid-bodies are commonly used to model unconscious characters in video games as a "ragdoll". However, ragdolls are rarely created with the intention of *accurately* modeling the dynamics of a human, and designed instead with the intent of creating interesting animations [74]. This work aims to develop a rigid-body approximation of a human which more accurately reflects the dynamics of a real person, with the intent of reducing modeling error as much as possible for the purposes of recreating real human motions.

The objectives that were considered most important in modeling a human using rigid bodies and led to the methods contained in this chapter are the following,

- Reproduce the mass properties of an average human in a rigid-body model.
- Reproduce the surface geometry of a human which will affect behaviour during contacts.
- Develop actuation and passive dynamics of the system in a reasonable way which will allow a full body motion control.
- Use methods which are computationally efficient.

4.1 Modeling Humans

Measuring accuracy of a model always reduces to making some comparison between a model and some source of real world data. In the case of this work, the only real world data available is considered to be motion capture, and some associated character definition. Motion capture data provides basic information about geometry of a character's skeletal structure, but this does not really provide information about the physical volume occupied by an actor. Even worse, the skeletal structure is really only optimized for animating a skinned mesh, and not for accurately modeling the bones and articulations of a human skeleton. A typical video game character animation skeleton from motion capture contains a small number of bones, 68 bones being cited in [75]. This is in contrast to the 206 bones [76] of a human skeleton. Video game character skeletons also assume all bones are connected with spherical joints, and maintain a fixed distance from a single parent. While this is convenient from the standpoint of character animation, many real bones such as the shoulders are connected to a large assortment of other bones through complex structures of soft tissue, and are not generally constrained to rotate around around a fixed center of rotation. Real human joints even in "simple" structures like a knee with relatively few degrees of freedom often have eccentric sliding motions between complex joint surfaces. Nonetheless a simplifying assumption will be made that the motion capture skeleton provides centers of rotation which are at least somewhat representative of a real human. Improving the skeleton in the motion capture character to more accurately represent a human will improve the accuracy of the methods developed here.

For the purposes of this chapter a pose $\mathcal{P}^b$ will be assumed to contain a symmetric T-pose character as shown in Figure 4.1. Surface geometry of a character is generally provided through a skinned mesh with associated linear blend skinning weights [75]. It is assumed that this mesh has been calibrated to represent the geometry of the motion capture actor's body somewhat accurately. Improving the accuracy of the mesh will improve the accuracy of the resulting physical

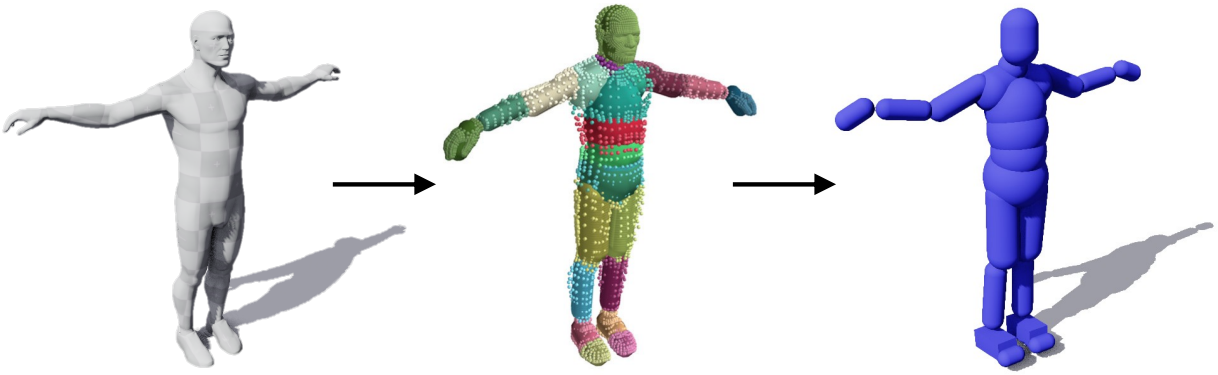


Figure 4.1: A skinned mesh is used to optimize a set of body geometries to approximate a human motion capture actor.

model being developed. A skinned mesh consists of a list of n_V vertex positions $\mathbf{V} \in \mathbb{R}^{n_V \times 3}$, a list of skinning weights $\mathbf{W} \in \mathbb{R}^{n_V \times N}$ giving the influence of each of the N character definition's bones on the vertices, and a list of triangles which are considered irrelevant for the purposes of this work. The weights for all bones corresponding to each vertex in $\mathbf{W}$ add to one,

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \dots & \mathbf{w}_{n_V} \end{bmatrix}$$

$$\mathbf{w}_i = \begin{bmatrix} w_{1,i} \\ \vdots \\ w_{N,i} \end{bmatrix}$$

$$\sum_{k=1}^{N} w_{k,i} = 1 \quad \forall\, i \in \{1, \dots, n_V\}\,.$$

The undeformed vertex positions in $\mathbf{V}$ are given for some *bind pose* which will be considered to be $\mathcal{P}^b$. The process of linear blend skinning allows different poses to modify the vertex positions in a way which smoothly deforms the mesh, sharing the different deformations of the skeleton at each point in a manner dependent on the weights associated to each bone. The new vertex positions $\mathbf{v}_i^{\text{new}}$ given an input pose $\mathcal{P}^{\text{new}}$ are determined using the bind pose,

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{n_V} \end{bmatrix} = \begin{bmatrix} v_{1,1} & \dots & v_{1,n_V} \\ \vdots & \ddots & \vdots \\ v_{3,1} & \cdots & v_{3,n_V} \end{bmatrix}$$

$$\mathbf{v}_i^{\text{new}} = \sum_{k=0}^{N} w_{k,i} \left(\overline{\mathbf{p}}_k^{\text{new}} + \overline{\mathbf{q}}_k^{\text{new}} (\overline{\mathbf{q}}_k^b)^{-1} \circ (\mathbf{v}_i - \overline{\mathbf{p}}_k^b) \right)$$

This consists of the weighted average of the rigid transformations associated with each bone. Because the weights are normalized, if an index k associated with a particular vertex i is $w_{k,i} = 1$, then bone k uniquely controls the vertex as if it were rigidly attached to it. The magnitude of the weights associated with each bone directly influence the amount that bone controls the defining surface geometry.

4.1.1 Collision Shape Optimization

Collision detection is one of the most performance heavy aspects of a physics simulation, with more complex geometry usually requiring more computation time [77]. Collision detection performance can be improved by using simple primitives such as spheres, capsules, and boxes to represent rigid-body surfaces. Skinning weights and vertices can be used to find primitive shapes which best approximate the geometry of the character.

Each bone has a simple primitive associated to it, either a box or a capsule. Choice of primitive was made manually. Boxes are used to represent feet because they provide a flat surface, and all other bones are represented by capsules. A process was developed in this work to find optimal primitive geometry parameters for each bone by using the skinned vertexes associated to them. Parameters defining the capsules are their relative position to their bone, their axial direction and length, and their radius. Parameters defining the boxes are their relative position and orientation to their bone, and their side lengths. Optimal geometry is found by performing an optimization

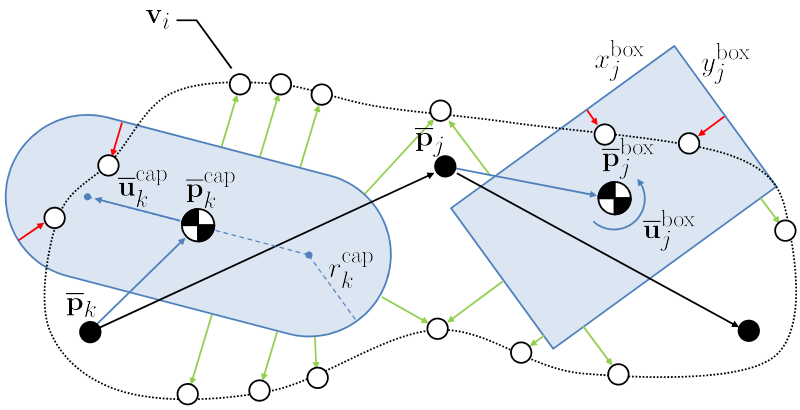


Figure 4.2: Simplified overview of the geometry optimization problem being solved.

which minimizes the distance of the primitive surface to the skinned vertexes associated to it, these distances are visualized as red and green arrows in Figure 4.2. It is also necessary to provide an additional low weight objective function term for minimizing the primitive volumes. The volume objective is necessary to obtain tightly fitting primitives, as surface distance alone will not achieve this goal in all cases.

A loss term can be formulated for each primitive by summing the weighted squared signed distance of the primitive to each skinned vertex, and adding a small regularization term for the volume minimization. For a capsule associated with bone index k the signed distance to some point $\mathbf{v}_i$ given a set of capsule defining parameters $\overline{\boldsymbol{\theta}}_k^{\text{cap}}$ can be calculated as follows,

$$\begin{aligned}
\overline{\boldsymbol{\theta}}_k^{\text{cap}} &= \begin{bmatrix} \overline{\mathbf{p}}_k^{\text{cap}\mathsf{T}} & \overline{\mathbf{u}}_k^{\text{cap}\mathsf{T}} & r_k^{\text{cap}} \end{bmatrix}^{\mathsf{T}} \\
\boldsymbol{d}_k^{\text{cap}}(\mathbf{v}_i) &= \mathbf{v}_i - \overline{\mathbf{p}}_k^{\text{cap}} \\
\text{SignDist}_{\text{cap}}(\mathbf{v}_i, \overline{\boldsymbol{\theta}}_k^{\text{cap}}) &= \left\| \boldsymbol{d}_k^{\text{cap}}(\mathbf{v}_i) - \text{clamp}\left(\boldsymbol{d}_k^{\text{cap}}(\mathbf{v}_i) \cdot \frac{\overline{\mathbf{u}}_k^{\text{cap}}}{\|\overline{\mathbf{u}}_k^{\text{cap}}\|^2}, -1, 1 \right) \overline{\mathbf{u}}_k^{\text{cap}} \right\| - r_k^{\text{cap}}.
\end{aligned}$$

The loss terms used for optimization are then straightforward to define. Surface proximity can be optimized using a weighted sum of squared signed distances over skinned vertexes, and volume can be minimized through a loss directly proportional to the capsule volume,

$$\begin{aligned}
L_{\text{surf}}^{\text{cap}}(\mathbf{V}, \overline{\boldsymbol{\theta}}_k^{\text{cap}}) &= \sum_{i=1}^{n_V} \frac{w_{k,i}}{n_V} \left(\text{SignDist}_{\text{cap}}(\mathbf{v}_i, \overline{\boldsymbol{\theta}}_k^{\text{cap}}) \right)^2 \\
L_{\text{vol}}^{\text{cap}}(\overline{\boldsymbol{\theta}}_k^{\text{cap}}) &= 2\pi \, \|\overline{\mathbf{u}}_k^{\text{cap}}\| \, (r_k^{\text{cap}})^2 + \frac{4\pi}{3} (r_k^{\text{cap}})^3.
\end{aligned}$$

In the previous definitions $\overline{\mathbf{p}}_k^{\text{cap}}$ is the position of the center of the capsule, $\overline{\mathbf{u}}_k^{\text{cap}}$ is the axis along which the capsule lies with magnitude equal to the half length, and r_k^{cap} is the radius of the capsule ends. A similar approach follows for box shaped primitives,

$$\begin{aligned}
\overline{\boldsymbol{\theta}}_k^{\text{box}} &= \begin{bmatrix} \overline{\mathbf{p}}_k^{\text{box}\mathsf{T}} & \overline{\mathbf{u}}_k^{\text{box}\mathsf{T}} & x_k^{\text{box}} & y_k^{\text{box}} & z_k^{\text{box}} \end{bmatrix}^{\mathsf{T}} \\
\boldsymbol{d}_k^{\text{box}}(\mathbf{v}_i) &= \text{abs}_{\text{E.wise}} \left(e^{-\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ (\mathbf{v}_i - \overline{\mathbf{p}}_k^{\text{box}}) - \frac{1}{2} \begin{bmatrix} x_k^{\text{box}} & y_k^{\text{box}} & z_k^{\text{box}} \end{bmatrix}^{\mathsf{T}} \right) \\
\text{SignDist}_{\text{box}}(\mathbf{v}_i, \overline{\boldsymbol{\theta}}_k^{\text{box}}) &= \left\| \max_{\text{E.wise}}(\boldsymbol{d}_k^{\text{box}}(\mathbf{v}_i), \mathbf{0}) \right\| + \min(\text{maxElement}(\boldsymbol{d}_k^{\text{box}}(\mathbf{v}_i)), 0) \\
L_{\text{surf}}^{\text{box}}(\mathbf{V}, \overline{\boldsymbol{\theta}}_k^{\text{box}}) &= \sum_{i=1}^{n_V} \frac{w_{k,i}}{n_V} \left(\text{SignDist}_{\text{box}}(\mathbf{v}_i, \overline{\boldsymbol{\theta}}_k^{\text{box}}) \right)^2 \\
L_{\text{vol}}^{\text{box}}(\overline{\boldsymbol{\theta}}_k^{\text{box}}) &= x_k^{\text{box}} y_k^{\text{box}} z_k^{\text{box}},
\end{aligned}$$

where $\overline{\boldsymbol{\theta}}_k^{\text{box}}$ contains the parameters to be optimized defining the box geometry. The position $\overline{\mathbf{p}}_k^{\text{box}}$ represents the center of the box, $\overline{\mathbf{u}}_k^{\text{box}}$ is the axis-angle parameterization of the box orientation, and $x_k^{\text{box}}, y_k^{\text{box}}, z_k^{\text{box}}$ are the dimensions of the box edges. The functions $\text{abs}_{\text{E.wise}}(\cdot)$ and $\max_{\text{E.wise}}(\cdot, \cdot)$ refer to the element-wise absolute value and maximum. The function $\text{maxElement}(\cdot)$ returns the maximum element value of the input matrix.

A global loss for all bones with a small weight $w_{\text{vol}} \approx 0.01$ for the volume terms is defined. The global parameter set for the character geometries $\overline{\boldsymbol{\theta}}_{\text{geom}}$ is defined here with $\text{shape}_k \in \{\text{cap}, \text{box}\}$ in the place of each bone k associated shape parameter's superscript to make it clear that shape choice can vary per bone. The global loss is thus defined,

$$\overline{\boldsymbol{\theta}}_{\text{geom}} = \begin{bmatrix} \overline{\boldsymbol{\theta}}_1^{\text{shape}_1} \\ \vdots \\ \overline{\boldsymbol{\theta}}_N^{\text{shape}_N} \end{bmatrix}$$

$$L^{\text{geom}}(\mathbf{V}, \overline{\boldsymbol{\theta}}_{\text{geom}}) = \sum_{k=1}^{N} L_{\text{surf}}^{\text{shape}_k}(\mathbf{V}, \overline{\boldsymbol{\theta}}_k^{\text{shape}_k}) + w_{\text{vol}} L_{\text{vol}}^{\text{shape}_k}(\overline{\boldsymbol{\theta}}_k^{\text{shape}_k}).$$

Optimal surface fitting geometry parameters $\overline{\boldsymbol{\theta}}_{\text{geom}}^{*}$ are found by solving the following optimization problem numerically,

$$\overline{\boldsymbol{\theta}}_{\text{geom}}^{*} = \arg\max_{\overline{\boldsymbol{\theta}}_{\text{geom}}} L^{\text{geom}}(\mathbf{V}, \overline{\boldsymbol{\theta}}_{\text{geom}}).$$

Stochastic gradient descent as covered in the previous sections is usually sufficient to obtain a local optima, but care should be taken to ensure some parameters are initialized to a non-zero value to prevent divisions by zero. Due to the use of conditional functions, some derivatives may be poorly defined at certain locations. This is generally handled by considering the derivative to be zero where it is difficult to define. In this work the optimization was implemented by using the machine learning library pytorch [78], this allows derivatives for the loss to be efficiently and accurately computed through automatic differentiation. Use of a machine learning library also allows the GPU to be utilized to accelerate the optimization significantly through parallelization since the loss contains many independent calculations which can later be summed together.

It is likely that the optimization will be initialized in such a way that gradient descent converges on solutions which are not close to the global optimal, particularly for the box shapes. The simplest solution is to use a brute force algorithm optimizing many random initializations of each shape's parameters and keeping the best converged solutions, for an example see Algorithm 4.12. More sophisticated algorithms like covariance matrix adaptation evolutionary strategy (CMA-ES) [79] could also be used to optimize the parameters. This geometry fitting optimization only needs to be run once to generate a useful result, so performance is not a primary concern. The skinned mesh, point cloud, and resulting character model approximated by optimized geometry primitives is show in Figure 4.1.

Algorithm 4.12 brute force geometry fitting

$\overline{\boldsymbol{\theta}}_k^* \leftarrow$ **arbitrary**
$L_k^* \leftarrow \infty$ ▷ initialize "best loss so far" with worst case
for many iterations do ▷ repeat until satisfied with result
  for each $k \in \{1, \ldots, N\}$ **do**
    for each matrix element j **of** $\overline{\boldsymbol{\theta}}_k^{\text{shape}_k}$ **do**
      $\overline{\theta}_{j,k}^{\text{shape}_k} \rightsquigarrow \text{unif}[-\phi_j, \phi_j]$ ▷ sample each element from appropriate distribution
    end for
    for each $k \in \{1, \ldots, N\}$ **do**
      ▷ perform SGD till convergence for each shape
      $\overline{\boldsymbol{\theta}}_k^{\text{SGD}}, L_k^{\text{SGD}} \leftarrow \text{SGD}(\overline{\boldsymbol{\theta}}_k^{\text{shape}_k}, L_{\text{surf}}^{\text{shape}_k}(\mathbf{V}, \overline{\boldsymbol{\theta}}_k^{\text{shape}_k}) + w_{\text{vol}} L_{\text{vol}}^{\text{shape}_k}(\overline{\boldsymbol{\theta}}_k^{\text{shape}_k}))$
      if $L_k^{\text{SGD}} < L_k^*$ **then**
        $L_k^* \leftarrow L_k^{\text{SGD}}$ ▷ L_k^{SGD} contains the loss associated with $\overline{\boldsymbol{\theta}}_k^{\text{SGD}}$
        $\overline{\boldsymbol{\theta}}_k^* \leftarrow \boldsymbol{\theta}_k^{\text{SGD}}$
      end if
    end for
  end for
end for
return $\boldsymbol{\theta}_k^*$ ▷ output the most optimal parameters found during iterations

4.1.2 Mass Property Approximation

After obtaining an optimal set of shape parameters, it is then necessary to determine how mass is distributed. A straightforward approach is to divide the mass equally among the body geometries by choosing a single appropriate density and applying it to all shapes when calculating mass properties. While this approach can be used for a reasonable approximation it neglects the fact that geometries on different bones are very likely to intersect, leading to areas with significantly higher mass concentrations due to the overlap.

In this work a method was devised to obtain a more accurate set of mass properties through a discretized integration. To begin, signed distance functions are adapted to define a function which can be used to return a mask indicating if global points are located within a shape,

$$\text{Inside}_{\text{shape}}(\overline{\mathbf{p}}, \overline{\boldsymbol{\theta}}^{\text{shape}}) = \begin{cases} 1 & \text{if SignDist}_{\text{shape}}(\overline{\mathbf{p}}, \overline{\boldsymbol{\theta}}^{\text{shape}}) < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Using this function it is possible to define mass density functions for each bone k,

$$\rho_k(\mathbf{r}) = \left(\frac{\text{Inside}_{\text{shape}_k}(\mathbf{r}, \overline{\boldsymbol{\theta}}_k^{\text{shape}_k})}{\varepsilon + \sum_{i=1}^{N} \text{Inside}_{\text{shape}_i}(\mathbf{r}, \overline{\boldsymbol{\theta}}_i^{\text{shape}_i})} \right) \frac{m_{\text{total}}}{V_{\text{total}}}.$$

Here m_{total} represents the desired total mass of the character, V_{total} is the total volume of the character, and ε is a small non-zero value with the sole purpose of preventing divisions by zero. These mass density functions are designed to share mass at each point equally among all shapes intersecting that point.

The density returned is zero where there are no intersections, which is useful when discretizing the volume integrals for mass properties. If all the character shapes are contained in some region of volume $\mathbb{V} \subset \mathbb{R}^3$, Then there is also some axis aligned bounding box volume B containing them with minimum positions $B_x^{\min}, B_y^{\min}, B_z^{\min}$ and maximum positions $B_x^{\max}, B_y^{\max}, B_z^{\max}$. These positions can be found by finding the min and max values along each dimension for the skinning vertices in $\mathbf{V}$, and adding some reasonable padding value $\Delta B \in \mathbb{R}^+$ to ensure that $\mathbb{V} \subset B$,

$$
\begin{aligned}
B_x^{\min} &= \min_i v_{0,i} - \Delta B \\
B_y^{\min} &= \min_i v_{1,i} - \Delta B \\
B_z^{\min} &= \min_i v_{2,i} - \Delta B \\
B_x^{\max} &= \max_i v_{0,i} + \Delta B \\
B_y^{\max} &= \max_i v_{1,i} + \Delta B \\
B_z^{\max} &= \max_i v_{2,i} + \Delta B.
\end{aligned}
$$

Then integration of density functions over the volume $\mathbb{V}$ can be written equivalently over the volume of the bounding box B,

$$
\mathbf{r}^{\mathbb{V}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}
$$

$$
\int\int\int_{\mathbb{V}} \rho_k(\mathbf{r}^{\mathbb{V}})\, dx\, dy\, dz = \int_{B_x^{\min}}^{B_x^{\max}} \int_{B_y^{\min}}^{B_y^{\max}} \int_{B_z^{\min}}^{B_z^{\max}} \rho_k(\mathbf{r}^{\mathbb{V}})\, dx\, dy\, dz.
$$

The box boundary makes the integral straightforward to evaluate using discretization into a $n_G \times n_G \times n_G$ cell regular grid. Each grid cell has a center position given by $\mathbf{r}^G_{i_x i_y i_z}$ where i_x, i_y, i_z are the cell indexes, and a cell volume V_G.

$$\mathbf{r}^G_{i_x i_y i_z} = \begin{bmatrix} \text{lerp}(B_x^{\min}, B_x^{\max}, \frac{i_x - 0.5}{n_G}) \\ \text{lerp}(B_y^{\min}, B_y^{\max}, \frac{i_y - 0.5}{n_G}) \\ \text{lerp}(B_z^{\min}, B_z^{\max}, \frac{i_z - 0.5}{n_G}) \end{bmatrix}$$

$$V_G = \left(\frac{B_x^{\max} - B_x^{\min}}{n_G}\right)\left(\frac{B_y^{\max} - B_y^{\min}}{n_G}\right)\left(\frac{B_z^{\max} - B_z^{\min}}{n_G}\right).$$

Using discretization the mass m_k of the rigid-body associated to bone k is approximated as follows,

$$\begin{aligned} m_k &= \int_{B_x^{\min}}^{B_x^{\max}} \int_{B_y^{\min}}^{B_y^{\max}} \int_{B_z^{\min}}^{B_z^{\max}} \rho_k(\mathbf{r}^{\mathbb{V}})\, dx\, dy\, dz \\ &\approx \sum_{i_x=1}^{n_G} \sum_{i_y=1}^{n_G} \sum_{i_z=1}^{n_G} \rho_k(\mathbf{r}^G_{i_x i_y i_z}) V_G. \end{aligned}$$

The center of mass position $\overline{\mathbf{p}}^b_{\mathcal{B}_k}$ for each rigid-body can also be approximated,

$$\begin{aligned} \overline{\mathbf{p}}^b_{\mathcal{B}_k} &= \frac{1}{m_k} \int_{B_x^{\min}}^{B_x^{\max}} \int_{B_y^{\min}}^{B_y^{\max}} \int_{B_z^{\min}}^{B_z^{\max}} \mathbf{r}^{\mathbb{V}} \rho_k(\mathbf{r}^{\mathbb{V}})\, dx\, dy\, dz \\ &\approx \frac{1}{m_k} \sum_{i_x=1}^{n_G} \sum_{i_y=1}^{n_G} \sum_{i_z=1}^{n_G} \mathbf{r}^G_{i_x i_y i_z} \rho_k(\mathbf{r}^G_{i_x i_y i_z}) V_G. \end{aligned}$$

The mass-moment of inertia for each rigid-body about the center of mass is similarly approximated,

$$\begin{aligned} \overline{\mathbf{I}}^b_{\mathcal{B}_k} &= \int\int\int_{\mathbb{V}} \left(\left\| \mathbf{r}^{\mathbb{V}} - \overline{\mathbf{p}}^b_{\mathcal{B}_k} \right\|^2 \mathbf{1} - \left(\mathbf{r}^{\mathbb{V}} - \overline{\mathbf{p}}^b_{\mathcal{B}_k}\right)\left(\mathbf{r}^{\mathbb{V}} - \overline{\mathbf{p}}^b_{\mathcal{B}_k}\right)^{\mathsf{T}} \right) \rho_k(\mathbf{r}^{\mathbb{V}})\, dx\, dy\, dz \\ &\approx \sum_{i_x=1}^{n_G} \sum_{i_y=1}^{n_G} \sum_{i_z=1}^{n_G} \left(\left\| \mathbf{r}^G_{i_x i_y i_z} - \overline{\mathbf{p}}^b_{\mathcal{B}_k} \right\|^2 \mathbf{1} - \left(\mathbf{r}^G_{i_x i_y i_z} - \overline{\mathbf{p}}^b_{\mathcal{B}_k}\right)\left(\mathbf{r}^G_{i_x i_y i_z} - \overline{\mathbf{p}}^b_{\mathcal{B}_k}\right)^{\mathsf{T}} \right) \rho_k(\mathbf{r}^G_{i_x i_y i_z}) V_G. \end{aligned}$$

So long as the grid resolution n_G is large enough the mass property approximations will be accurate enough. All quantities are calculated with respect to the global reference frame, with the character in the bind pose $\mathcal{P}^b$.

It is common that physics libraries represent rigid-body states using their center of mass position and orientation of their principal axes of inertia with respect to the world frame. If this is the case it is important for each bone to obtain the reference frame which contains the principal

moments of inertia and is centered on the center of mass. The collision shapes should then be represented with resepect to this frame.

The transformations can be carried out by first performing eigendecomposition of $\overline{\mathbf{I}}^{b}_{\mathcal{B}_k}$,

$$\overline{\mathbf{I}}^{b}_{\mathcal{B}_k} = \overline{\mathbf{R}}^{b}_{\mathcal{B}_k} \mathbf{I}_{\mathcal{B}_k} (\overline{\mathbf{R}}^{b}_{\mathcal{B}_k})^{\mathsf{T}}.$$

The matrix $\mathbf{I}_{\mathcal{B}_k}$ is diagonal with entries equal to the principal moments of inertia. It is the mass moment of inertia about the center of mass, resolved in the principal axis aligned frame. Rotation matrix $\overline{\mathbf{R}}^{b}_{\mathcal{B}_k}$ represents the orientation of the rigid-body principal axes with respect to the global frame. The quaternion parameterization $\overline{\mathbf{q}}^{b}_{\mathcal{B}_k}$ of this rotation can also be defined,

$$\overline{\mathbf{R}}^{b}_{\mathcal{B}_k} \mathbf{x} = \overline{\mathbf{q}}^{b}_{\mathcal{B}_k} \circ \mathbf{x}, \quad \forall\, \mathbf{x} \in \mathbb{R}^3.$$

Using $\overline{\mathbf{p}}^{b}_{\mathcal{B}_k}$ and $\overline{\mathbf{q}}^{b}_{\mathcal{B}_k}$, the geometry parameters relative to the principal axis aligned frame (from now on referred to as the principal frame) can be determined. For a capsule the principal frame $\boldsymbol{\theta}^{\text{cap}}_k$ geometry parameters are,

$$\begin{aligned}
\boldsymbol{\theta}^{\text{cap}}_k &= \begin{bmatrix} \mathbf{p}^{\text{cap}}_k \\ \mathbf{u}^{\text{cap}}_k \\ r^{\text{cap}}_k \end{bmatrix} \\
\mathbf{p}^{\text{cap}}_k &= (\overline{\mathbf{q}}^{b}_{\mathcal{B}_k})^{-1} \circ (\overline{\mathbf{p}}^{\text{cap}}_k - \overline{\mathbf{p}}^{b}_{\mathcal{B}_k}) \\
\mathbf{u}^{\text{cap}}_k &= (\overline{\mathbf{q}}^{b}_{\mathcal{B}_k})^{-1} \circ \overline{\mathbf{u}}^{\text{cap}}_k.
\end{aligned}$$

For a box the principal frame $\boldsymbol{\theta}^{\text{box}}_k$ geometry parameters are,

$$\begin{aligned}
\boldsymbol{\theta}^{\text{box}}_k &= \begin{bmatrix} \mathbf{p}^{\text{box}}_k \\ \mathbf{u}^{\text{box}}_k \\ x^{\text{box}}_k \\ y^{\text{box}}_k \\ z^{\text{box}}_k \end{bmatrix} \\
\mathbf{p}^{\text{box}}_k &= (\overline{\mathbf{q}}^{b}_{\mathcal{B}_k})^{-1} \circ (\overline{\mathbf{p}}^{\text{box}}_k - \overline{\mathbf{p}}^{b}_{\mathcal{B}_k}) \\
\mathbf{u}^{\text{box}}_k &= 2\, \widetilde{\log}((\overline{\mathbf{q}}^{b}_{\mathcal{B}_k})^{-1} e^{\frac{1}{2}\tilde{\mathbf{u}}^{\text{box}}_k}).
\end{aligned}$$

All principal frame geometry parameters are then contained in a single parameter list,

$$\boldsymbol{\theta}_{\text{geom}} = \begin{bmatrix} \boldsymbol{\theta}_1^{\text{shape}_1} \\ \vdots \\ \boldsymbol{\theta}_N^{\text{shape}_N} \end{bmatrix}.$$

Each rigid-body is associated uniquely to a bone k. The rigid-body and bone are assumed to transform together as a rigid unit. There is a constant transformation between the frame of the rigid-body and the frame of the bone. This transformation will be defined in terms of a local positional offset in the bone frame $\boldsymbol{\psi}_{\mathcal{B}_k} \in \mathbb{R}^3$ and a local rotational offset $\mathbf{q}_{\boldsymbol{\psi}_k}$. Utilizing bone information from the global bind pose $\overline{\mathcal{P}}^b$,

$$\begin{aligned} \boldsymbol{\psi}_{\mathcal{B}_k} &= (\overline{\mathbf{q}}_k^b)^{-1} \circ (\overline{\mathbf{p}}_{\mathcal{B}_k}^b - \overline{\mathbf{p}}_k^b) \\ \mathbf{q}_{\boldsymbol{\psi}_k} &= (\overline{\mathbf{q}}_k^b)^{-1} \overline{\mathbf{q}}_{\mathcal{B}_k}^b. \end{aligned}$$

4.2 Representing Physically Simulated Characters

The concept of a "multibody pose" is now developed. Given the global bind pose $\overline{\mathcal{P}}^b$, an associated global multibody bind pose $\overline{\mathcal{M}}^b$ is defined,

$$\overline{\mathcal{M}}^b = \left(\overline{\mathbf{p}}_{\mathcal{B}_1}^b, \ldots, \overline{\mathbf{p}}_{\mathcal{B}_N}^b, \overline{\mathbf{q}}_{\mathcal{B}_1}^b, \ldots, \overline{\mathbf{q}}_{\mathcal{B}_N}^b\right).$$

Each element of the ordered set corresponds to the position or orientation state of the rigid-body associated with the bone normally at that index in a pose's ordered set. Given arbitrary pose $\overline{\mathcal{P}}^a$, a pose function which transforms the global pose to it's corresponding global multibody pose $\overline{\mathcal{M}}^a$ can also be defined,

$$\begin{aligned} \overline{\mathcal{M}}^a &= \text{ToMultiBody}(\overline{\mathcal{P}}^a) \\ &= \left(\overline{\mathbf{p}}_1^a + \overline{\mathbf{q}}_1^a \circ \boldsymbol{\psi}_{\mathcal{B}_1}, \ldots, \overline{\mathbf{p}}_N^a + \overline{\mathbf{q}}_N^a \circ \boldsymbol{\psi}_{\mathcal{B}_N}, \overline{\mathbf{q}}_1^a \mathbf{q}_{\boldsymbol{\psi}_1}, \ldots, \overline{\mathbf{q}}_N^a \mathbf{q}_{\boldsymbol{\psi}_N}\right). \end{aligned}$$

The inverse operation is also defined,

$$\begin{aligned} \overline{\mathcal{P}}^a &= \text{FromMultiBody}(\overline{\mathcal{M}}^a) \\ &= \left(\overline{\mathbf{p}}_{\mathcal{B}_1}^a - \overline{\mathbf{q}}_{\mathcal{B}_1}^a \mathbf{q}_{\boldsymbol{\psi}_1}^{-1} \circ \boldsymbol{\psi}_{\mathcal{B}_1}, \ldots, \overline{\mathbf{p}}_{\mathcal{B}_N}^a - \overline{\mathbf{q}}_{\mathcal{B}_N}^a \mathbf{q}_{\boldsymbol{\psi}_N}^{-1} \circ \boldsymbol{\psi}_{\mathcal{B}_N}, \overline{\mathbf{q}}_{\mathcal{B}_1}^a \mathbf{q}_{\boldsymbol{\psi}_1}^{-1}, \ldots, \overline{\mathbf{q}}_{\mathcal{B}_N}^a \mathbf{q}_{\boldsymbol{\psi}_N}^{-1}\right). \end{aligned}$$

There is thus a bijection between global poses and global multibody poses. Similar to global poses, global multibody poses can be time dependendent,

$$\begin{aligned}\overline{\mathcal{M}}^a(t) &= \text{ToMultiBody}(\overline{\mathcal{P}}^a(t)) \\ \overline{\mathcal{P}}^a(t) &= \text{FromMultiBody}(\overline{\mathcal{M}}^a(t)).\end{aligned}$$

The relationship between rigid-bodies and bones ($\boldsymbol{\psi}_{\mathcal{B}_k}$ and $\mathbf{q}_{\boldsymbol{\psi}_k}$) are not time dependent in this work so the transformation functions work as expected even if time dependent arguments are used.

Global multibody pose velocities $\overline{\mathcal{W}}^a$ are also defined,

$$\overline{\mathcal{W}}^b = \left(\dot{\overline{\mathbf{p}}}^b_{\mathcal{B}_1}, \ldots, \dot{\overline{\mathbf{p}}}^b_{\mathcal{B}_N}, \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_1}, \ldots, \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_N}\right),$$

where $\dot{\overline{\mathbf{p}}}^b_{\mathcal{B}_k}$ gives the velocity of the rigid-body center of mass resolved in the global frame, and $\overline{\boldsymbol{\omega}}^b_{\mathcal{B}_k}$ gives the angular velocity of the rigid-body principal orientation resolved in the global frame. There exists a mapping between arbitrary global pose-velocities $\overline{\mathcal{V}}^a$ in a given pose and global multibody pose-velocities $\overline{\mathcal{W}}^a$, allowing velocity transformation functions to be defined,

$$\begin{aligned}\overline{\mathcal{W}}^a &= \text{ToMultiBodyVel}(\overline{\mathcal{P}}^a, \overline{\mathcal{V}}^a) \\ &= \left(\dot{\overline{\mathbf{p}}}^a_1 + \overline{\boldsymbol{\omega}}^a_1 \times \overline{\mathbf{q}}^a_1 \circ \boldsymbol{\psi}_{\mathcal{B}_1}, \ldots, \dot{\overline{\mathbf{p}}}^a_N + \overline{\boldsymbol{\omega}}^a_N \times \overline{\mathbf{q}}^a_N \circ \boldsymbol{\psi}_{\mathcal{B}_N}, \overline{\boldsymbol{\omega}}^a_1, \ldots, \overline{\boldsymbol{\omega}}^a_N\right) \\ \overline{\mathcal{V}}^a &= \text{FromMultiBodyVel}(\overline{\mathcal{M}}^a, \overline{\mathcal{W}}^a) \\ &= \left(\dot{\overline{\mathbf{p}}}^a_{\mathcal{B}_1} - \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_1} \times \overline{\mathbf{q}}^a_{\mathcal{B}_1}\mathbf{q}^{-1}_{\boldsymbol{\psi}_1} \circ \boldsymbol{\psi}_{\mathcal{B}_1}, \ldots, \dot{\overline{\mathbf{p}}}^a_{\mathcal{B}_N} - \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_N} \times \overline{\mathbf{q}}^a_{\mathcal{B}_N}\mathbf{q}^{-1}_{\boldsymbol{\psi}_N} \circ \boldsymbol{\psi}_{\mathcal{B}_N}, \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_1}, \ldots, \overline{\boldsymbol{\omega}}^b_{\mathcal{B}_N}\right).\end{aligned}$$

These are also unchanged if there is time dependence,

$$\begin{aligned}\overline{\mathcal{W}}^a(t) &= \text{ToMultiBodyVel}(\overline{\mathcal{P}}^a(t), \overline{\mathcal{V}}^a(t)) \\ \overline{\mathcal{V}}^a(t) &= \text{FromMultiBodyVel}(\overline{\mathcal{M}}^a(t), \overline{\mathcal{W}}^a(t)).\end{aligned}$$

4.2.1 Joint Constraints

Multi-body simulations of jointed objects generally take one of two common forms, *maximal coordinate simulations* and *reduced coordinate simulations* [80]. There are advantages and disadvantages to both methods. Reduced coordinate simulations are formulated such that many constrained degrees of freedom in the equations of motion being solved are eliminated. Reduced coordinate simulations allow for high accuracy, joints cannot be stretched by design since the degrees of freedom which would allow this are non-existent. The major downside is that solving for forces from external constraints such as contacts can be costly.

Maximal coordinate simulations involve simulating the equations of motion for the full 6 degrees of freedom of each rigid-body in addition to solving for constraint forces which maintain the joint and contact relationships. This naturally allows for some compliance in the joints since constraint forces preventing this may compete in the solution process. The joint drift is also a result of design choices, the primary one being that numerical solutions of equations of motion are performed at the velocity level. Drift is generally corrected by applying spring damper like correction forces [80], but for stability reasons these require multiple steps to eliminate error since stiffness cannot be too high.

The rigid-bodies forming the character must be constrained together to mimic the joints of a human. Two different types of joint models are appropriate for this purpose: hinge joints and spherical joints. Both of these joints limit all translational movement, only allowing rotation between the bodies they connect. It should be noted using these to model human joints is only an approximation, human joints are built from soft tissues which are stiff but flexible, and generally allow for limited amounts of translation. The relative rotation of bones does not occur at a fixed center of rotation either because joints function through linkages and complex sliding surfaces. While these details are not explicitly modeled, they are not altogether ignored if a maximal coordinate simulation is used.

Allowing for translational shifts in the simulation due to drift can improve model accuracy. Maximal coordinate simulations allow the elasticity and damping of the drift correction to be tweaked, which can be used to try and mimic the behaviour of the human joints being modeled. Hinge joints are used to model the connections of the toe bodies to the foot, the knee joints, and the elbow joint. All these joints of course are not perfect hinges in reality, so compliance allows for limited amounts of movement when needed. Spherical joints are used to model all other joints between rigid-bodies.

Whether the joint is a hinge joint or a spherical joint, an appropriate joint frame must be selected. All joints form a connection between two bodies. The joint has its own reference frame and position which must be specified. In this work a character has been derived from animation data so it is a natural choice to choose joints between the rigid-bodies forming the character which exactly mirror the structure of the associated bones in the animation skeleton. This makes the kinematics of the multibody character identical to those of the animated character.

Each body associated to a non-root bone $k \in \{2, \ldots, N\}$ is connected to its parent by a joint. Global joint positions are defined here as $\overline{\mathbf{p}}^b_{k_A:k_B}$, where the subscript indicates a joint between body with index k_A and body with index k_B, and the superscript indicates a pose identifier. Global joint frame orientations are similarly defined $\overline{\mathbf{q}}^b_{k_A:k_B}$. Then given a global pose $\overline{\mathcal{P}}^b$ the global joint positions and frame orientations are obtained as follows in the bind pose,

$$\begin{aligned} \overline{\mathbf{p}}^b_{\mathfrak{p}(k):k} &= \overline{\mathbf{p}}^b_k, \quad \forall\ k \in \{2, \ldots, N\} \\ \overline{\mathbf{q}}^b_{\mathfrak{p}(k):k} = \overline{\mathbf{q}}^b_{k:\mathfrak{p}(k)} &= \overline{\mathbf{q}}^b_{\mathfrak{p}(k)}. \end{aligned}$$

Note that the joint has two frame orientations associated to it since the joints allow for relative rotation, however in the bind pose these are coincident (the bind pose is used to define the default configuration of the joints). Joint position and frame orientations with respect to each of the joined bodies are constants. The local joint position with respect to body k_A is given by $\mathbf{p}_{k_A:k_B}$ and with respect to body k_B is given by $\mathbf{p}_{k_B:k_A}$. The local orientation with respect to body k_A is given by $\mathbf{q}_{k_A:k_B}$ and with respect to body k_B is given by $\mathbf{q}_{k_B:k_A}$,

$$\begin{aligned} \mathbf{p}_{\mathfrak{p}(k):k} &= (\overline{\mathbf{q}}^b_{\mathcal{B}_{\mathfrak{p}(k)}})^{-1} \circ (\overline{\mathbf{p}}^b_{\mathfrak{p}(k):k} - \overline{\mathbf{p}}^b_{\mathcal{B}_{\mathfrak{p}(k)}}), \quad \forall\ k \in \{2, \ldots, N\} \\ \mathbf{p}_{k:\mathfrak{p}(k)} &= (\overline{\mathbf{q}}^b_{\mathcal{B}_k})^{-1} \circ (\overline{\mathbf{p}}^b_{\mathfrak{p}(k):k} - \overline{\mathbf{p}}^b_{\mathcal{B}_k}) \\ \mathbf{q}_{\mathfrak{p}(k):k} &= (\overline{\mathbf{q}}^b_{\mathcal{B}_{\mathfrak{p}(k)}})^{-1} \overline{\mathbf{q}}^b_{\mathfrak{p}(k):k} \\ \mathbf{q}_{k:\mathfrak{p}(k)} &= (\overline{\mathbf{q}}^b_{\mathcal{B}_k})^{-1} \overline{\mathbf{q}}^b_{k:\mathfrak{p}(k)}. \end{aligned}$$

Using the local values it is then possible to determine the global joint frame positions and orientations in any arbitrary pose $\mathcal{P}^a$,

$$\begin{aligned} \overline{\mathbf{p}}^a_{\mathfrak{p}(k):k} &= \overline{\mathbf{q}}^a_{\mathcal{B}_{\mathfrak{p}(k)}} \circ \mathbf{p}_{\mathfrak{p}(k):k} + \overline{\mathbf{p}}^a_{\mathcal{B}_{\mathfrak{p}(k)}}, \quad \forall\ k \in \{2, \ldots, N\} \\ \overline{\mathbf{p}}^a_{k:\mathfrak{p}(k)} &= \overline{\mathbf{q}}^a_{\mathcal{B}_k} \circ \mathbf{p}_{k:\mathfrak{p}(k)} + \overline{\mathbf{p}}^a_{\mathcal{B}_k} \\ \overline{\mathbf{q}}^a_{\mathfrak{p}(k):k} &= \overline{\mathbf{q}}^a_{\mathcal{B}_{\mathfrak{p}(k)}} \mathbf{q}_{\mathfrak{p}(k):k} \\ \overline{\mathbf{q}}^a_{k:\mathfrak{p}(k)} &= \overline{\mathbf{q}}^a_{\mathcal{B}_k} \mathbf{q}_{k:\mathfrak{p}(k)}. \end{aligned}$$

The following relationship then holds true between the two frames representing the joint configuration,

$$
\begin{aligned}
(\overline{\mathbf{q}}^a_{\mathfrak{p}(k):k})^{-1}(\overline{\mathbf{q}}^a_{k:\mathfrak{p}(k)}) &= \left(\overline{\mathbf{q}}^a_{\mathcal{B}_{\mathfrak{p}(k)}}(\overline{\mathbf{q}}^b_{\mathcal{B}_{\mathfrak{p}(k)}})^{-1}\overline{\mathbf{q}}^b_{\mathfrak{p}(k):k}\right)^{-1}\left(\overline{\mathbf{q}}^a_{\mathcal{B}_k}(\overline{\mathbf{q}}^b_{\mathcal{B}_k})^{-1}\overline{\mathbf{q}}^b_{k:\mathfrak{p}(k)}\right) \\
&= \left(\overline{\mathbf{q}}^a_{\mathfrak{p}(k)}\mathbf{q}_{\psi_{\mathfrak{p}(k)}}(\overline{\mathbf{q}}^b_{\mathfrak{p}(k)}\mathbf{q}_{\psi_{\mathfrak{p}(k)}})^{-1}\overline{\mathbf{q}}^b_{\mathfrak{p}(k)}\right)^{-1}\left(\overline{\mathbf{q}}^a_k\mathbf{q}_{\psi_k}(\overline{\mathbf{q}}^b_k\mathbf{q}_{\psi_k})^{-1}\overline{\mathbf{q}}^b_{\mathfrak{p}(k)}\right) \\
&= \left(\overline{\mathbf{q}}^a_{\mathfrak{p}(k)}\right)^{-1}\left(\overline{\mathbf{q}}^a_k(\overline{\mathbf{q}}^b_k)^{-1}\overline{\mathbf{q}}^b_{\mathfrak{p}(k)}\right) \\
&= (\overline{\mathbf{q}}^a_{\mathfrak{p}(k)})^{-1}\overline{\mathbf{q}}^a_k(\overline{\mathbf{q}}^b_k)^{-1}\overline{\mathbf{q}}^b_{\mathfrak{p}(k)} \\
&= \mathbf{q}^a_k(\mathbf{q}^b_k)^{-1} \\
(&= \mathbf{q}^a_k \text{ if } \mathbf{q}^b_k \text{ is identity}).
\end{aligned}
$$

This means that if the bind pose is chosen such that all orientations are identity (this is a matter of modifying the character definition and can always be done without any negative consequence) the relative orientation of the two joint configuration frames for the joint connecting $\mathcal{B}_{\mathfrak{p}(k)}$ to body $\mathcal{B}_k$ is equal to the local orientation of bone k in the pose. This convention is adopted for convenience throughout the rest of this work as it greatly simplifies conversion of poses to joint configurations.

If a hinge joint is used then a hinge axis must be specified. This should be the same as the axis chosen for two joint IK. $\hat{\mathbf{u}}^{\text{hinge}}_k$ will be used to represent the direction of the rotation axis for a hinge joint connecting bone k to its parent.

4.3 Efficiently Simulating Physics

In this work the open source library Bullet Physics was used for physics simulation. Bullet provides a well optimized maximal coordinate rigid-body simulator with support for a variety of joint constraints. Bullet also handles collision detection and management of contact constraints. A variety of constraint solvers are available, in this work the projected Gauss-Seidel constraint solver was used.

A character definition file containing the global bind pose $\overline{\mathcal{P}}^b$, global multibody bind pose $\overline{\mathcal{M}}^b$, hinge axes, principal frame geometry parameters $\boldsymbol{\theta}_{\text{geom}}$, and mass properties was created using the procedures outlined earlier. This file will be referred to as the "sim character file".

A C++ Program was created to load the sim character file and create a multibody system in Bullet Physics which has the appropriate collision geometry, mass properties, and joint constraints. Gravity is set to 9.81 m/s^2 in the negative Z direction. An immovable infinite plane passing through the origin and normal to gravity is also created to simulate the ground.

The simulation is set up to run using 32 constraint solver iterations, with a simulation fre-

quency of 60 Hz. These values were chosen based on feedback from developers familiar with typical physics engine settings in video-game productions in order to capture the simulation quality expected in a production setting. A proprietary game engine was used to create a system for rendering graphics and handling user input. Dear ImGui [81] was used to produce a graphical user interface that allowed simulation settings to be tweaked interactively. Features were implemented to allow for a user with a mouse to move the camera around and interact with the physics simulation by applying external forces to objects.

4.3.1 Stable Actuation With Large Time-Steps

Without actuation forces the character is only a ragdoll and does not behave very realistically. Actuation allows for the simulation of muscle like forces, and also allows for the passive dynamics of the character to be controlled.

One method of providing actuation is to directly apply equal and opposite forces and torques to jointed pairs of rigid-bodies composing the character. This ensures that the forces and torques do not change the total linear momentum $\mathbf{P}$ or angular momentum $\mathbf{L}$ of the character. This can be shown using the notation of section 2.3.2,

$$\begin{aligned}\mathbf{P} &= \sum_{i=1}^{N} m_{\mathcal{B}_i} \dot{\bar{\mathbf{p}}}_{\mathcal{B}_i} \\ \mathbf{L} &= \sum_{i=1}^{N} \overline{\mathbf{R}}^{b}_{\mathcal{B}_k} \mathbf{I}_{\mathcal{B}_k} (\overline{\mathbf{R}}^{b}_{\mathcal{B}_k})^{\mathsf{T}} \overline{\boldsymbol{\omega}}_{\mathcal{B}_i}.\end{aligned}$$

Taking time derivatives the changes in momentum are shown equivalent to the sum of the forces and torques acting on the individual bodies,

$$\begin{aligned}\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}t} &= \sum_{i=1}^{N} m_{\mathcal{B}_i} \ddot{\bar{\mathbf{p}}}_{\mathcal{B}_i} = \sum_{i=1}^{N} \bar{\mathbf{f}}_{\mathcal{B}_i} = \sum_{i=1}^{N} \bar{\mathbf{f}}^{\text{ext}}_{\mathcal{B}_i} + \sum_{i=1}^{N} \bar{\mathbf{f}}^{\text{int}}_{\mathcal{B}_i} = \sum_{i=1}^{N} \bar{\mathbf{f}}^{\text{ext}}_{\mathcal{B}_i} + \mathbf{0} \\ \frac{\mathrm{d}\mathbf{L}}{\mathrm{d}t} &= \sum_{i=1}^{N} \overline{\mathbf{R}}^{b}_{\mathcal{B}_k} \mathbf{I}_{\mathcal{B}_k} (\overline{\mathbf{R}}^{b}_{\mathcal{B}_k})^{\mathsf{T}} \dot{\overline{\boldsymbol{\omega}}}_{\mathcal{B}_i} + \overline{\boldsymbol{\omega}}^{\times}_{\mathcal{B}_i} \overline{\mathbf{R}}^{b}_{\mathcal{B}_k} \mathbf{I}_{\mathcal{B}_k} (\overline{\mathbf{R}}^{b}_{\mathcal{B}_k})^{\mathsf{T}} \overline{\boldsymbol{\omega}}_{\mathcal{B}_i} \\ &= \sum_{i=1}^{N} \overline{\mathbf{m}}_{\mathcal{B}_i} = \sum_{i=1}^{N} \overline{\mathbf{m}}^{\text{ext}}_{\mathcal{B}_i} + \sum_{i=1}^{N} \overline{\mathbf{m}}^{\text{int}}_{\mathcal{B}_i} = \sum_{i=1}^{N} \overline{\mathbf{m}}^{\text{ext}}_{\mathcal{B}_i} + \mathbf{0}.\end{aligned}$$

Internal forces and torques do not change the total momentum because they sum to zero. Applying forces and torques in equal and opposite pairs to the bodies is equivalent to generating internal actuation forces connecting bodies.

One issue with applying these pairwise forces and torques to simulate muscles is that this can cause constraint violations in the simulation if large time-steps are used. In Bullet it was found that using constraints on relative angular velocity between jointed bodies provides more stable actuation when large timesteps are used. This is conceptually equivalent to applying torques on the bodies that maintain a specified angular velocity, however the actuation torques are determined by the constraint solver simultaneous to the joint constraints which prevents large constraint violations.

A desired relative orientation between bodies is achieved by changing the velocity constraint target each time-step using a simple proportional-derivative (PD) controller which attempts to track a movable position target. The relative orientation in the case of a hinge joint associated with a bone k is set by a single angular target position $\theta_k^{\oplus}$. The desired velocity of the hinge is then specified to track this position using the PD controller with a proportional gain of $\beta_{\mathrm{P}}^{\mathrm{hinge}}$ and derivative gain of $\beta_{\mathrm{D}}^{\mathrm{hinge}}$ which controls damping. Each time-step the constraint velocity is updated as follows,

$$\dot{\theta}_k^{\mathrm{hinge}} \leftarrow \beta_{\mathrm{P}}^{\mathrm{hinge}}(\theta_k^{\oplus} - \theta_k^{\mathrm{hinge}}) - \beta_{\mathrm{D}}^{\mathrm{hinge}}\dot{\theta}_k^{\mathrm{hinge}}.$$

Bullet does not implement a proper spherical joint constraint in its maximal coordinate solver so a 3-axis gimbal joint is used instead. The gimbal joint state is well described using a Euler angle parameterization. Matrix $\mathbf{R}_X(\theta_k^{\mathrm{EulerX}})$ refers to a rotation of magnitude $\theta_k^{\mathrm{EulerX}}$ around the X axis, $\mathbf{R}_Y(\theta_k^{\mathrm{EulerY}})$ refers to a rotation of magnitude $\theta_k^{\mathrm{EulerY}}$ around the Y axis, and $\mathbf{R}_Z(\theta_k^{\mathrm{EulerZ}})$ refers to a rotation of magnitude $\theta_k^{\mathrm{EulerZ}}$ around the Z axis. The 3 Euler angles control a rotation matrix $\mathbf{R}_k^{\mathrm{Euler}}$ which has a related quaternion parameterization $\mathbf{q}_k^{\mathrm{Euler}}$,

$$\begin{aligned}\mathbf{R}_k^{\mathrm{Euler}} &= \mathbf{R}_Z(\theta_k^{\mathrm{EulerZ}})\mathbf{R}_Y(\theta_k^{\mathrm{EulerY}})\mathbf{R}_X(\theta_k^{\mathrm{EulerX}}) \\ \mathbf{q}_k^{\mathrm{Euler}} \circ \mathbf{x} &= \mathbf{R}_k^{\mathrm{Euler}}\mathbf{x} \quad \forall\, \mathbf{x} \in \mathbb{R}^3.\end{aligned}$$

A PD controlled velocity constraint is used to drive each axis of the gimbal and the PD target for each of these is generated by re-parameterizing a target quaternion into the individual PD targets for the different axes.

Given the input quaternion $\mathbf{q} = [q_x \; q_y \; q_z \; \eta]^\mathsf{T}$, the XYZ order Euler angle representation is given by,

$$\text{ToEuler}(\mathbf{q}) = \begin{cases} \begin{bmatrix} \text{arctan2}(2q_x\eta - 2q_yq_z, 1 - 2q_x^2 - q_y^2) \\ \text{arcsin}(2q_xq_z + 2q_y\eta) \\ \text{arctan2}(2q_z\eta - 2q_xq_y, 1 - 2q_y^2 - 2q_z^2) \end{bmatrix} & \text{if } |2q_xq_z + 2q_y\eta| < 1 \\ \begin{bmatrix} \text{arctan2}(2q_yq_z - 2q_x\eta, 1 - 2q_x^2 - q_z^2) \\ \text{arcsin}(2q_xq_z + 2q_y\eta) \\ 0 \end{bmatrix} & \text{otherwise.} \end{cases}$$

One solution is to give a target orientation $\mathbf{q}_k^\oplus$ and translate this directly to individual gimbal-angle tracking velocities using PD control as if the gimbals were a stack of hinges,

$$\begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ \dot{\theta}_k^{\text{EulerY}} \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix} \leftarrow \beta_\text{P}^\text{Euler} \left(\text{ToEuler}(\mathbf{q}_k^\oplus) - \begin{bmatrix} \theta_X^\text{Euler} \\ \theta_Y^\text{Euler} \\ \theta_Z^\text{Euler} \end{bmatrix} \right) - \beta_\text{D}^\text{Euler} \begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ \dot{\theta}_k^{\text{EulerY}} \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix}.$$

However updating this way will not take the shortest rotational path. In order to take the shortest path it is necessary to set the angular velocity which follows the shortest path by taking into account the kinematics of the system.

Angular velocity of the system is related to the velocity of the individual gimbal angles. For an XYZ order gimbal (X is the roll axis, Y is the pitch axis, Z is the yaw axis), the local angular velocity of the gimbals with respect to their parent reference frame are the angular rates in the directions of the rotation axes,

$$\begin{aligned} \boldsymbol{\omega}_k^{\text{EulerX } w.r.t. \text{ EulerY}} &= \begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ 0 \\ 0 \end{bmatrix} \\ \boldsymbol{\omega}_k^{\text{EulerY } w.r.t. \text{ EulerZ}} &= \begin{bmatrix} 0 \\ \dot{\theta}_k^{\text{EulerY}} \\ 0 \end{bmatrix} \\ \boldsymbol{\omega}_k^{\text{EulerZ}} &= \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix}, \quad (w.r.t. \; \mathcal{B}_{\mathfrak{p}(k)} \text{ frame}). \end{aligned}$$

where the Z gimbal angular velocity is with respect to the associated bone the gimbal system is attached to. The angular velocity associated with a time varying Euler angle parameterization of rotation can be given in terms of the gimbal angle velocities,

$$\begin{aligned}
\boldsymbol{\omega}_k^{\text{Euler}} &= \boldsymbol{\omega}_k^{\text{EulerX}} + \boldsymbol{\omega}_k^{\text{EulerY}} + \boldsymbol{\omega}_k^{\text{EulerZ}} \\
&= \mathbf{R}_Z(\theta_k^{\text{EulerZ}})\mathbf{R}_Y(\theta_k^{\text{EulerY}}) \begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ 0 \\ 0 \end{bmatrix} + \mathbf{R}_Z(\theta_k^{\text{EulerZ}}) \begin{bmatrix} 0 \\ \dot{\theta}_k^{\text{EulerY}} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_k^{\text{EulerY}} \cos\theta_k^{\text{EulerZ}} & -\sin\theta_k^{\text{EulerZ}} & 0 \\ \cos\theta_k^{\text{EulerY}} \sin\theta_k^{\text{EulerZ}} & \cos\theta_k^{\text{EulerZ}} & 0 \\ -\sin\theta_k^{\text{EulerY}} & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ \dot{\theta}_k^{\text{EulerY}} \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix}.
\end{aligned}$$

Taking the inverse is possible in most cases and can be used to set the gimbal rates to achieve a specified angular velocity. The pseudo-inverse is used in gimbal locked positions where special case handling is needed. This has the implication that only the closest projection of the input angular velocity onto a subspace can be achieved in gimbal locked configurations. The following scheme is used to set gimbal rates based on $\boldsymbol{\omega}$, an arbitrary desired angular velocity,

$$\text{ToGimbal}(\boldsymbol{\omega}) = \begin{cases} \begin{bmatrix} \cos\theta_k^{\text{EulerZ}} \sec\theta_k^{\text{EulerY}} & \sin\theta_k^{\text{EulerZ}} \sec\theta_k^{\text{EulerY}} & 0 \\ -\sin\theta_k^{\text{EulerZ}} & \cos\theta_k^{\text{EulerZ}} & 0 \\ \cos\theta_k^{\text{EulerZ}} \tan\theta_k^{\text{EulerY}} & \sin\theta_k^{\text{EulerZ}} \tan\theta_k^{\text{EulerY}} & 1 \end{bmatrix} \boldsymbol{\omega} & \text{if } \left|\sin\theta_k^{\text{EulerY}}\right| < 1 \\ \begin{bmatrix} 0 & 0 & -1/2 \\ -\sin\theta_k^{\text{EulerZ}} & \cos\theta_k^{\text{EulerZ}} & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \boldsymbol{\omega} & \text{if } \sin\theta_k^{\text{EulerY}} = 1 \\ \begin{bmatrix} 0 & 0 & 1/2 \\ -\sin\theta_k^{\text{EulerZ}} & \cos\theta_k^{\text{EulerZ}} & 0 \\ 0 & 0 & 1/2 \end{bmatrix} \boldsymbol{\omega} & \text{otherwise.} \end{cases}$$

Then a PD controller based update which slerps to $\mathbf{q}_{\oplus}^{\text{Euler}}$ rather than treating each axis as a hinge can be defined,

$$\begin{bmatrix} \dot{\theta}_k^{\text{EulerX}} \\ \dot{\theta}_k^{\text{EulerY}} \\ \dot{\theta}_k^{\text{EulerZ}} \end{bmatrix} \leftarrow \text{ToGimbal}(2\beta_{\text{P}}^{\text{Euler}}\widetilde{\log}((\mathbf{q}_k^{\text{Euler}})^{-1}\mathbf{q}_k^{\oplus}) - \beta_{\text{D}}^{\text{Euler}}\boldsymbol{\omega}_k^{\text{Euler}}).$$

4.3.2 Internal Force Adjustments

The runtime cost of a physics engine increases as the time-step size decreases. Runtime cost also increases as the number of constraint solver iterations are increased. The constraint solver determines forces which project the system back onto the manifold where all constraints are satisfied. It is possible to achieve lower runtime costs at a fixed time-step size by reducing the number of constraint solver iterations, however this has the effect that the system is left in a state where the constraints are not as completely satisfied each time-step. In a maximal coordinate simulation of a humanoid this manifests as lowered stiffness in the joints. Joined bodies are easier to compress together or stretch apart.

If few constraint solver iterations are used, the low constraint stiffness can result in gravity itself compressing the character. A method of counteracting this effect by applying internal forces was devised. For each pair of jointed bodies internal forces can be applied which push bodies apart along the axis formed by their centers of mass. This force can be a constant for all bodies with magnitude f^{pressure}, acting somewhat like an internal pressure constantly resisting compression forces. Having the forces pass through the center of mass also means that no internal moments are generated. Given a character with global multibody pose $\overline{\mathcal{M}}^a$, internal forces are calculated as follows,

$$\begin{aligned}
\mathbf{f}_{k:\mathfrak{p}(k)}^{\text{int}} &= \frac{\overline{\mathbf{p}}_{\mathcal{B}_k}^a - \overline{\mathbf{p}}_{\mathcal{B}_{\mathfrak{p}(k)}}^a}{\left\| \overline{\mathbf{p}}_{\mathcal{B}_k}^a - \overline{\mathbf{p}}_{\mathcal{B}_{\mathfrak{p}(k)}}^a \right\| + \varepsilon} f^{\text{pressure}} \quad \forall\, k \in \{2, \ldots, N\} \\
\mathbf{f}_{\mathfrak{p}(k):k}^{\text{int}} &= -\mathbf{f}_{k:\mathfrak{p}(k)}^{\text{int}} \\
\mathbf{f}_k^{\text{int}} &= \mathbf{f}_{k:\mathfrak{p}(k)}^{\text{int}} + \sum_{i \in \mathfrak{c}(k)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} \\
\mathbf{f}_1^{\text{int}} &= \sum_{i \in \mathfrak{c}(1)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}}.
\end{aligned}$$

These of course sum to zero, first note that the sum of all sets of children indexes is equivalent to the set of non-root bones,

$$\mathfrak{c}(1) + \mathfrak{c}(2) + \ldots + \mathfrak{c}(N) = \{2, \ldots, N\}\,.$$

Then internal forces summing to zero can be proven by expansion,

$$\begin{aligned}
\sum_{k=1}^{N} \mathbf{f}_k^{\text{int}} &= (\mathbf{f}_1^{\text{int}}) + (\mathbf{f}_2^{\text{int}}) + \ldots + (\mathbf{f}_N^{\text{int}}) \\
&= \left(\sum_{i \in \mathfrak{c}(1)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} \right) + \left(\mathbf{f}_{2:\mathfrak{p}(2)}^{\text{int}} + \sum_{i \in \mathfrak{c}(2)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} \right) + \ldots + \left(\mathbf{f}_{N:\mathfrak{p}(N)}^{\text{int}} + \sum_{i \in \mathfrak{c}(N)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} \right) \\
&= \mathbf{f}_{2:\mathfrak{p}(2)}^{\text{int}} + \ldots + \mathbf{f}_{N:\mathfrak{p}(N)}^{\text{int}} + \sum_{i \in \mathfrak{c}(1)+\mathfrak{c}(2)+\ldots+\mathfrak{c}(N)} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} \\
&= \sum_{i=2}^{N} \mathbf{f}_{i:\mathfrak{p}(i)}^{\text{int}} + \sum_{i=2}^{N} \mathbf{f}_{\mathfrak{p}(i):i}^{\text{int}} = \sum_{i=2}^{N} \mathbf{f}_{i:\mathfrak{p}(i)}^{\text{int}} - \sum_{i=2}^{N} \mathbf{f}_{i:\mathfrak{p}(i)}^{\text{int}} = \mathbf{0}.
\end{aligned}$$

where $\varepsilon \in \mathbb{R}^+$ is a small non-zero value to prevent division by zero. These forces are tuned to counteract any character compression due to gravity in a standing position. Applying the internal forces does not necessarily decrease the accuracy of the model. In the case of a real human body intra-abdominal pressure is thought to play a significant role in supporting loads which would otherwise be managed by spinal muscles [82]. The internal forces described here play a similar role, preventing the need for the joint actuation to maintain rigidity along certain body axes, instead providing some passive anisotropic rigidity.

Chapter 5

Character Control System

The content of the previous chapters describe a wide variety of subjects. This chapter is devoted to putting it all together for the purposes of developing a control system which allows a physically simulated humanoid character to be steered by a user in a realistic manner. Additionally, everything up to this point has been designed with the intent of developing this method to have a low enough performance cost for its application to fit within realistic design constraints imposed by application in a video-game. This work attempts to be realistic about these constraints, it is assumed that games already maximize use of hardware so new methods must be extremely conservative in their use of additional resources in order for their application to be feasible.

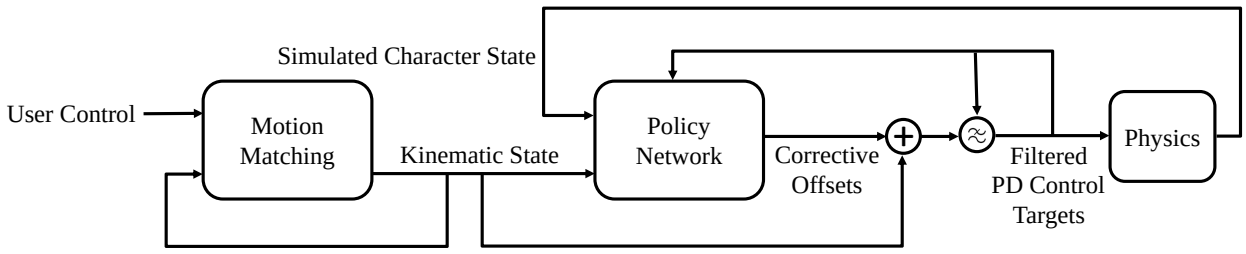


Figure 5.1: Block diagram of the control system developed in this work.

The control system is designed with the purpose of actuating a physically simulated character in a manner which tracks the output of a motion matching based kinematic character controller. A schematic is given in Figure 5.1. Previous work has demonstrated the possibility of tracking motion capture clips using deep reinforcement learning based methods. The main novel contribution of this work is a method which allows for responsive and natural looking locomotion control of a user controlled character which mirrors the behaviour of a production quality kinematic character controller. This method was published for SIGGRAPH Asia 2019 in ACM Transactions

on Graphics November 2019, titled "DReCon: data-driven responsive control of physics-based characters" [1] and is considered state-of-the-art.

5.1 Kinematic Character Controller

The kinematic character controller generates reference trajectories which are subsequently tracked by a simulated character controller. The goal is to synthesize a high quality reference motion from a motion capture database which follows user input requirements. Video-games generally require human characters to be able to perform independent control of style, movement direction, movement speed, and facing direction. There is also a requirement that the current behaviour a character is performing can abruptly change, for example a character may be running forward, but then abruptly decide to turn around and run backwards. User intent should effectively be translated to character behaviours through a simple and familiar interface. Actions such as these should occur as instantaneously as realistically possible. Additionally, the animation being generated should not come at an unrealistically high runtime cost. Motion matching meets all these demands so it was found to be a good fit.

The motion matching system used was designed around the task of locomotion. Crouching and standing locomotion were chosen as distinct styles which should be supported since they are common in third person video-games. Styles of low speed and high speed motion were also found to be important in the case of standing motion, as it is common that a character can move around at a slow pace and a hurried pace. This meant that two hard features were used (crouching/standing and walking/running), and the data is split into 3 distinct combinations,

1. Standing-walk.
2. Standing-run.
3. Crouching-walk.

Crouching-run has no data associated to it. Fine grained control of movement speed is also supported through the soft features, but a brisk walk and a slow run were found to be distinct enough to merit being classified as separate styles. Walk motions were classified as those without an aerial phase or hurried pace, and run motions as those where there is some aerial phase. For motion where the heading and movement directions are not aligned, "walk" classification was used when motion involved careful sidestepping by crossing one foot over the other, while "run" classification was used for fast sideways shuffles, this is shown in Figure 5.2. Hard feature tags were associated to ranges of frames manually using a purpose built tool. Animations $\mathcal{P}^{a_1}(\cdot), \ldots, \mathcal{P}^{a_m}(\cdot)$ are processed into a motion matching database. These total to about 10 minutes of motion capture.

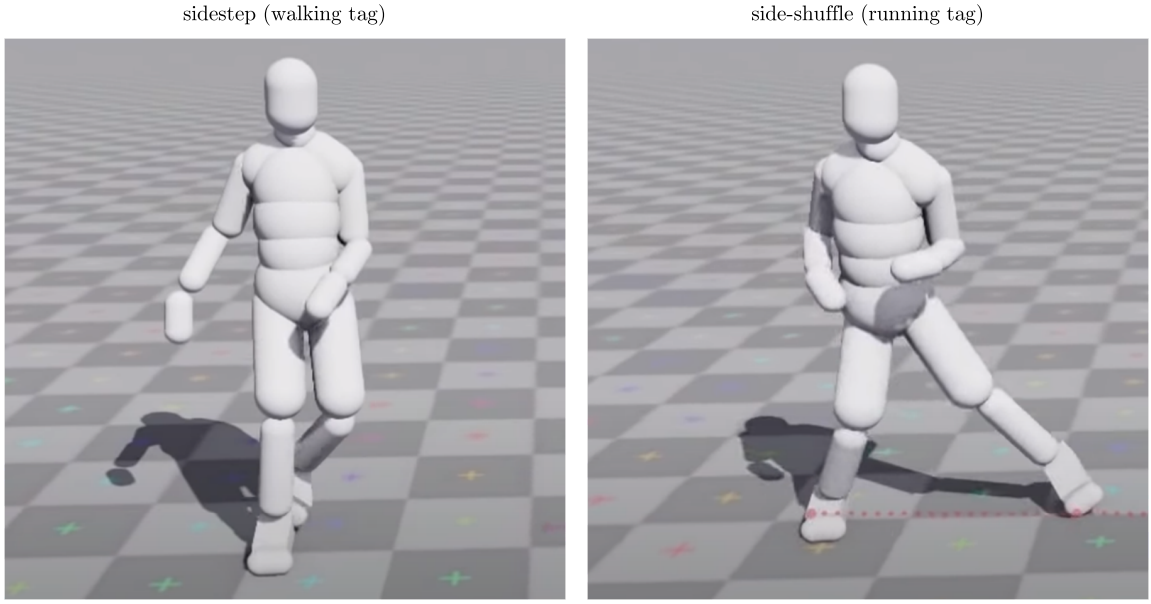


Figure 5.2: Comparison of the sideways character locomotion styles tagged as “walk” and “run”.

Each animation has associated boolean tags $\mathbf{b}^{a_1}, \ldots, \mathbf{b}^{a_m} \in \mathbb{B}^2$ denoting whether the motion is crouching/standing and walking/running.

The mapping $\mathfrak{B}(\cdot, \cdot, \cdot)$ from data to feature space will now be described. For an arbitrary animation a_i and frame j in that animation, the mapping generates a large column matrix. To simplify the description of this matrix it is split up into relevant groups of soft features denoted by an object $\mathfrak{s}^{a_{i,j}}_{\text{name}}$, where “name” can be altered. The superscript indicates a dependence on a particular animation and frame.

$$\mathfrak{B}(\mathcal{P}^{a_i}(\cdot), j, \mathbf{b}^{a_i}) = \begin{bmatrix} \mathfrak{s}^{a_{i,j}}_{\text{root-vel}} \\ \mathfrak{s}^{a_{i,j}}_{\text{left-foot-pos}} \\ \mathfrak{s}^{a_{i,j}}_{\text{left-foot-vel}} \\ \mathfrak{s}^{a_{i,j}}_{\text{right-foot-pos}} \\ \mathfrak{s}^{a_{i,j}}_{\text{right-foot-vel}} \\ \mathfrak{s}^{a_{i,j}}_{\text{trajectory}} \\ \mathfrak{s}^{a_{i,j}}_{\text{headings}} \\ \mathbf{b}^{a_i} \end{bmatrix}.$$

The time in an animation associated with frame j is dependent on the animation framerate (FPS) and will be denoted t_j,

$$t_j = j \cdot \text{FPS}^{-1}.$$

The soft feature groupings are constructed as follows, with k_{lf} and k_{rf} used to refer to the bone indexes of the left and right feet respectively,

$$\begin{aligned}
\mathfrak{s}^{a_{i,j}}_{\text{root-vel}} &= \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\dot{\overline{\mathbf{p}}}^{a_i}_{1}(t_j) \\
\mathfrak{s}^{a_{i,j}}_{\text{left-foot-pos}} &= \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\left(\overline{\mathbf{p}}^{a_i}_{k_{\text{lf}}}(t_j) - \overline{\mathbf{p}}^{a_i}_{\text{CRF}}(t_j)\right) \\
\mathfrak{s}^{a_{i,j}}_{\text{left-foot-vel}} &= \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\dot{\overline{\mathbf{p}}}^{a_i}_{k_{\text{lf}}}(t_j) \\
\mathfrak{s}^{a_{i,j}}_{\text{right-foot-pos}} &= \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\left(\overline{\mathbf{p}}^{a_i}_{k_{\text{rf}}}(t_j) - \overline{\mathbf{p}}^{a_i}_{\text{CRF}}(t_j)\right) \\
\mathfrak{s}^{a_{i,j}}_{\text{right-foot-vel}} &= \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\dot{\overline{\mathbf{p}}}^{a_i}_{k_{\text{rf}}}(t_j).
\end{aligned}$$

Trajectory and headings are represented in 2D since these features are used to steer horizontal movement and direction. The convention adopted here for gravity is $\mathbf{g} = \begin{bmatrix} 0 & 0 & -9.81\ m/s^2 \end{bmatrix}^{\mathsf{T}}$, so it is the Z components which are omitted,

$$\mathfrak{s}^{a_{i,j}}_{\text{trajectory}} = \begin{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\left(\overline{\mathbf{p}}^{a_i}_{1}(t_j + 0.33\ s) - \overline{\mathbf{p}}^{a_i}_{\text{CRF}}(t_j)\right) \\
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\left(\overline{\mathbf{p}}^{a_i}_{1}(t_j + 0.66\ s) - \overline{\mathbf{p}}^{a_i}_{\text{CRF}}(t_j)\right) \\
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\left(\overline{\mathbf{p}}^{a_i}_{1}(t_j + 1.00\ s) - \overline{\mathbf{p}}^{a_i}_{\text{CRF}}(t_j)\right)
\end{bmatrix}$$

$$\mathfrak{s}^{a_{i,j}}_{\text{headings}} = \begin{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j + 0.33\ s) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j + 0.66\ s) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j)^{-1}\overline{\mathbf{R}}^{a_i}_{\text{CRF}}(t_j + 1.00\ s) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{bmatrix}.$$

Because the future time in the trajectory/headings soft features can overflow past the length of an animation and give undefined results, data within one second from the animation ending must be omitted from the database. This prevents data near the end of an animation from being searched/matched during a query. The features are weighted equally, with the exception of hard features which are by definition given an infinitely large weighting.

5.1.1 Queries From User Input

Queries are designed to generate points in feature space which allow the kinematic character to be steered while remaining close to the manifold of animations in the database. This means that queries must search for animations which are nearby to the feature mapping of the current kinematic character state but also allow for some manipulation of the query in order to meet high-level control requirements specified by a user.

Features related to the state of a character at a given frame take on values derived from the current pose. This allows for the animation to retain consistency. Features related to the future state of the character can be manipulated in order for a desired future behaviour to be specified since this does not cause discontinuity. This steers the search to select animations which achieve a specified future. Nonetheless, it is still important for the desired future behaviour to be realizable given the current state of the character. Motion matching can generate an animation which realizes a future outcome only if such an animation exists within the database.

In order to make specification of future outcomes useful, the features related to the future are made as general as possible. The best choice found was trajectory and heading positions up to 1 second in the future with respect to the current character reference frame. A user can then specify desired values for these and almost always get a match. This is further enforced by constraining these query features to be smooth and realistic.

Video-games are usually controlled using a scheme designed to be compatible with a standard game controller that has dual analog sticks, face buttons, and analog triggers. The left stick and right stick provide 2D values constrained within the unit circle, which are represented in this work using $\mathbf{g}_{\text{l-stick}}, \mathbf{g}_{\text{r-stick}} \in \{\mathbf{x} \in \mathbb{R}^2 \mid \|\mathbf{x}\| \leq 1\}$. The left stick is used to control movement direction, and the right stick is used to control heading direction. A single analog trigger input is also considered, represented here using $g_{\text{trigger}} \in [0, 1]$. The trigger is used to control movement speed and transitions between walking and running. A single face button $g_{\text{button}} \in \mathbb{B}$ is used to control whether the character is standing or crouched.

The unnormalized query $\underline{\mathbf{Q}}$ is constructed in groups of features which correspond to those previously described for the feature space,

$$\underline{\mathbf{Q}} = \begin{bmatrix} \mathbf{q}_{\text{root-vel}} \\ \mathbf{q}_{\text{left-foot-pos}} \\ \mathbf{q}_{\text{left-foot-vel}} \\ \mathbf{q}_{\text{right-foot-pos}} \\ \mathbf{q}_{\text{right-foot-vel}} \\ \mathbf{q}_{\text{trajectory}} \\ \mathbf{q}_{\text{headings}} \\ \mathbf{q}_{\text{style}} \end{bmatrix}.$$

Queries for the locomotion controller in this work are constructed from the latest pose $\mathcal{P}^{\text{MM}}$ output by motion matching and the user input contained in $\mathbf{g}_{\text{l-stick}}, \mathbf{g}_{\text{r-stick}}, g_{\text{trigger}}, g_{\text{button}}$. The timestep for motion matching animation updates is assumed to be Δt. Using these values,

$$\begin{aligned}
\mathbf{q}_{\text{root-vel}} &= (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \dot{\overline{\mathbf{p}}}^{\text{MM}}_{1} \\
\mathbf{q}_{\text{left-foot-pos}} &= (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \left(\overline{\mathbf{p}}^{\text{MM}}_{k_{\text{lf}}} - \overline{\mathbf{p}}^{\text{MM}}_{\text{CRF}} \right) \\
\mathbf{q}_{\text{left-foot-vel}} &= (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \dot{\overline{\mathbf{p}}}^{\text{MM}}_{k_{\text{lf}}} \\
\mathbf{q}_{\text{right-foot-pos}} &= (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \left(\overline{\mathbf{p}}^{\text{MM}}_{k_{\text{rf}}} - \overline{\mathbf{p}}^{\text{MM}}_{\text{CRF}} \right) \\
\mathbf{q}_{\text{right-foot-vel}} &= (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \dot{\overline{\mathbf{p}}}^{\text{MM}}_{k_{\text{rf}}}.
\end{aligned}$$

The trajectory and heading are generated from inputs and the current state by utilizing values generated by critically damped spring dampers tracking target values. Given a constant controlling the natural frequency α and some initial conditions, the following $\boldsymbol{f}_{\text{crit}}(\cdot)$ functions give the position and velocity of a critically damped system at a specified time t tracking some constant desired target value $\mathbf{x}_d$ [83],

$$\begin{aligned}
\mathbf{x}(t) &= \boldsymbol{f}^1_{\text{crit}}(\mathbf{x}(0), \dot{\mathbf{x}}(0), \mathbf{x}_d, \alpha, t) = \mathbf{x}_d + ((\mathbf{x}(0) - \mathbf{x}_d) + (\dot{\mathbf{x}}(0) + \alpha(\mathbf{x}(0) - \mathbf{x}_d))t)e^{-\alpha t} \\
\dot{\mathbf{x}}(t) &= \boldsymbol{f}^2_{\text{crit}}(\mathbf{x}(0), \dot{\mathbf{x}}(0), \mathbf{x}_d, \alpha, t) = (\dot{\mathbf{x}}(0) - (\dot{\mathbf{x}}(0) + \alpha(\mathbf{x}(0) - \mathbf{x}_d))\alpha t)e^{-\alpha t}.
\end{aligned}$$

Using a change of variables $\dot{\mathbf{y}} = \mathbf{x}$, the previous equations can be adapted to a closed form solution of the position of a system using a spring damper to track a constant velocity $\dot{\mathbf{y}}_d = \mathbf{x}_d$,

$$\begin{aligned}
\mathbf{y}(t) &= \boldsymbol{f}^0_{\text{crit}}(\mathbf{y}(0), \dot{\mathbf{y}}(0), \ddot{\mathbf{y}}(0), \dot{\mathbf{y}}_d, \alpha, t) \\
&= \mathbf{y}(0) + \dot{\mathbf{y}}_d t + (\dot{\mathbf{y}}(0) - \dot{\mathbf{y}}_d)\frac{1 - e^{-\alpha t}}{\alpha} + (\ddot{\mathbf{y}}(0) + \alpha(\dot{\mathbf{y}}(0) - \dot{\mathbf{y}}_d))\frac{1 - (\alpha t + 1)e^{-\alpha t}}{\alpha^2} \\
\dot{\mathbf{y}}(t) &= \boldsymbol{f}^1_{\text{crit}}(\dot{\mathbf{y}}(0), \ddot{\mathbf{y}}(0), \dot{\mathbf{y}}_d, \alpha, t) = \dot{\mathbf{y}}_d + ((\dot{\mathbf{y}}(0) - \dot{\mathbf{y}}_d) + (\ddot{\mathbf{y}}(0) + \alpha(\dot{\mathbf{y}}(0) - \dot{\mathbf{y}}_d))t)e^{-\alpha t} \\
\ddot{\mathbf{y}}(t) &= \boldsymbol{f}^2_{\text{crit}}(\dot{\mathbf{y}}(0), \ddot{\mathbf{y}}(0), \dot{\mathbf{y}}_d, \alpha, t) = (\ddot{\mathbf{y}}(0) - (\ddot{\mathbf{y}}(0) + \alpha(\dot{\mathbf{y}}(0) - \dot{\mathbf{y}}_d))\alpha t)e^{-\alpha t}.
\end{aligned}$$

These simple spring damper derived values are used to generate smoothly varying future headings/trajectories for the query. The kinematic character controller has some internal states which are initialized to reasonable default values (zero or identity) and updated each time-step to prevent rapid changes,

$$\mathbf{g}_{\text{traj}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \text{lerp}(v_{\text{walk}}, v_{\text{run}}, g_{\text{trigger}})(\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{g}_{\text{l-stick}}$$

$$\begin{bmatrix} \mathbf{z}^{\text{MM}}_{\text{traj-vel}} \\ \mathbf{z}^{\text{MM}}_{\text{traj-acc}} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{f}^1_{\text{crit}}(\mathbf{z}^{\text{MM}}_{\text{traj-vel}}, \mathbf{z}^{\text{MM}}_{\text{traj-acc}}, \mathbf{g}_{\text{traj}}, \alpha_{\text{traj}}, \Delta t) \\ \boldsymbol{f}^2_{\text{crit}}(\mathbf{z}^{\text{MM}}_{\text{traj-vel}}, \mathbf{z}^{\text{MM}}_{\text{traj-acc}}, \mathbf{g}_{\text{traj}}, \alpha_{\text{traj}}, \Delta t) \end{bmatrix},$$

where v_{walk} is a value denoting the max walking speed of the character, v_{run} denotes the max running speed of the character, and α_{traj} is a tunable parameter controlling the natural frequency of the trajectory spring damper tracking the user input velocity $\mathbf{g}_{\text{traj}}$. The variable g_{head} is defined to convert the stick inputs to desired heading angles, and α_{head} is a tunable parameter controlling the natural frequency of the heading spring damper,

$$g_{\text{head}} = \begin{cases} \text{if } \|\mathbf{g}_{\text{r-stick}}\| > \text{deadzone}: \\ \quad \text{atan2}\Big(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^{\mathsf{T}} (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\mathbf{g}_{\text{r-stick}}}{\|\mathbf{g}_{\text{r-stick}}\|}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^{\mathsf{T}} (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\mathbf{g}_{\text{r-stick}}}{\|\mathbf{g}_{\text{r-stick}}\|} \Big) \\ \text{else if } \|\mathbf{g}_{\text{l-stick}}\| > \text{deadzone}: \\ \quad \text{atan2}\Big(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}^{\mathsf{T}} (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\mathbf{g}_{\text{l-stick}}}{\|\mathbf{g}_{\text{l-stick}}\|}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}^{\mathsf{T}} (\overline{\mathbf{R}}^{\text{MM}}_{\text{CRF}})^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\mathbf{g}_{\text{l-stick}}}{\|\mathbf{g}_{\text{l-stick}}\|} \Big) \\ \text{otherwise}: 0. \end{cases}$$

Note that g_{head} takes on a value aligned with the movement direction (the left stick) if the right stick is not being used. Heading also has related internal state which is updated each timestep,

$$\begin{bmatrix} \mathbf{z}^{\text{MM}}_{\text{head-dir}} \\ \mathbf{z}^{\text{MM}}_{\text{head-vel}} \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{f}^{1}_{\text{crit}}(\mathbf{z}^{\text{MM}}_{\text{head-dir}}, \mathbf{z}^{\text{MM}}_{\text{head-vel}}, [\cos g_{\text{head}} \ \ \sin g_{\text{head}}]^{\mathsf{T}}, \alpha_{\text{head}}, \Delta t) \\ \boldsymbol{f}^{2}_{\text{crit}}(\mathbf{z}^{\text{MM}}_{\text{head-dir}}, \mathbf{z}^{\text{MM}}_{\text{head-vel}}, [\cos g_{\text{head}} \ \ \sin g_{\text{head}}]^{\mathsf{T}}, \alpha_{\text{head}}, \Delta t) \end{bmatrix}.$$

The query entries for trajectory and heading are then created by evaluating the spring damper derived positions at the relevant times in the future,

$$\mathfrak{q}_{\text{trajectory}} = \begin{bmatrix} \boldsymbol{f}^{0}_{\text{crit}}(\mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{traj-vel}}, \mathbf{z}^{\text{MM}}_{\text{traj-acc}}, \mathbf{g}_{\text{traj}}, \alpha_{\text{traj}}, 0.33\ s) \\ \boldsymbol{f}^{0}_{\text{crit}}(\mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{traj-vel}}, \mathbf{z}^{\text{MM}}_{\text{traj-acc}}, \mathbf{g}_{\text{traj}}, \alpha_{\text{traj}}, 0.66\ s) \\ \boldsymbol{f}^{0}_{\text{crit}}(\mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{traj-vel}}, \mathbf{z}^{\text{MM}}_{\text{traj-acc}}, \mathbf{g}_{\text{traj}}, \alpha_{\text{traj}}, 1.00\ s) \end{bmatrix}$$

$$\mathfrak{q}_{\text{heading}} = \begin{bmatrix} \boldsymbol{f}^{1}_{\text{crit}}([1\ 0]^{\mathsf{T}}, \mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{head-dir}}, \alpha_{\text{head}}, 0.33\ s) \\ \boldsymbol{f}^{1}_{\text{crit}}([1\ 0]^{\mathsf{T}}, \mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{head-dir}}, \alpha_{\text{head}}, 0.66\ s) \\ \boldsymbol{f}^{1}_{\text{crit}}([1\ 0]^{\mathsf{T}}, \mathbf{0}, \mathbf{z}^{\text{MM}}_{\text{head-dir}}, \alpha_{\text{head}}, 1.00\ s) \end{bmatrix}.$$

The hard feature queries are contained in $\mathfrak{q}_{\text{style}}$. Qualitatively speaking the hard features are set such that,

$$\mathfrak{q}_{\text{style}} = \begin{cases} \text{crouched + walking} & \text{if } g_{\text{button}} \text{ is down} \\ \text{standing + walking} & \text{else if } g_{\text{trigger}} \leq 0.5 \\ \text{standing + running} & \text{otherwise.} \end{cases}$$

Searches are triggered every 0.166 seconds (10 frames) by the motion matching search timer, but are also triggered by large gamepad changes. The previous time-step's values of $\mathbf{g}_{\text{traj}}$ and $[\cos g_{\text{head}} \ \ \sin g_{\text{head}}]^{\mathsf{T}}$ are compared against their current value, and if the magnitude of the differences surpass a large enough threshold a motion matching search is triggered. This allows for a faster reaction to abrupt input changes.

Numeric values for constants are omitted since these are usually dataset dependent. In general any constants should be manually tuned to find which values function best, for example the natural frequency for the spring dampers used to make trajectory predictions can be tweaked to obtain higher responsiveness, however they should not be set so responsive that they constantly query for data which does not exist. Providing a visualization of the query positions for trajectory/heading features and those of the best matching animation frame will make any issues with the tuning of the constants immediately apparent and very rapid to correct. Queries should rarely be far from their matches in a properly tuned system.

5.2 Simulated Character Controller

The kinematic character controller serves to produce a visually realistic and responsive character, however it does not necessarily produce a physically achievable motion. While data may come from a physically derived source such as motion capture, motion matching is constantly blending data and applying post processing effects which have no guarantee of respecting any realistic dynamics. Techniques like inertialization and IK can be used to improve the plausibility of the dynamics, but these are ultimately just useful heuristics which produce nice looking kinematic results.

Previous research has demonstrated that motion capture reference clips can be reproduced by an actuated simulated character using deep reinforcement learning [53, 55, 56]. Interestingly it has not even been necessary to faithfully simulate a realistic human. The character imitating the motion capture can be quite different in proportion, shape, and mass distribution than the motion capture actor being imitated. Reinforcement learning allows a control policy to be found that within reasonable limits can correct for modeling error in the simulation when imitating a motion capture clip.

In this work the robustness of the reinforcement learning is leveraged to correct errors in the reference motions which are synthesized by the kinematic character controller in addition to correcting for modeling errors in the simulation. A large effort is made in ensuring plausible mass properties for the multibody character in order to reduce the modeling error as much as possible. The kinematic character controller is only expected to generate a motion plan to guide the physical character in a plausible way based on user inputs. Low level non-linear control details necessary for balance and actuation are learned by a simulated character control policy. This will be referred to simply as "the policy". Use of motion matching and heuristics constrains the motion plan to remain near the manifold of realistic human motions, minimizing the possibility that the policy will discover movement strategies with no basis in actual human behaviour.

Qualitatively, the control objective for the policy is easily defined. Get a physically simulated character to follow a user controlled kinematic reference motion as closely as possible over long timescales through internally generated actuation forces. It should not be understated that this is a tall order, the simulated character must track full-body motions involving complex contact with the environment, and the motion being followed can unexpectedly change in drastic ways since the user is a completely unpredictable agent. This means the controller must be robust to the many possible ways behaviour of the reference motion may branch. The character must also maintain these motions for long periods, falling over into unrecoverable states is not allowed.

The complexity of the control objective is the reason why RL is a good fit for this problem. Online optimization based methods like model predictive control (MPC) are also a good fit. The

issue with MPC is that it generally requires expensive computations, making its use unrealistic for application in a video-game where only a small computation budget is available. The strength of RL is that control in a variety of states can be optimized to maximize the quality of long term outcomes, but this necessitates structuring the training process such that trajectories sampling all useful system states are gathered. The policy will only learn to perform optimally for series of states which have been sampled a significant number of times, that is the policy will only be optimal when the system is within the distribution states which have been sampled. The quality of the policy is directly related to the design of the training procedure used to optimize its performance, so a great many design choices which follow in this chapter are made with the intent of manipulating the design of the policy in order to modify characteristics of the training procedure in a favorable way.

5.2.1 State Design

The state provided to the policy is designed to provide all the necessary information required to achieve the control objective. The policy must be able to infer information about the system it is affecting and about the goal it is trying to achieve in order to learn an effective closed-loop control strategy. For example it would be impossible to steer to a specified location if the policy was not given that location in some form, or if it was unaware of the position it was steering from. If critical information is omitted only open-loop control may be possible, and performance will only be optimal in an average sense with respect to the distribution of samples seen during training.

In general "information" provided to a policy does not need to be parameterized any particular way if a neural network is used to process states into actions. However, the goal of this work is to keep the systems as computationally efficient as possible. This means small, memoryless feed-forward networks are favored. These networks have a limited capacity, so the simpler the transformation from input to output is the better. Similar to the logic behind the design of features in motion matching, it is useful to represent state information in a form which is invariant to rigid transformations and rotations of the simulated character. This for example allows walking north vs. east in a straight line to be represented by the same state and minimizes the requirement of the neural network to learn which states can be considered similar through sampling.

For the purposes of this chapter the pose and pose-velocity of the simulated character at the current time-step will be given by $\mathcal{P}^{\text{sim}}$ and $\mathcal{V}^{\text{sim}}$. The current time-step kinematic controller character pose and pose-velocity are, as before, given by $\mathcal{P}^{\text{MM}}$ and $\mathcal{V}^{\text{MM}}$. The state contains information about various character bones.

A subset of bone indexes which were found to be critical are used to construct the state,

$$
\begin{aligned}
k_{\text{lf}} &\implies \text{left foot bone index} \\
k_{\text{rf}} &\implies \text{right foot bone index} \\
k_{\text{sp}} &\implies \text{middle spine bone index} \\
k_{\text{he}} &\implies \text{head bone index} \\
k_{\text{la}} &\implies \text{left forearm bone index} \\
k_{\text{ra}} &\implies \text{right forearm bone index.}
\end{aligned}
$$

Providing more than these bones was found to make little difference or in some cases actually be detrimental to the policy performance, possibly due to the increased dimensionality. The center of mass position and velocity for the whole character in an arbitrary pose $\mathcal{P}^a$ is defined as follows,

$$
\begin{aligned}
\overline{\mathbf{p}}^a_{cm} &= \frac{\sum_{k=1}^{N} m_k \overline{\mathbf{p}}^a_{\mathcal{B}_k}}{\sum_{k=1}^{N} m_k} \\
\dot{\overline{\mathbf{p}}}^a_{cm} &= \frac{\sum_{k=1}^{N} m_k \dot{\overline{\mathbf{p}}}^a_{\mathcal{B}_k}}{\sum_{k=1}^{N} m_k}.
\end{aligned}
$$

The state $\mathbf{s}$ provided to the policy each time-step is constructed by concatenating various state grouping matrices,

$$
\mathbf{s} = \begin{bmatrix}
\mathbf{s}_{\text{kin-cm-vel}} \\
\mathbf{s}_{\text{sim-cm-vel}} \\
\mathbf{s}_{\text{sim-cm-vel}} - \mathbf{s}_{\text{kin-cm-vel}} \\
\mathbf{s}_{\text{cm-desired}} \\
\mathbf{s}_{\text{cm-diff}} \\
\mathbf{s}_{\text{sim}} \\
\mathbf{s}_{\text{sim}} - \mathbf{s}_{\text{kin}} \\
\mathbf{a}^{\text{prev}}
\end{bmatrix}.
$$

These terms are defined as follows,

$$
\begin{aligned}
\mathbf{s}_{\text{kin-cm-vel}} &= (\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{cm}^{\text{MM}} \\
\mathbf{s}_{\text{sim-cm-vel}} &= (\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{cm}^{\text{sim}} \\
\mathbf{s}_{\text{cm-desired}} &= \mathbf{g}_{\text{traj}} \\
\mathbf{s}_{\text{cm-diff}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} (\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{cm}^{\text{sim}} - \mathbf{g}_{\text{traj}} \\
\mathbf{s}_{\text{sim}} &= \begin{bmatrix}
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{lf}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{rf}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{sp}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{he}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{la}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{ra}}}^{\text{sim}} - \overline{\mathbf{p}}_{cm}^{\text{sim}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{lf}}}^{\text{sim}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{rf}}}^{\text{sim}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{sp}}}^{\text{sim}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{he}}}^{\text{sim}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{la}}}^{\text{sim}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{ra}}}^{\text{sim}}
\end{bmatrix} \\
\mathbf{s}_{\text{kin}} &= \begin{bmatrix}
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{lf}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{rf}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{sp}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{he}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{la}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\left(\overline{\mathbf{p}}_{k_{\text{ra}}}^{\text{MM}} - \overline{\mathbf{p}}_{cm}^{\text{MM}}\right) \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{lf}}}^{\text{MM}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{rf}}}^{\text{MM}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{sp}}}^{\text{MM}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{he}}}^{\text{MM}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{la}}}^{\text{MM}} \\
(\overline{\mathbf{R}}_{\text{CRF}}^{\text{MM}})^{-1}\dot{\overline{\mathbf{p}}}_{k_{\text{ra}}}^{\text{MM}}
\end{bmatrix}.
\end{aligned}
$$

The exact structure of the state was derived through an iterative design procedure. The state of the simulated and kinematic character's body, limbs, and how these relate to the current user control requirements can for the most part be determined from the state. However, the choice of bones,

the choice of reference frames, and the choice to provide error terms such as $\mathbf{s}_{\text{sim-cm-vel}} - \mathbf{s}_{\text{kin-cm-vel}}$ was made not through particularly careful design decisions, but by comparing various different designs on the basis of their performance in experiments. This type of iterative design necessitated a framework which facilitated rapid experimentation and qualitative comparison. Many different state, action, reward and network architectures were tested to produce the somewhat arbitrary looking design presented here.

5.2.2 Action Design

The policy takes in the state $\mathbf{s}$ for the given time-step and produces a control action. The control action allows the policy to control actuation of the simulated character. The policy is optimized to learn an actuation strategy which meets the control objective of following the user controlled kinematic reference motion.

It is possible to just give the policy full control over the actuators, allowing it to completely manage the internal forces moving the character each frame. In practice this type of full control seems to be a poor design choice for a variety of reasons. To understand why this is the case it is important to first consider the manner which the policy is trained. Using PPO as the learning algorithm, the policy and the estimate of the value function are represented by multi-layer neural networks. As is standard practice, these neural networks have their weights initialized to small normally distributed random values centered around zero, and biases initialized to zero [65]. The output of these untrained networks for any given input is typically some arbitrary small value. This has the implication that the value function estimate starts off in a state where it supplies completely inaccurate estimates. The actions taken by the untrained policy are usually useless, randomly varying around zero due to the exploratory sampling (recall that the policy outputs a normal distribution which is sampled to select an action). The result of all this is that training begins by exploring randomly around a 'zero' action, and continues for some time until the value function estimate accuracy improves. The value function estimate only improves in states which have been sampled, and the sampling is quite arbitrary since the policy has been selecting actions without any clear intention.

As is the case for any non-convex numeric optimization, the quality of the local optima found are dependent on where the optimization is initialized. This brings up the question of how the initialization can be improved. One useful strategy is to modify the training to encourage useful states to be sampled more. This can be done by initializing the simulated character in poses generated by the kinematic character controller each episode, and reducing the sampling of unfavorable states such as those where the character has fallen over by prematurely ending the episode [55]. Biasing the training like this is certainly useful, but another strategy can be used as well. Imagine the policy could be initialized such that it were already near a well performing

optima, then the optimization would be likely to converge on this optima. Equivalently, the output of the policy can be transformed in such a way that even an untrained network is close to performing the correct action.

In the case of following the kinematic character motion, a naive approach is to set PD control targets for each actuator on the simulated character that attempt to track the kinematic character pose by matching joint angles. This makes the simulated character vaguely mime the motions of the kinematic character, and if their poses are similar enough the tracking will be of decent quality at least for a small duration. Of course this "open-loop" approach provides no feedback to correct global differences in the pose, so it is not be able to maintain global alignment of the kinematic and simulated characters for very long. Nonetheless this open-loop control is *close* to performing the correct motion. It turns out that such a simple controller provides a much better starting point than an untrained neural network providing PD control targets directly.

The strategy used to improve the quality of the initialization is to consider the output of the policy to be "corrections" to the naive open-loop PD targets. The naive system provides feedforward control, and the policy closes the loop by providing feedback control. This means that the untrained network then outputs values centered around zero that cause actuation nearly equivalent to the open-loop control. This immediately improves the initial performance of the policy, and biases the sampling to actions which are already performing basic tracking.

Another factor to consider is the dimension of the actions. A policy which outputs a larger dimension action will generally be more difficult to train as each action dimension requires extra exploration to discover what behaviours are optimal in a given state. Larger action dimensions will need more sampling and will also likely require larger networks to obtain good performance. For this reason the dimension of the action should be kept as small as possible. An alternative to having the policy output an action for every actuator is to have it output offsets for only a subset of the actuators. The remaining actuators can instead be managed completely by open-loop control. This allows the amount of sampling required to be significantly reduced, and also prevents some undesirable local optima where actuators compete against each other. The actuators associated with the following bone indexes are used, the associated bodies are visualized in Figure 5.3.

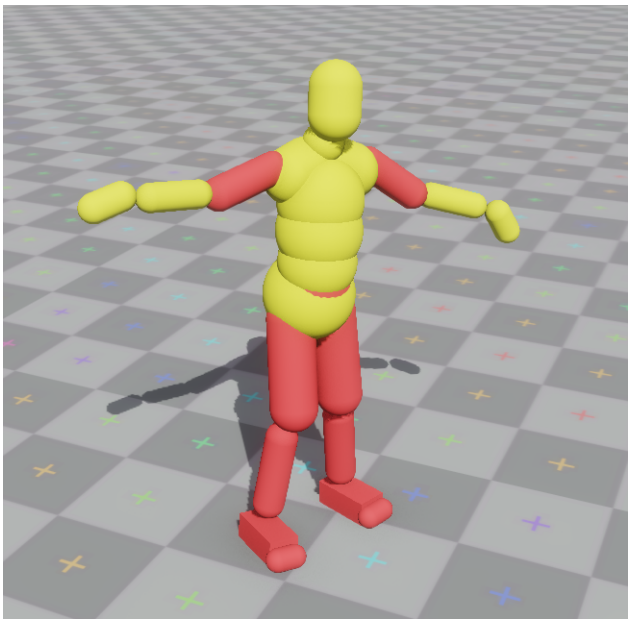


Figure 5.3: Character bodies with parent joints managed purely through open-loop control are coloured yellow here, while those that have parent joint actuation modified by policy actions are coloured red.

$$
\begin{aligned}
k_{\text{lu}} &\implies \text{left upper leg bone index} \\
k_{\text{ll}} &\implies \text{left leg bone index} \\
k_{\text{lf}} &\implies \text{left foot bone index} \\
k_{\text{lt}} &\implies \text{left toe bone index} \\
k_{\text{ru}} &\implies \text{right upper leg bone index} \\
k_{\text{rl}} &\implies \text{right leg bone index} \\
k_{\text{rf}} &\implies \text{right foot bone index} \\
k_{\text{rt}} &\implies \text{right toe bone index} \\
k_{\text{sp}} &\implies \text{middle spine bone index} \\
k_{\text{he}} &\implies \text{head bone index} \\
k_{\text{le}} &\implies \text{left arm bone index} \\
k_{\text{re}} &\implies \text{right arm bone index.}
\end{aligned}
$$

An inherent issue with utilizing reinforcement learning for this control task is that random sampling of actions leads to sampling of trajectories with noisy variations in actuation. This is an issue because ultimately the policy learns to perform tracking which performs well over long timescales, so the negative effects of high frequency oscillations during tracking are mostly av-

eraged away. Over the course of long periods of time both high frequency and low frequency oscillation around the tracking target result in similar returns, so the optimization will not particularly favor one over the other. If the policy samples trajectories with high frequency oscillations the outcome of training will almost invariably be a policy at a local optima which tracks with high frequency oscillations. An example step response visualizing this issue is shown in Figure 5.4. The solutions to this issue adopted in this work were to constrain the policy to perform smoother actuation and to limiting the frequency at which the policy operates. Smoothed actuation is performed by filtering the policy output each time-step using an exponentially weighted moving average filter. The frequency at which the policy operates is limited by only allowing the

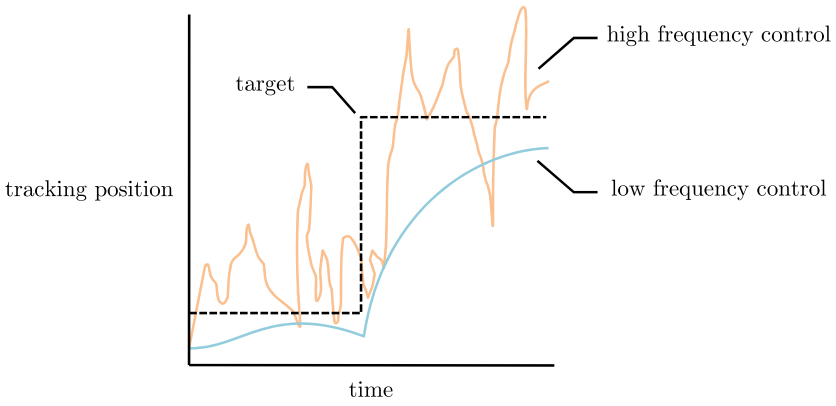


Figure 5.4: The step response of high frequency controllers may be better than low frequency controllers and yield better average rewards, but can easily be much noisier.

policy output to update after a specified number of simulation steps and holding it constant at the previous value during steps where it has not updated. While this of course introduces some delay, it also improves the overall performance. Not only are high frequency variations limited, but action quality also improves because credit assignment is facilitated during training. With fewer actions in a given interval of time it is less likely that actions with a negative effect are learned since subsequent actions cannot immediately counteract mistakes. With a lower evaluation frequency trajectories with high returns can only be achieved through action choices with a meaningful positive effect.

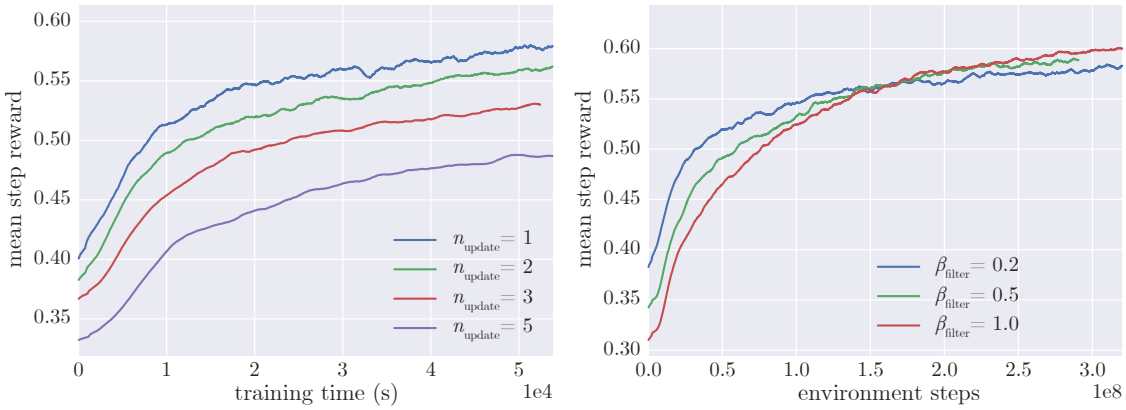
The action output by the policy is a matrix containing terms to set the hinge and spherical joint target offsets. While hinge target offsets are well represented using a scalar value, spherical joint target offsets are orientations and as such an appropriate rotation parameterization must be chosen. A parameterization without algebraic constraints is preferred as then the neural network will not need to learn how to constrain its output.

To keep the output unconstrained a 3 component axis-angle parameterization is chosen. The action is then constructed as follows,

$$\mathbf{a} \leftarrow \left[\boldsymbol{a}_{k_{\text{lu}}}^{\mathsf{T}} \; a_{k_{\text{ll}}} \; \boldsymbol{a}_{k_{\text{lf}}}^{\mathsf{T}} \; a_{k_{\text{lt}}} \; \boldsymbol{a}_{k_{\text{ru}}}^{\mathsf{T}} \; a_{k_{\text{rl}}} \; \boldsymbol{a}_{k_{\text{rf}}}^{\mathsf{T}} \; a_{k_{\text{rt}}} \; \boldsymbol{a}_{k_{\text{sp}}}^{\mathsf{T}} \; \boldsymbol{a}_{k_{\text{he}}}^{\mathsf{T}} \; \boldsymbol{a}_{k_{\text{le}}}^{\mathsf{T}} \; \boldsymbol{a}_{k_{\text{re}}}^{\mathsf{T}} \right]^{\mathsf{T}}.$$

where all non-scalar values are axis-angle parameterizations representing spherical joint target offsets, e.g. $\boldsymbol{a}_k \in \mathbb{R}^3$, and all scalar values represent hinge joint target offsets. The components comprising $\mathbf{a}$ are given indices associated with the joints they relate to, for example $\boldsymbol{a}_k$ refers to a spherical joint target offset for the joint connecting $\mathcal{B}_k$ to its parent $\mathcal{B}_{\mathfrak{p}(k)}$. These action values are output by the policy, filtered, and converted into actuation targets.

The processing of actions into actuation of the simulated character is detailed in Algorithm 5.13. Note that $\mathbf{a}^{\text{prev}}$ which was included in the state appears in the algorithm. This term is included in the state $\mathbf{s}$ so that the dynamic effects of the filter can be known to the policy. The value of n_{update} controls the policy evaluation frequency. In this work $n_{\text{update}} = 2$ was found to be a good choice, having the policy provide an updated action every other time-step. Models were trained with various n_{update} values to determine which worked best, the results of this comparison are shown in Figure 5.5a. Lower evaluation frequencies can significantly impact average rewards, but they also reduce noise and runtime cost. Training at a lowered rate then increasing it later to a desirable value can also eliminate noise, but then runtime cost savings are lost. The filter constant



(a) Reducing policy evaluation frequency using larger n_{update} tends to result in lower average rewards, but less noisy motion and lower runtime cost.

(b) Effect of β_{filter} on trained policy performance. Paradoxically stronger filtering produces qualitatively better results, even if average rewards are lower.

Figure 5.5: Effect of filter design variations.

β_{filter} controls the strength of a recursively calculated exponentially weighted moving average filter. A setting of $\beta_{\text{filter}} = 0.2$ was found to be a good choice which reduces high frequency noise in trained policies. Models trained with various values of β_{filter} were compared, with results

shown in Figure 5.5b. Although lack of filtering is shown to lead to higher average rewards, a visual comparison of the trained policies reveals that strong filtering produces qualitatively nicer character motion with less noisy movement. A neural network is used to represent the mean of the policy probability density function for a given state and has parameters θ, that is $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}) = \mathcal{F}_{\pi}(\mathbf{s}, \boldsymbol{\theta})$. The policy takes on the form of a multivariate normal distribution for each state, $\pi_{\boldsymbol{\theta}}(\mathbf{s}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{s}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$. The covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is not state dependent, and has only diagonal entries (only variances are non-zero). The variances are included as parameters in $\boldsymbol{\theta}$ which can be optimized during training. The neural network used for the policy takes on a simple form, with 110 input units since $\mathbf{s} \in \mathbb{R}^{110}$, and 25 output units since $\mathbf{a} \in \mathbb{R}^{25}$.

The value function estimate associated with policy $\pi_{\boldsymbol{\theta}}$ is represented by a separate neural network with parameters $\boldsymbol{\phi}_{\boldsymbol{\theta}}$, that is $V_{\boldsymbol{\theta}}(\mathbf{s}) = \mathcal{F}_{V}(\mathbf{s}, \boldsymbol{\phi}_{\boldsymbol{\theta}})$. This network has 110 input units since $\mathbf{s} \in \mathbb{R}^{110}$, and a single output unit representing the value function estimate. The value function estimate network has the same number of hidden layers with the same width as the network used for the policy.

The activation functions chosen for all units were tanh. ReLU activations are usually favoured in machine learning because they prevent vanishing and exploding graidents better than other activation functions [65], but in this work it was observed that tanh activations tended to produce better visual quality output. Trained policies using tanh gave smoother character movement compared to trained policies using ReLU, possibly due to the small network sizes. A similar effect can be found for small neural networks performing binary classification. As shown in Figure 5.6, the classification boundary for ReLU networks take on a polygonal shape, while tanh networks produce a smooth boundary. It is likely a similar type of sharp polygonal structure exists when using ReLU in the output of the trained policy given small variations in state, leading to the unfavorable jerky movements which were observed in experiments. The hidden layer count and

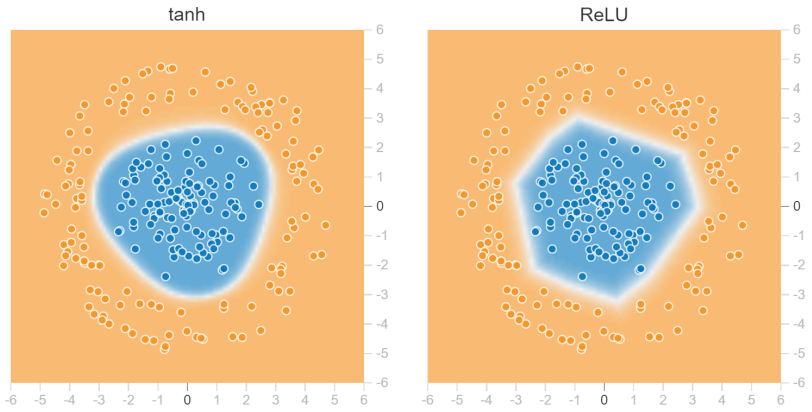


Figure 5.6: Visualization of the binary classification boundaries produced by small neural networks (2 hidden layers, 4 units wide) using tanh and ReLU activation functions.

Algorithm 5.13 action application

initialize kinematic character controller & simulation
$\mathbf{a}^{\text{prev}} \leftarrow \mathbf{0}$
$i \leftarrow n_{\text{update}}$
for each timestep do
  step kinematic character controller to update $\mathcal{P}^{\text{MM}}$, $\mathcal{V}^{\text{MM}}$
  if $i \geq n_{\text{update}}$ **then** ▷ reduce policy evaluation frequency
    $i \leftarrow 0$
    $\mathbf{s} \leftarrow$ update using latest $\mathcal{P}^{\text{MM}}$, $\mathcal{V}^{\text{MM}}$, $\mathcal{P}^{\text{sim}}$, $\mathcal{V}^{\text{sim}}$, $\mathbf{a}^{\text{prev}}$
    $\mathbf{a} \leftsquigarrow \pi_{\boldsymbol{\theta}}(\mathbf{s})$
  end if
  $\mathbf{a}^{\text{filter}} \leftarrow \beta_{\text{filter}}\mathbf{a} + (1 - \beta_{\text{filter}})\mathbf{a}^{\text{prev}}$
  for each $k \in \{$**set of hinge joint indexes**$\}$ **do**
    if $k \in \left\{k_{\text{lu}}, k_{\text{ll}}, k_{\text{lf}}, k_{\text{lt}}, k_{\text{ru}}, k_{\text{rl}}, k_{\text{rf}}, k_{\text{rt}}, k_{\text{sp}}, k_{\text{he}}, k_{\text{le}}, k_{\text{re}}\right\}$ **then**
      (add filtered offset to kinematic pose targets on policy controlled joints)
      $\theta_k^{\oplus} \leftarrow 2\,\widetilde{\log}(\mathbf{q}_k^{\text{MM}}) \cdot \hat{\mathbf{u}}_k^{\text{hinge}} + a_k^{\text{filter}}$
    else
      (track kinematic pose directly on open-loop joints)
      $\theta_k^{\oplus} \leftarrow 2\,\widetilde{\log}(\mathbf{q}_k^{\text{MM}}) \cdot \hat{\mathbf{u}}_k^{\text{hinge}}$ ▷ dot product to get signed hinge angle
    end if
    $\dot{\theta}_k^{\text{hinge}} \leftarrow \beta_{\text{P}}^{\text{hinge}}(\theta_k^{\oplus} - \theta_k^{\text{hinge}}) - \beta_{\text{D}}^{\text{hinge}}\dot{\theta}_k^{\text{hinge}}$
  end for
  for each $k \in \{$**set of spherical joint indexes**$\}$ **do**
    if $k \in \left\{k_{\text{lu}}, k_{\text{ll}}, k_{\text{lf}}, k_{\text{lt}}, k_{\text{ru}}, k_{\text{rl}}, k_{\text{rf}}, k_{\text{rt}}, k_{\text{sp}}, k_{\text{he}}, k_{\text{le}}, k_{\text{re}}\right\}$ **then**
      (filtered offset rotations combined with pose on policy controlled joints)
      $\mathbf{q}_k^{\oplus} \leftarrow \mathbf{q}_k^{\text{MM}} e^{\frac{1}{2}\tilde{\boldsymbol{a}}_k^{\text{filter}}}$
    else
      (track kinematic pose directly on open-loop joints)
      $\mathbf{q}_k^{\oplus} \leftarrow \mathbf{q}_k^{\text{MM}}$
    end if
    $\begin{bmatrix}\dot{\theta}_k^{\text{EulerX}} \\ \dot{\theta}_k^{\text{EulerY}} \\ \dot{\theta}_k^{\text{EulerZ}}\end{bmatrix} \leftarrow \text{ToGimbal}(2\beta_{\text{P}}^{\text{Euler}}\widetilde{\log}((\mathbf{q}_k^{\text{Euler}})^{-1}\mathbf{q}_k^{\oplus}) - \beta_{\text{D}}^{\text{Euler}}\boldsymbol{\omega}_k^{\text{Euler}})$
  end for
  step simulation forward to update $\mathcal{P}^{\text{sim}}$, $\mathcal{V}^{\text{sim}}$
  $\mathbf{a}^{\text{prev}} \leftarrow \mathbf{a}^{\text{filter}}$
  $i \leftarrow i + 1$
end for

width were chosen by comparing different settings in experiments and choosing the design which produced the best results while minimizing runtime cost. A comparison of different hidden unit counts is shown in Figure 5.7. Two hidden layers were found to perform equally well to larger numbers of hidden layers. The width of the individual layers was chosen to be 128 units, as smaller widths seemed to reduce tracking performance, and larger widths trained too slowly because of increased runtime costs.

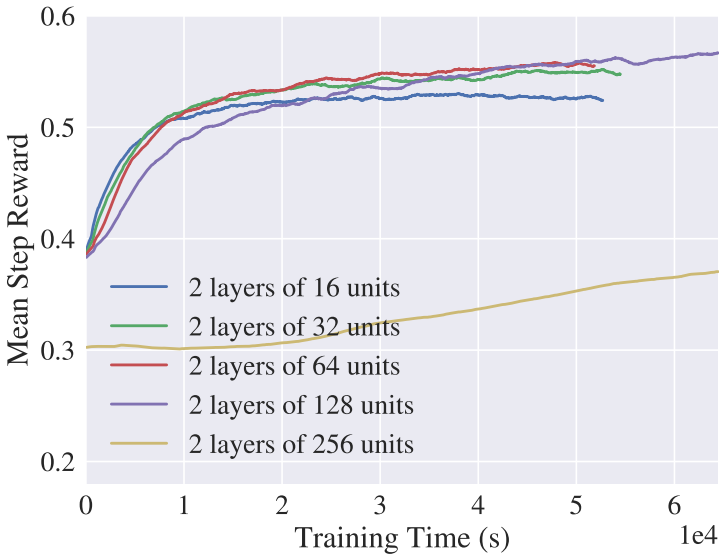


Figure 5.7: Effect of varying neural network hidden unit count on policy performance.

5.2.3 Reward Design

Up to this point the control objective has been qualitatively described. This section covers the design of the reward signal which encourages the policy to achieve the objective. In simplified terms the control objective is to have the simulated character follow a trajectory. In classical control trajectory tracking problems have been well studied. One useful idea to use for our purposes is that of a nominal trajectory. An arbitrary second order differential equation representing a physical system with time varying control forces $\mathbf{u}(t)$ can be rewritten in the following first order form,

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)).$$

A nominal trajectory consists of a time varying state $\underline{\mathbf{x}}(t)$ and control force $\underline{\mathbf{u}}(t)$ that are a solution to the differential equation,

$$\dot{\underline{\mathbf{x}}}(t) = \mathbf{f}(\underline{\mathbf{x}}(t), \underline{\mathbf{u}}(t)),$$

also known as the nominal solution. Any arbitrary time varying state and set of control forces can be written in terms of a nominal trajectory and perturbations $\delta\underline{\mathbf{x}}(t), \delta\underline{\mathbf{u}}(t)$ to the nominal trajectory,

$$\begin{aligned}
\mathbf{x}(t) &= \underline{\mathbf{x}}(t) + \delta\underline{\mathbf{x}}(t) \\
\mathbf{u}(t) &= \underline{\mathbf{u}}(t) + \delta\underline{\mathbf{u}}(t) \\
\dot{\underline{\mathbf{x}}}(t) + \delta\dot{\underline{\mathbf{x}}}(t) &= \mathbf{f}(\underline{\mathbf{x}}(t) + \delta\underline{\mathbf{x}}(t), \underline{\mathbf{u}}(t) + \delta\underline{\mathbf{u}}(t)).
\end{aligned}$$

The system can be linearized around the nominal trajectory by neglecting higher order terms of its Taylor expansion. The nominal solution can be subtracted and this results in a linear time-varying system,

$$\begin{aligned}
\mathbf{A}(t) &= \left.\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}(t)}\right|_{\mathbf{x}(t)=\underline{\mathbf{x}}(t), \mathbf{u}(t)=\underline{\mathbf{u}}(t)} \\
\mathbf{B}(t) &= \left.\frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)}\right|_{\mathbf{x}(t)=\underline{\mathbf{x}}(t), \mathbf{u}(t)=\underline{\mathbf{u}}(t)} \\
\delta\dot{\underline{\mathbf{x}}}(t) &= \mathbf{A}(t)\delta\underline{\mathbf{x}}(t) + \mathbf{B}(t)\delta\underline{\mathbf{u}}(t).
\end{aligned}$$

Given that the control objective is to follow a given trajectory, it is useful to think in terms of nominal trajectories. The reason any of this is of interest to the design of the reward is due to the important theoretical implications nominal trajectories have. If a physically achievable desired trajectory is given, it will be possible to find a set of time varying open-loop control values which together with the desired trajectory form a nominal trajectory. In general this means a set of open-loop control values exist starting from a certain state on the nominal trajectory that continue to follow the trajectory without any feedback. Note that not every desired trajectory has an associated nominal trajectory, for example it is impossible for a pig to fly (a desired trajectory with no associated nominal trajectory), but it is possible for it to walk around any which way with its eyes closed (desired trajectories which do have associated nominal trajectories). If a nominal trajectory exists then for small perturbations the dynamics of the perturbations are well approximated by the linear time varying differential equation. This means that a course correction back onto the nominal trajectory can be achieved for small perturbations by finding control forces which reduce the state perturbation to zero.

In classical control globally optimal controllers for linear time varying systems can be found if the objective function is convex. For trajectory tracking tasks the typical strategy is to find the control forces which minimize a cost that is quadratic with respect to the perturbed state and control magnitude. This defines a linear quadratic regulator (LQR) and under conditions where

the system is controllable and weighting matrices $\mathbf{Q}(t), \mathbf{R}(t)$ are specified a *globally* optimal controller can be found,

$$\mathbf{J}_\pi(\delta\underline{\mathbf{x}}(t)) = \int_0^\infty \left(\delta\underline{\mathbf{x}}^\mathsf{T}(t)\mathbf{Q}(t)\delta\underline{\mathbf{x}}(t) + \delta\underline{\mathbf{u}}^\mathsf{T}(t)\mathbf{R}(t)\delta\underline{\mathbf{u}}(t)\right) dt$$
$$\arg\min_\pi \mathbf{J}_\pi(\delta\underline{\mathbf{x}}(t)) = \pi^*(\delta\underline{\mathbf{x}}(t)) = -\mathbf{K}(t)\delta\underline{\mathbf{x}}(t).$$

The main takeaway is that for trajectory tracking with small perturbations the optimal controller will be some form of time varying linear state feedback control. Costs are simply negative rewards so this is similar to finding the policy which maximizes a value function arising from a quadratic reward.

In the context of this work the kinematic reference motion is the desired trajectory. The open-loop controls are $\approx \underline{\mathbf{u}}(t)$ and the kinematic reference motion poses are $\approx \underline{\mathbf{x}}(t)$. These can be considered to form a pseudo nominal trajectory. While the open-loop controls previously described do not allow the character to follow the reference motion indefinitely they still partially perform this task. The state of the simulated character can be considered $\mathbf{x}(t) = \underline{\mathbf{x}}(t) + \delta\underline{\mathbf{x}}(t)$, and thus the difference between state of the simulated character and the kinematic reference gives the state perturbation $\delta\underline{\mathbf{x}}(t) = \mathbf{x}(t) - \underline{\mathbf{x}}(t)$. The output of the policy is transformed into a small perturbation to the open-loop targets, the policy effectively outputs $\delta\underline{\mathbf{u}}(t)$.

Given this knowledge of trajectory tracking problems in a classical control context the advantage of specifying the actions as offsets from an open-loop controller becomes apparent. With an appropriately constructed reward small perturbations to the optimal policy should resemble linear state feedback and the transformation of terms resembling perturbations in the state to control actions will be a simple relationship which is easy for a neural network to approximate.

The cost in the trajectory tracking LQR formulation involves weighted terms which are quadratic in the state perturbation and quadratic in the control perturbation. If the weight matrix is identity then the state perturbations are equivalent to squared distances since $\mathbf{x}^\mathsf{T}\mathbf{x} = \|\mathbf{x}\|^2$. The minimum cost will be zero and will occur when the perturbation is zero. The rewards will be designed with the same qualitative behaviour in mind taking on a maximum value when the tracking is aligned with the trajectory. The reward is structured into multiple terms which are combined together,

$$r = r_{\text{fall}}(r_{\text{pos}} + r_{\text{vel}} + r_{\text{local}} + r_{\text{cm-vel}}).$$

Each frame the pose and pose-velocity of the simulated character $\mathcal{P}^{\text{sim}}, \mathcal{V}^{\text{sim}}$ and kinematic reference character $\mathcal{P}^{\text{MM}}, \mathcal{V}^{\text{MM}}$ are used to calculate the values composing the reward terms. In order to compare position and orientation information simultaneously multiple reference points $\boldsymbol{v}_k^i, i \in \{1, \ldots, 6\}$ rigidly attached to the each body $k \in \{1, \ldots, N\}$ are used. The point locations

are chosen to represent the face centers of the tightest fitting oriented bounding box around the geometry. For a capsule shape associated with body k the local positions with respect to the body frame are constants which are found by using the bind pose $\mathcal{P}^b$ and the capsule parameters from section 4.1,

$$
\begin{aligned}
\overline{\mathbf{v}}_k^{\text{cap}} &= \begin{cases} \overline{\mathbf{u}}_k^{\text{cap}} & \text{if } \|\overline{\mathbf{u}}_k^{\text{cap}}\| > 0 \\ \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} & \text{otherwise} \end{cases} \\
\hat{\overline{\mathbf{v}}}_k^{\text{cap}\perp} \cdot \hat{\overline{\mathbf{v}}}_k^{\text{cap}} &= 0,\ \overline{\mathbf{v}}_k^{\text{cap}\perp} \neq \mathbf{0} \\
\boldsymbol{v}_k^1 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} + (\|\overline{\mathbf{u}}_k^{\text{cap}}\| + r_k^{\text{cap}}) \hat{\overline{\mathbf{v}}}_k^{\text{cap}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^2 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} - (\|\overline{\mathbf{u}}_k^{\text{cap}}\| + r_k^{\text{cap}}) \hat{\overline{\mathbf{v}}}_k^{\text{cap}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^3 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} + r_k^{\text{cap}} \hat{\overline{\mathbf{v}}}_k^{\text{cap}\perp} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^4 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} - r_k^{\text{cap}} \hat{\overline{\mathbf{v}}}_k^{\text{cap}\perp} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^5 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} + r_k^{\text{cap}} \hat{\overline{\mathbf{v}}}_k^{\text{cap}\perp} \times \hat{\overline{\mathbf{v}}}_k^{\text{cap}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^5 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{cap}} - r_k^{\text{cap}} \hat{\overline{\mathbf{v}}}_k^{\text{cap}\perp} \times \hat{\overline{\mathbf{v}}}_k^{\text{cap}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right).
\end{aligned}
$$

If the collision geometry is a box then instead the constants are found the following way using the box parameters,

$$
\begin{aligned}
\boldsymbol{v}_k^1 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} + e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{x_k^{\text{box}}}{2} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^2 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} - e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{x_k^{\text{box}}}{2} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^3 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} + e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{y_k^{\text{box}}}{2} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^4 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} - e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{y_k^{\text{box}}}{2} \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^5 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} + e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{z_k^{\text{box}}}{2} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right) \\
\boldsymbol{v}_k^6 &= (\overline{\mathbf{q}}_{\mathcal{B}_k}^b)^{-1} \circ \left(\left(\overline{\mathbf{p}}_k^{\text{box}} - e^{\frac{1}{2}\tilde{\overline{\mathbf{u}}}_k^{\text{box}}} \circ \frac{z_k^{\text{box}}}{2} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{\mathsf{T}} \right) - \overline{\mathbf{p}}_{\mathcal{B}_k}^b \right).
\end{aligned}
$$

The r_{pos} term rewards minimization of the local horizontal position errors, global vertical position errors, and global orientation errors of all bodies between the kinematic reference and simulated characters,

$$r_{\text{pos}} = \exp\left(\frac{-10}{N}\sum_{i=1}^{6}\sum_{k=1}^{N}\left\|(\overline{\mathbf{p}}_{\mathcal{B}_k}^{\text{MM}} + \overline{\mathbf{q}}_{\mathcal{B}_k}^{\text{MM}} \circ \boldsymbol{v}_k^i - \overline{\mathbf{p}}_{\text{CRF}}^{\text{MM}}) - (\overline{\mathbf{p}}_{\mathcal{B}_k}^{\text{sim}} + \overline{\mathbf{q}}_{\mathcal{B}_k}^{\text{sim}} \circ \boldsymbol{v}_k^i - \overline{\mathbf{p}}_{\text{CRF}}^{\text{sim}})\right\|\right).$$

Positional alignment rewards are horizontally translation invariant to allow drift between the characters without penalty. This was found to be necessary to prevent the accumulation of small positional errors from being considered a catastrophic failure, yielding more robust policies than global tracking where drift is corrected at all costs even if motion quality must suffer. Because orientations and vertical positions are globally tracked, the overall style and user controlability of the motion is still preserved.

The r_{vel} term rewards minimization of the global velocity errors and global angular velocity errors of all bodies between the kinematic reference and simulated characters,

$$r_{\text{vel}} = \exp\left(\frac{-1}{N}\sum_{i=1}^{6}\sum_{k=1}^{N}\left\|(\dot{\overline{\mathbf{p}}}_{\mathcal{B}_k}^{\text{MM}} + \overline{\boldsymbol{\omega}}_{\mathcal{B}_k}^{\text{MM}} \times \boldsymbol{v}_k^i) - (\dot{\overline{\mathbf{p}}}_{\mathcal{B}_k}^{\text{sim}} + \overline{\boldsymbol{\omega}}_{\mathcal{B}_k}^{\text{sim}} \times \boldsymbol{v}_k^i)\right\|\right).$$

The r_{local} term encourages the local orientations of each non-root body with respect to its parent to be the same between the the kinematic reference and simulated characters. In other words, this term rewards the poses for being similar while ignoring any rigid transformations,

$$r_{\text{local}} = \exp\left(\frac{-10}{N}\sum_{k=2}^{N} 2\left\|\widetilde{\log}\left((\mathbf{q}_k^{\text{MM}})^{-1}\mathbf{q}_k^{\text{sim}}\right)\right\|\right).$$

The $r_{\text{cm-vel}}$ term rewards minimization of the global center of mass velocity error between the kinematic reference and simulated characters,

$$r_{\text{local}} = \exp\left(-\left\|\dot{\overline{\mathbf{p}}}_{cm}^{\text{MM}} - \dot{\overline{\mathbf{p}}}_{cm}^{\text{sim}}\right\|\right).$$

The r_{fall} term is a little different than the others. This term multiplies all the other terms, modulating the total reward which can be achieved. The value of this term depends on the distance between the heads (index k_{he}) of the kinematic reference and simulated characters in a complex way, taking on a value of 1 when the heads are within a threshold distance (≈ 0.2 meters) of each other and rapidly decreasing to zero when they pass this threshold,

$$\text{clamp}\left(1.3 - 1.4\left\|(\overline{\mathbf{p}}_{\mathcal{B}_{k_{\text{he}}}}^{\text{MM}} - \overline{\mathbf{p}}_{\text{CRF}}^{\text{MM}}) - (\overline{\mathbf{p}}_{\mathcal{B}_{k_{\text{he}}}}^{\text{sim}} - \overline{\mathbf{p}}_{\text{CRF}}^{\text{sim}})\right\|, 0, 1\right).$$

The purpose of having this type of modulation is to severely discourage fallen states. During training the character will fall over many times and the states on the ground are usually unrecoverable unless extreme behaviours are learned. In most cases these extreme behaviours are not discovered and the policy instead optimizes for some locally optimal but entirely useless behaviour. The r_{fall} modulation makes it so that all fallen states outside the threshold head distance have a reward quickly approaching zero. This means that there is no locally optimal useless behaviour, the only way to get non-zero rewards is to return to a pose which is within the threshold head distance. Within the threshold distance all the standard reward terms have a relative weighting to encourage tracking.

The relative weighting of reward terms was based purely on trial and error experimentation. Weightings were not found to be particularly sensitive to small variations. Most of the reward

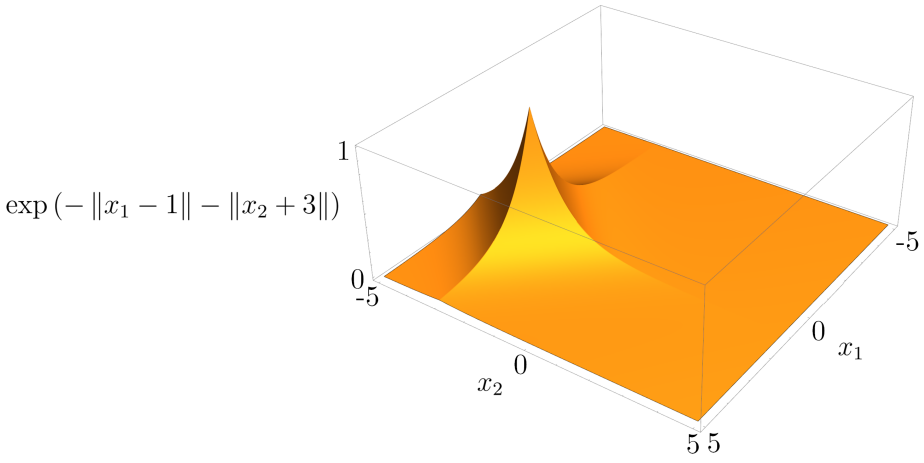


Figure 5.8: Visualization of a simplified reward landscape for two body tracking

terms are structured to be the exponential function of a sum of negative weighted error norm terms. The reason for this is clarified if a simplified example is used. Consider a reward function of the form,

$$\exp\left(-\|x_1 - 1\| - \|x_2 + 3\|\right).$$

The associated rewards for various x_1 and x_2 are plotted in Figure 5.8. The reward clearly takes on a maximum value where $\|x_1 - 1\| = 0$ and $\|x_2 + 3\| = 0$. This means maximum reward is achieved only when all negative weighted error norms being summed are minimized. The same logic applies for the terms forming r, the maximum reward will be achieved only when all errors in the sum are simultaneously minimized, that is only when all bodies are tracking well.

Using the properties of exponents and the fact norms can never take on a negative value this can be shown explicitly as an inequality,

$$\begin{aligned}
\exp\left(-\sum_{k=1}^{N}\|\delta\mathbf{x}_k\|\right) &= \prod_{k=1}^{N}\exp\left(-\|\delta\mathbf{x}_k\|\right) \\
\exp(-\|\delta\mathbf{x}_k\|) &\leq 1 \quad \forall\, k \in \{1,\ldots,N\} \\
\prod_{k\in\{1,\ldots,N\}-\{k_{\text{worst}}\}}\exp\left(-\|\delta\mathbf{x}_k\|\right) &\leq 1 \\
\exp\left(-\|\delta\mathbf{x}_{k_{\text{worst}}}\|\right)\prod_{k\in\{1,\ldots,N\}-\{k_{\text{worst}}\}}\exp\left(-\|\delta\mathbf{x}_k\|\right) &\leq \exp\left(-\|\delta\mathbf{x}_{k_{\text{worst}}}\|\right) \\
\prod_{k=1}^{N}\exp\left(-\|\delta\mathbf{x}_k\|\right) &\leq \exp\left(-\|\delta\mathbf{x}_{k_{\text{worst}}}\|\right) \\
\exp\left(-\sum_{k=1}^{N}\|\delta\mathbf{x}_k\|\right) &\leq \exp\left(-\max\left\{\|\delta\mathbf{x}_1\|,\ldots,\|\delta\mathbf{x}_N\|\right\}\right).
\end{aligned}$$

With this form the body with the largest error term sets an upper limit on the maximum attainable reward. even if tracking is perfect for every body except one $\|\delta\mathbf{x}_k\| = 0, k \in \{1,\ldots,N\} - \{k_{\text{worst}}\}$, the worst tracked body with index k_{worst} will limit the reward which can be achieved to $\exp\left(-\|\delta\mathbf{x}_{k_{\text{worst}}}\|\right)$. A reward formed in this way weighs tracking improvements on the bodies with the worst errors more significantly. This discourages difficult to optimize bodies from being ignored and leads to tracking which is overall more visually similar to the kinematic reference motion.

No reward term is associated with the control effort since a maximum allowable torque of 200 Nm is set for the joint actuators. This was found to be necessary because rewards discouraging control effort had a tendency to cause training to generate policies which exert no control effort whatsoever unless carefully tuned. Torque limits ensure the policy can use as much control effort as the actuators allow, enabling bang-bang control behaviours to be learned if they are optimal. One typical purpose of penalizing actuator effort is to prevent high frequency behaviours. The filtering on the policy output and the PD control gains on the actuators allow the step response characteristics to be constrained as desired, meaning high frequency behaviours can still be prevented.

5.3 Training Control Policies

Policy training is carried out by using PPO as the reinforcement learning algorithm. OpenAI's "baselines" project [84] provides a well tested implementations of PPO. The distributed CPU version of the algorithm from "baselines" running 8 workers in parallel was used to achieve an average of 2500 environment steps per second on a single machine. Training is broken down into various relevant stages,

1. Initialization
2. Trajectory Sampling
3. Episode Resets
4. PPO.

The training follows a specific procedure. First everything is initialized in a manner which will hopefully help the optimization converge to the best policy possible. Next trajectories are sampled as training data for the reinforcement learning algorithm. When adequate samples have been gathered PPO is applied to the set of sampled trajectories to obtain an improved policy and value function estimate. After an iteration new trajectories are sampled using the improved policy. The process of sampling and PPO iterations is continued until the policy performance has been observed to have reached a performance plateau where no significant further improvement seems achievable. The average reward across all samples in an iteration gives a good indicator of overall policy performance.

5.3.1 Initialization

Initialization is performed only once at the very start of training. In this stage the neural networks and optimizer are initialized. All the weights associated with a particular non input layer ℓ of a neural network are contained in a weight matrix $\mathbf{W}_\ell$. The layer is considered to have a width of $m_\ell \in \mathbb{Z}^+$. Weight matrices for each layer are initialized as follows,

$$\begin{aligned} \underline{\mathbf{W}}_\ell &\rightsquigarrow \mathcal{N}(\mathbf{0}, \mathbf{1}) \\ \mathbf{W}_\ell &\leftarrow \frac{\underline{\mathbf{W}}_\ell}{\sqrt{\underline{\mathbf{W}}_\ell^\mathsf{T} \underline{\mathbf{W}}_\ell}}. \end{aligned}$$

Biases for the various units in each layer are initialized to zero $\mathbf{B}_\ell \leftarrow \mathbf{0}$. Because the actions are designed such that a value of zero will perform the open-loop control derived from the kinematic reference, it is favorable for the policy network to output very small values when training begins.

To initialize the policy this way the output layer weights $\underline{\mathbf{W}}_{\ell_o}$ of the neural network used by the policy are initialized differently. Each weight value is sampled from a uniform distribution ranging from -0.01 to 0.01,

$$\mathbf{W}_{\ell_o} \leftsquigarrow \text{multivariateUniformDistribution}(-0.01, 0.01).$$

The advantage of this final layer initialization strategy is that it puts the initial policy closer to a favorable local optima. This was validated by comparing the performance of policies initialized with the strategy to those initialized with a default weight initialization strategy. This comparison is given in Figure 5.9.

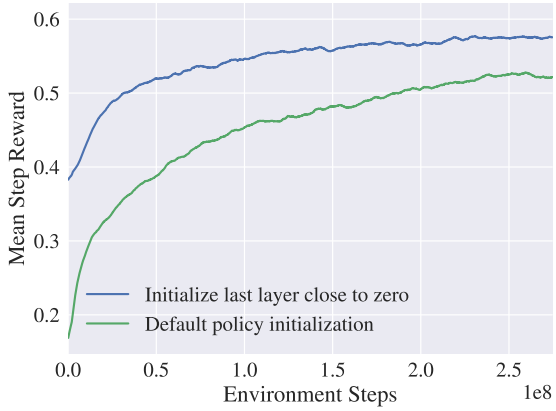


Figure 5.9: Effect of initializing the weights of the final policy network layer to smaller magnitude values.

The policy also has associated non neural network related parameters. The covariance matrix of the action distribution is set to always be diagonal $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \mathbf{1}\boldsymbol{\sigma}_{\boldsymbol{\theta}}$. The variances are controlled by $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$ and these are all initialized such that $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \mathbf{1}$. The parameters for the policy network and the value function estimate network are then initialized as follows,

$$\boldsymbol{\theta} \leftarrow \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_{\ell_o} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{\ell_o} \\ \boldsymbol{\sigma}_{\boldsymbol{\theta}} \end{bmatrix}, \qquad \boldsymbol{\phi}_{\boldsymbol{\theta}} \leftarrow \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_{\ell_o} \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{\ell_o} \end{bmatrix}.$$

5.3.2 Trajectory Sampling

The human gamepad user, kinematic character controller, physical simulation of a character, and control policy together form the agent and environment. This system can still be considered modeled by a markov decision process even though the complexity may have increased and the state and actions are part of continuous spaces rather than discrete sets. The number of possible states and potential actions associated with each state are now enormous so the associated graph is difficult to conceptualize. Nonetheless sampling an action from the policy and ticking the system forward in time is equivalent to making a decision which results in a transition to another state with an associated reward. Sampling actions as time progresses and recording the results of each action still allows sampling of trajectories from the MDP.

In this work the division between agent and environment is somewhat abstract. The "agent" consists only of the act of sampling from the policy, and the environment is everything else. The human gamepad user, kinematic reference motion being controlled by them, the underlying simulation, and the details of the action to actuation transformation, all these form the environment. This leads to a particularly pressing problem: how can sampling millions of times from the MDP be automated for training if a human's actions are part of the underlying MDP? A change of perspective is needed. The human user is fundamentally *unpredictable*, their actions can change at a whim and their goals are completely external factors which are unknown. It is not necessary to actually have a human user for training purposes so long as the distribution of behaviours a human user might have is appropriately sampled. It should however be mentioned that the statement a human user is unpredictable does not imply they act completely randomly, just that their intentions are unknown. Humans have definite constraints on their behaviour which need to be considered. As an example, a human will generally not toggle a button hundreds of times a second, this is unrealistic given that they have to manipulate a gamepad using their bodies. A human user will behave unpredictably moment to moment but their behaviours still fit within a predictable distribution, for example steering a character around they will habitually generate inputs which result in straight line motion, turning motions to orient toward a location, rapid direction changes when they make a mistake, and transitions between styles will generally be infrequent.

To automate training an artificial human user is required. The artificial user generates gamepad inputs which are then used by the kinematic character controller and the policy. The artificial inputs should be representative of inputs a human would make. It should sample both worst case scenarios which make control difficult and more typical trajectories that a human would perform when trying to move a character around.

User behaviour modifying events were designed to act as an artificial user during training. The artificial user is described by Algorithm 5.14 which sets the value of $\mathbf{g}_{\text{l-stick}}$, $\mathbf{g}_{\text{r-stick}}$, g_{trigger},

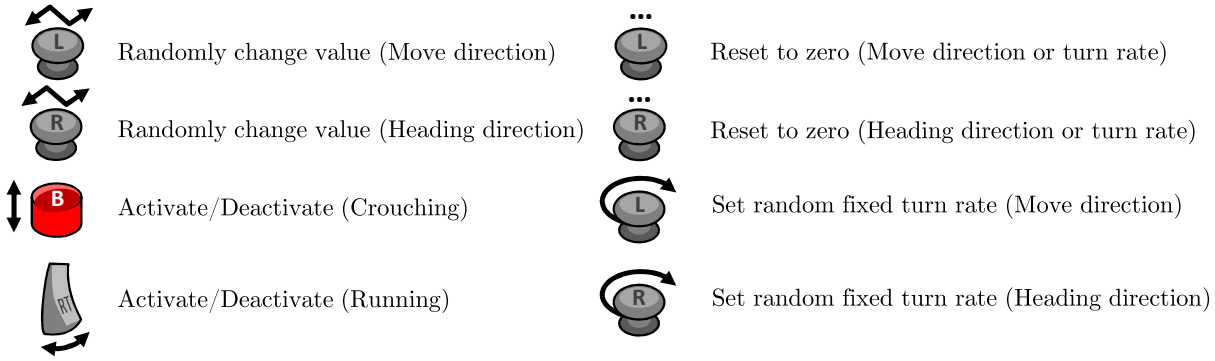


Figure 5.10: Control behaviour change events which are possible for the artificial user.

g_{button} each time-step during training. Every time-step there is a probability of user behaviour modifying events occurring. The behaviour modifications which can occur are summarized in Figure 5.10. These are randomly changing the left and right stick direction held, toggling the face button and trigger states between being held on or off, setting the left and right sticks to their neutral position, setting a fixed turning rate for the left ($\omega_{\text{l-stick}}$) and right ($\omega_{\text{r-stick}}$) sticks of up to 1 rotation per second, and setting the turning rate for the left and right sticks to zero. Each time-step the the turning rates are multiplied with the step duration Δt to rotate the associated sticks by the proper amount. This models the behaviour of a user steering a character in various curves. Each control behaviour modifying event has a probability of occurring in any given time-step of 0.001. This low probability allows both behaviours with reasonably rapid user input changes as well as with infrequent input changes to be sampled. The probability of the events occurring each time-step has a significant effect. Selecting a probability which is too high results in a behavioural distribution dominated by rapid changes which make no long term progress, and setting it too low results in a behavioural distribution dominated by one or very few behaviours that will not be robust to successive unexpected user input changes.

5.3.3 Episode Resets

At the beginning of a trajectory the agent and environment must be initialized, the states the system can start in define the starting state distribution. During the trajectory it is possible that a termination condition is triggered where the trajectory is considered to end. The termination of the current trajectory and re-initialization of a new trajectory together form the concept of an episode reset. The resets are a fundamental aspect of the training procedure since they have a strong influence on the distribution of states which are sampled. With deep reinforcement learning, performance is only guaranteed to be optimized with respect to states in the sampling distribution. The starting state distribution and termination conditions are one of the primary

Algorithm 5.14 artificial user

$\mathbf{g}_{\text{l-stick}} \leftarrow \mathbf{0}$
$\mathbf{g}_{\text{r-stick}} \leftarrow \mathbf{0}$
$g_{\text{trigger}} \leftarrow 0$
$g_{\text{button}} \leftarrow 0$
$\omega_{\text{l-stick}} \leftarrow 0$
$\omega_{\text{r-stick}} \leftarrow 0$
for each timestep do
    with probability (0.001) **do**
        $\phi \leftsquigarrow \text{unif}(0, 2\pi)$ radians
        $\mathbf{g}_{\text{l-stick}} \leftarrow \begin{bmatrix}\cos\phi & \sin\phi\end{bmatrix}^{\mathsf{T}}$
    with probability (0.001) **do**
        $\phi \leftsquigarrow \text{unif}(0, 2\pi)$ radians
        $\mathbf{g}_{\text{r-stick}} \leftarrow \begin{bmatrix}\cos\phi & \sin\phi\end{bmatrix}^{\mathsf{T}}$
    with probability (0.001) **do**
        $\omega_{\text{l-stick}} \leftsquigarrow \text{unif}(-2\pi, 2\pi)$ radians/second
    with probability (0.001) **do**
        $\omega_{\text{r-stick}} \leftsquigarrow \text{unif}(-2\pi, 2\pi)$ radians/second
    with probability (0.001) **do**
        $g_{\text{trigger}} \leftarrow 1 - g_{\text{trigger}}$
    with probability (0.001) **do**
        $g_{\text{button}} \leftarrow 1 - g_{\text{button}}$
    with probability (0.001) **do**
        $\mathbf{g}_{\text{l-stick}} \leftarrow \mathbf{0}$
    with probability (0.001) **do**
        $\mathbf{g}_{\text{r-stick}} \leftarrow \mathbf{0}$
    with probability (0.001) **do**
        $\omega_{\text{l-stick}} \leftarrow 0$
    with probability (0.001) **do**
        $\omega_{\text{r-stick}} \leftarrow 0$
    end
    $\mathbf{g}_{\text{l-stick}} \leftarrow \begin{bmatrix}\cos(\omega_{\text{l-stick}}\Delta t) & -\sin(\omega_{\text{l-stick}}\Delta t) \\ \sin(\omega_{\text{l-stick}}\Delta t) & \cos(\omega_{\text{l-stick}}\Delta t)\end{bmatrix} \mathbf{g}_{\text{l-stick}}$
    $\mathbf{g}_{\text{r-stick}} \leftarrow \begin{bmatrix}\cos(\omega_{\text{r-stick}}\Delta t) & -\sin(\omega_{\text{r-stick}}\Delta t) \\ \sin(\omega_{\text{r-stick}}\Delta t) & \cos(\omega_{\text{r-stick}}\Delta t)\end{bmatrix} \mathbf{g}_{\text{r-stick}}$
end for

means to manipulate which states policy improvement focuses on. Consequently these should be chosen in a way that will constrain sampling to regions that are useful.

Properly designed resets can be viewed as a means to do importance sampling, enabling training to converge with a tractable number of samples. Training is a time consuming process and it is not feasible to exhaustively sample every possible state due to the large number of dimensions and continuous state space. Only states that allow the control objective to be achieved and the policy to be optimized should be sampled in significant quantity. It is wasteful to sample states that will not be reached in typical operation, for example states where the character is upside down, fallen over, flying through the air, etc. It is impossible for the character to perform locomotion in such scenarios. The controller is focused on achieving locomotion behaviours, and so sampling should be focused only on cases where locomotion is achievable or has been disturbed but is still able to recover relatively easily.

The re-initialization scheme using in this work is particularly simple. The kinematic character controller and artificial user are never reset, producing a continuous stream of generated animation. Each time-step the updated pose and pose-velocity of the kinematic reference motion are available in $\mathcal{P}^{\text{MM}}$ and $\mathcal{V}^{\text{MM}}$. When a reset is triggered all forces on the simulated character bodies are cleared and the bodies are set in a state such that pose and pose-velocity are equivalent to those of the kinematic character. This means the following holds after a reset,

$$\mathcal{P}^{\text{sim}} = \mathcal{P}^{\text{MM}}$$
$$\mathcal{V}^{\text{sim}} = \mathcal{V}^{\text{MM}}.$$

This ensures that the initial simulated character states are close to the actual distribution of kinematic character states that can be generated. More importantly this also initializes the simulated character in the highest possible value states. States with the *global maximum* reward will always be sampled during training. This is a significant help during training since the value function estimate will quickly be optimized to accurately represent the highest possible value states, and the policy can then quickly discover which actions take the system towards these high value states.

Various termination conditions are specified. The first is simply a condition which is triggered to end a trajectory when 40000 samples have been gathered since the previous PPO iteration. The next iteration of PPO is then triggered. Termination is also triggered if a trajectory is longer than 20 seconds. To prevent this from biasing training the final reward is set such that it is equal to the value function estimate evaluated at the terminal state. This approximates the sequence of rewards which would have been gathered had the trajectory run its full course, ensuring that the value of samples are not negatively impacted due to the arbitrary termination event unrelated to the quality of the actions taken by the policy.

The second termination condition is meant to prevent the gathering of samples in unrecoverable states. This is somewhat ill posed as it is impossible to determine which states are unrecoverable when the policy is attempting to discover strategies to recover in the first place. For this reason the best that can be done is to define a heuristic which roughly classifies if a state is unrecoverable. The heuristic which was found to work well in this work was to check the distance between the heads of the simulated and kinematic reference characters. The reward term r_{fall} evaluated using the current state is explicitly used to perform this check. If $r_{\text{fall}} = 0$ the characters are considered to have diverged in such a way that a reset should be triggered. It is assumed that once this termination criteria has been met the character would not be likely to exit states which yield $r_{\text{fall}} = 0$ and hence the terminal reward is zero since the return from these dead-ends is zero.

5.4 Results

After training a set of optimal policy parameters $\boldsymbol{\theta}^*$ are obtained. The quality of the policies which result from training are difficult to assess if only the rewards are observed. The training of an optimal policy does not guarantee qualitative objectives desired for a useful system are met. The system also operates somewhat differently in training compared to in a user facing runtime application. During training the artificial user acts erratically and actions are sampled from a distribution in order to explore and discover which are optimal. In a user facing runtime application a human user steers the character around and the best known action in a given state is chosen deterministically by always choosing an action which is the mean of the optimal policy's distribution in a given state. Training only optimizes behaviour with respect to an approximation of the user facing runtime application. To properly evaluate the results simple performance studies were designed. These measure or demonstrate the capabilities of the final physically-based character controller in application oriented scenarios.

5.4.1 Robustness Analysis

An impact testing experiment was designed to measure the robustness of the controller to external perturbations. In order to measure the effect of disturbances a cube shaped rigid-body was launched every second at the simulated character as it was being controlled. The cube is launched from a uniformly random sampled position on a hemisphere with a diameter of 4 m centered at the simulated character's center of mass, see Figure 5.12a for a visualization of this setup. Upon launching the cube is given a random orientation and a fixed velocity of 5 m/s towards a uniformly random sampled target location between $\overline{\mathbf{p}}^{\text{sim}}_{\text{CM}} - 0.5[0\ 0\ 1]^{\mathsf{T}}$m and $\overline{\mathbf{p}}^{\text{sim}}_{\text{CM}} + 0.5[0\ 0\ 1]^{\mathsf{T}}$m. This causes the cube to frequently impact the character in various ways, with

the maximum possible impact energy increasing as the mass of the cube is increased. The cube remains in the scene at the position it settles in until it is launched again, meaning there is a potential for the character to run and trip over it or kick it between launches. During the impact experiment the character is automatically controlled to run at 3 m/s without strafing and changes direction every 4 seconds. A trained policy with deterministic actions is used.

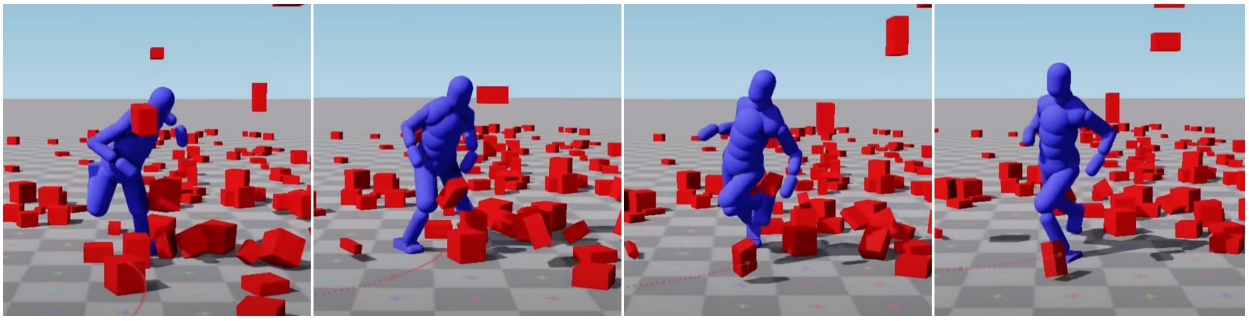
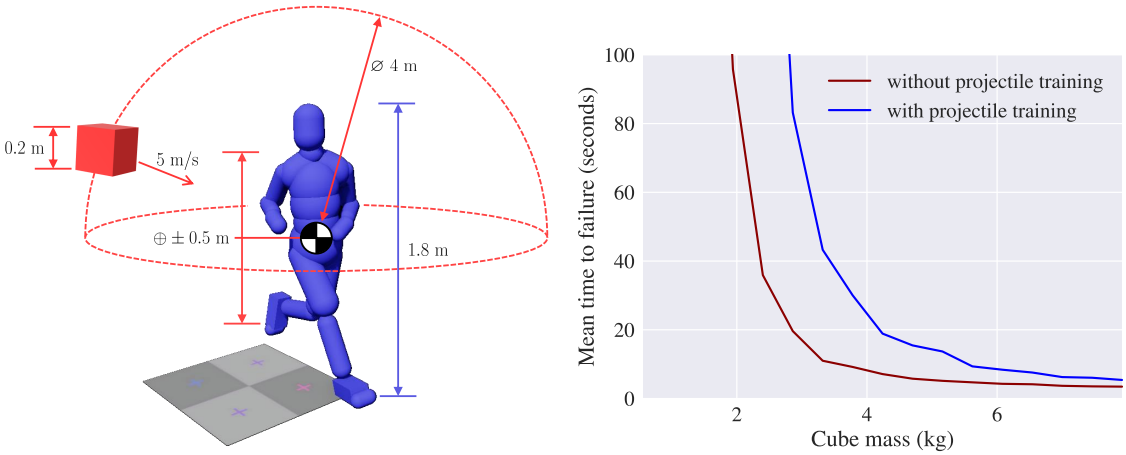


Figure 5.11: The trained policy is quite robust, recovering even when being pelted by multiple objects and running over rubble covered terrain. This sequence of snapshots are ordered left to right. Objects in mid air are flying toward the character.



(a) Measurements related to the impact testing experiment.

(b) The average time between falls for impact testing with a cube of various masses.

Figure 5.12: Impact testing setup and results

Impact testing data has been compiled for incrementally increasing cube masses between 0.01 kg and 8 kg. At each cube mass increment 1 million time-steps are computed. The average time between resets is recorded measuring how frequently the character falls over from the impact test at the specified cube mass. This data is visualized in Figure 5.12b. An additional line is plotted for a policy which has been trained in an environment with cube impacts of mass varying between

Table 5.1: Comparison of responsiveness to other work. The estimated time required to make a 90° turn is compared. **values from cited papers are estimated from similar behaviour in published videos and are inexact.*

Method	**Rise Time**
(Ours)	1.2s
Peng et al.[55] - *Multi-Clip Reward*	*2.0s
Liu and Hodgins[54] - *Running Skill Graph*	*5.1s

0.01 kg and 8 kg. The robustness of the policy with respect to impacts can be improved by doing this, but it causes some loss of tracking quality since a more statically stable style is learned.

Even the policy trained without impact training has good performance. Figure 5.11 shows a sequence of snapshots of a simulated character in an environment with many objects constantly being launched toward it as it runs over loose rubble like terrain. The character is significantly perturbed by an impact to the head in the leftmost frame yet manages to recover balance as shown in the rightmost frame even while contending with the many objects in the way. Keep in mind the character has no awareness of these objects but can nonetheless maintain control and balance.

5.4.2 Responsiveness Analysis

Human users are sensitive to delays in control [85]. Delays make control difficult since a user cannot immediately correct control errors and so they must instead constantly plan ahead. The responsiveness of a trained physical character control system has been measured to quantify the delays inherent in the system. The responsiveness of the method developed here is roughly compared to the results demonstrated by two similar previous works. Both Peng et al. [55] and Liu and Hodgins [54] have published video results of their methods being used to steer a character's motion with abrupt motion direction and heading changes. In order to estimate the time required to achieve a 90° turn, associated videos have been examined to find moments where steering changes occur abruptly. This was also measured for our method. A comparison of the rise time is given in Table 5.1. Our method is shown to achieve a level of responsiveness which is favorable.

Responsiveness is not only dependent on the performance of the policy in tracking the reference motion, but also on the responsiveness of the the kinematic character controller to an abrupt control change. The kinematic character controller necessarily takes some time to achieve a change in direction because real human beings take time to change direction. Because of this measuring response time purely with respect to the change in user input does not clearly demonstrate if responsiveness can be further improved. A straightforward method was devised to compare the responsiveness of the simulated character controller to the responsiveness of the

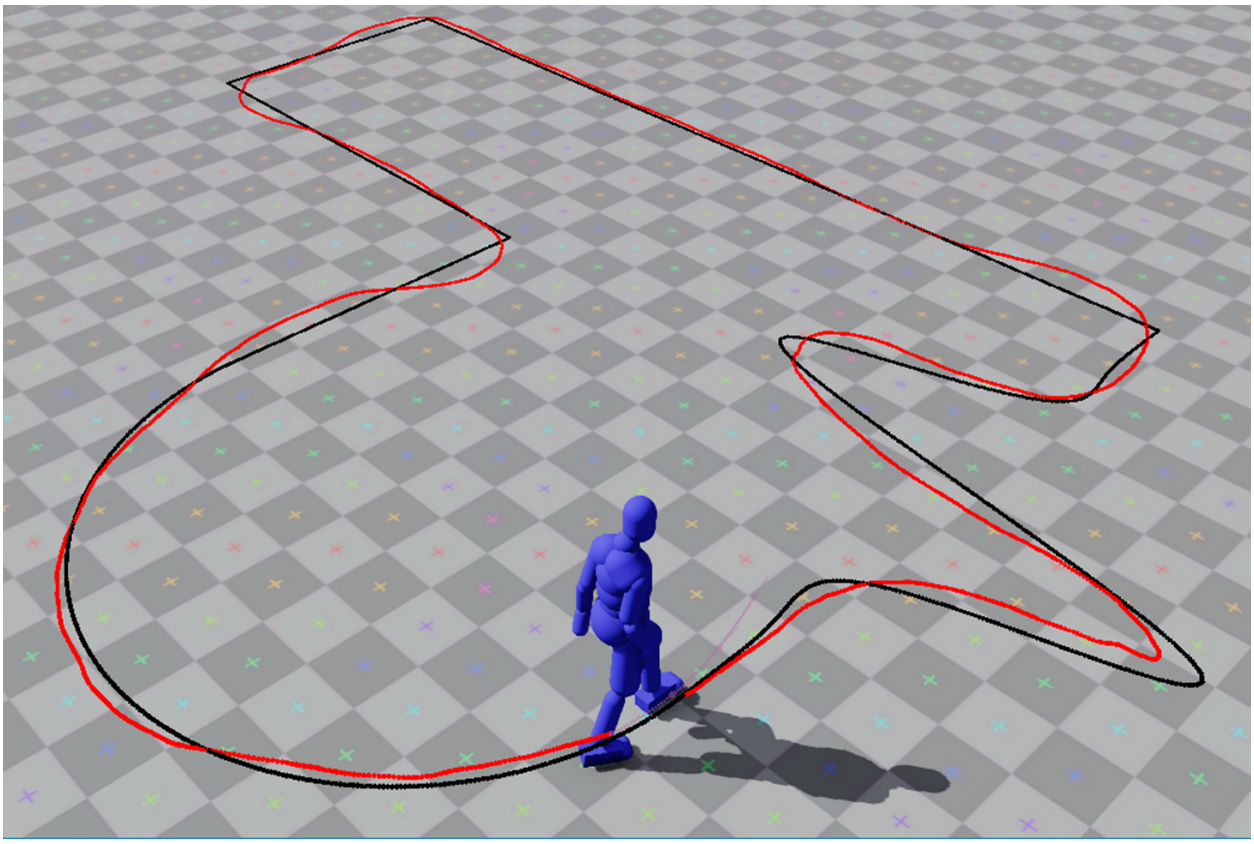


Figure 5.13: A human user was instructed to steer their character at walking pace to follow a reference path (black line) as closely as possible. The path taken by the user controlled character's projected center of mass is visualized (red line), indicating that the task can be completed with a high degree of accuracy. Ground tiles are 0.5 m $\times$ 0.5 m.

kinematic character controller. The step response for straight line motion involving a running character turning around to run the opposite direction is plotted. The plot is shown in Figure 5.14 and clearly shows that the response time of the kinematic character is only negligibly faster than the response time for the simulated character. The limiting factor for responsiveness seems to be the responsiveness of the generated animation. Responsiveness can only be significantly improved by using a dataset with a more agile style of motion or by requesting more unrealistic motion.

The steerability of a physically simulated character was measured by making a human user control the simulated character with a standard gamepad. The user was instructed to steer a walking character to follow a reference path on the ground as closely as possible. The path was constructed to include various features such as a sinusoidal curve, straight lines, sharp 90° turns, and a gradual turn. The path was visible to the user as a dark black line on the floor. The character center of mass location projected on the ground was used to draw a red line indicating the path taken by the user controlled character.

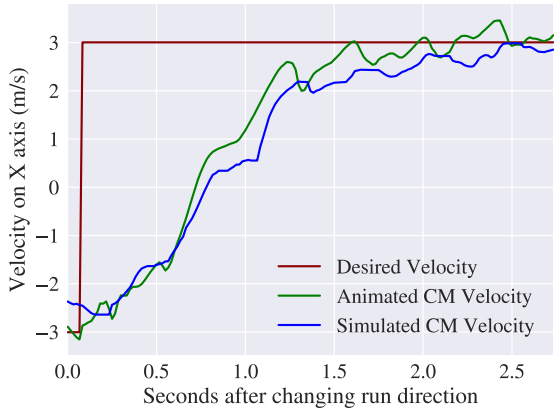


Figure 5.14: Center of mass velocities for a simulated and kinematic character requested to change from running -3 m/s to 3 m/s.

A result of this experiment is shown in Figure 5.13. A human user is capable of staying on course with a high degree of accuracy, never straying more than 0.5 m from the path. The largest tracking errors are seen where rapid changes in direction are required, such as at corners. This is a result of the character having inertia but is also attributed to the non-zero control delay which make abrupt steering changes a bit more difficult.

5.4.3 Self Collision Analysis

In most experiments self collision between bodies composing the character have been disabled. This speeds up training and improves the ability to track the reference motion since blends may sometimes generate animations with intersections. In most cases the lack of self collision is not noticeable because self collisions do not frequently occur during locomotion. Turning off all self collision helps because the collision geometry of the character is designed to accurately represent the surface of the character and mass properties through intersections. No effort was made to ensure realistic clearances exist between body parts. Nonetheless, it is possible to support self collisions between certain bodies. No body should allow collisions with directly attached bodies that are likely to always be intersecting, but for example the legs can be made to collide with each other and the upper torso, the arms can be made to collide with each other and the lower torso, and so forth. When enabling these self collisions care should be taken to set the friction between body parts as low as reasonably possible to minimize binding when glancing collisions occur.

Successful policies can be trained with self collision enabled, however this has negative consequences. Because self collisions can lead to falls, the policy learns strategies which minimize intersections as much as possible. This has the effect of altering the locomotion style in ways which widen the stance to move the legs further apart, as shown in Figure 5.15, ensuring fewer collisions occur. This tends to look a little unnatural, so if self collisions are desired it is rec-

ommend to use a second set of smaller collision geometries which offer more clearance, likely reducing the possibility of stance widening.

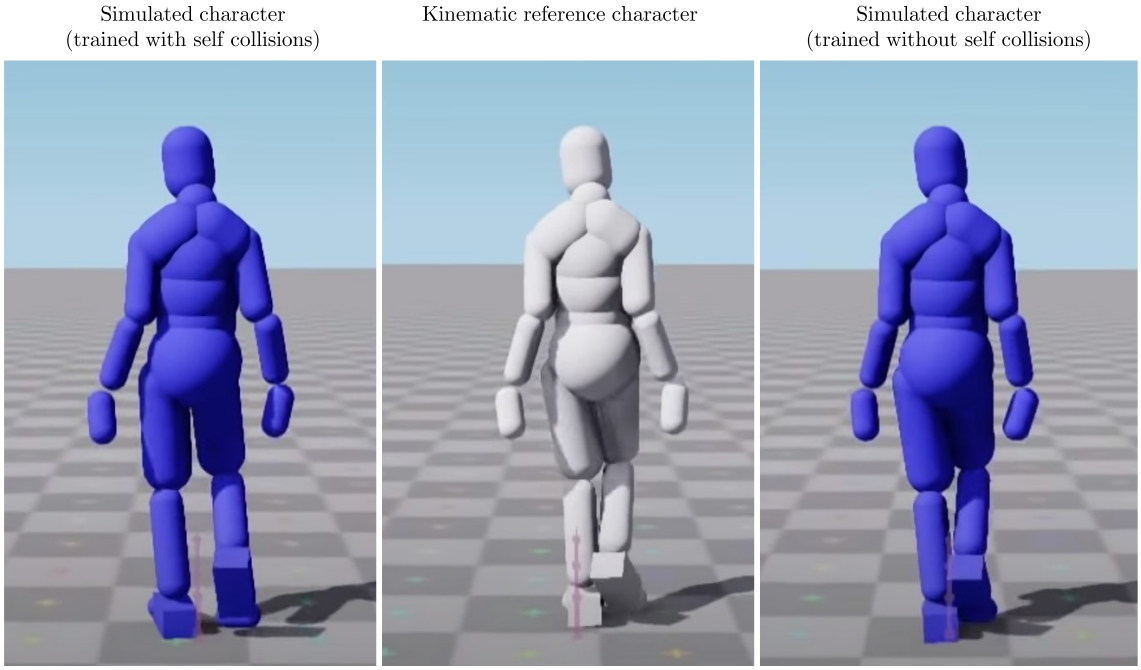


Figure 5.15: Training with self collisions causes the resulting policy to widen the character stance compared to the reference motion. This reduces the incidence of self collisions.

5.4.4 Ablation Studies

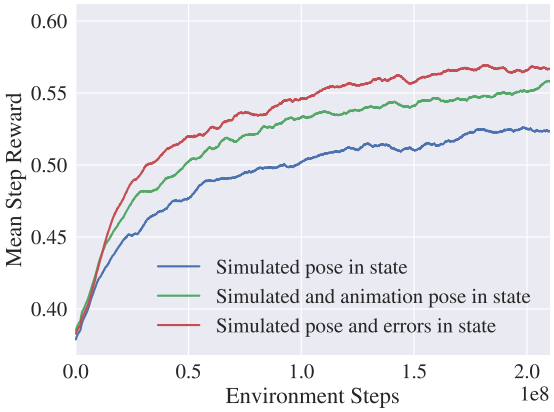
The systems described in this work are complex and unpredictable. The design of the state representation, action representation, and rewards required many iterations. This necessitated using intuition and previous work to decide what design choices were worth investigating. However once design choices began to yield useful results design iterations were performed to decide on what worked best in terms of producing the best performing policies and meeting the objectives of the project. The final design of the state, action, and reward are a result of countless design iterations, weeks of experiments, and a large number of disappointing failures.

The state, action, and reward are structured in ways which might seem quite arbitrary, so ablation experiments were performed to help build confidence in their design. These involved both quantitative comparisons of the average reward values obtained by these policies during training, as well as qualitative comparisons of the resulting policies in the user facing runtime application.

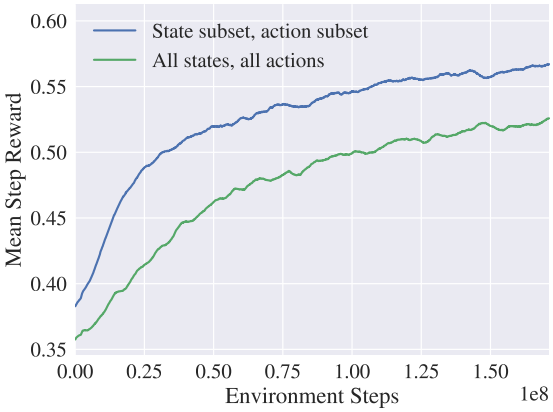
In order to perform feedback based tracking the state given to the policy must contain information about the physical character as well as the kinematic character being tracked. Policies without kinematic character information may still function using open-loop control. An ablation study was designed to determine the effects of omitting information about the kinematic character from the state, and to compare other parameterizations of the state in order to determine which worked best. One parameterization tested is of course that which was presented in section 5.2.1, while others omit the "error" terms in the state $(\mathbf{s}_{\text{sim-cm-vel}} - \mathbf{s}_{\text{kin-cm-vel}})$ and $(\mathbf{s}_{\text{sim}} - \mathbf{s}_{\text{kin}})$, one instead providing the kinematic state directly through $\mathbf{s}_{\text{kin-cm-vel}}$ and $\mathbf{s}_{\text{kin}}$, and the other lacking it altogether. A comparison of policies trained to operate with each of these variations is given in Figure 5.16a. Surprisingly the policies without information about the kinematic character in the state still function, albeit with reduced performance. The state parameterization which includes the error terms outperforms the state parameterization that includes the unmodified kinematic character terms. The reason why this state design works better is difficult to deduce, but one possibility is that it makes linear feedback policies easier to learn.

The choices to only provide information about a subset of the character's bodies in the state and to make the policy's actions only capable of affecting a subset of joints were both originally made to reduce the dimension of the learning problem and speed up training. The expectation was that these choices would negatively affect the performance since less information would be available to correct errors. An ablation study was designed to measure the effect of this choice, comparing the performance of a policy trained to use the state of all bodies and output actions for all joints to a policy trained to use the subset described in the prior section. The results of this study are given in Figure 5.16b. Surprisingly, the policy with access to more information and fuller control of the character yields lower average rewards. This may be because exploration and credit assignment are much simpler with a smaller action dimension.

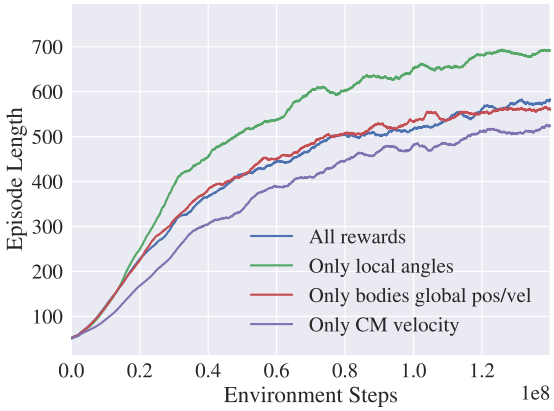
The reward is designed with multiple simultaneous goals in mind. These goals are captured in each of the terms used to construct the reward. An ablation study was designed to determine the effect of simplifying the reward by utilizing different subsets of the terms. Because the reward value itself would change, comparing these on the basis of the average reward would be somewhat meaningless. Instead a comparison in terms of the the average episode length is made to track policy improvement and ability to prevent falls. This comparison is shown in Figure 5.16c. The graph demonstrates that all rewards are capable of generating policies which learn to prevent falls, with a simple reward promoting local tracking doing the best of all. However, the difference in motion quality between the policies is clear. A policy which was only rewarded for center of mass tracking learns strange behaviours which cause it to take large strides and fling itself around. A policy which was only rewarded for local joint angle tracking moves with small steps and does not make significant progress performing locomotion. A policy which only rewards tracking



(a) Comparison of different state parameterizations.



(b) Comparison of different state parameterizations.



(c) Comparison of policies trained to optimize with different rewards.

Figure 5.16: Various ablation study results.

of its bodies' positions and velocities compromises on overall movement speed and local pose similarity in comparison to the other policies. Providing all the terms produces policies giving rise to simulated character motion most similar to the kinematic character controller output.

5.4.5 Runtime Cost Analysis

The runtime cost of the final controller was considered one of the most important design variables in this work. The runtime cost of the final system on a Intel Xeon E5-1650 V3 CPU is given in Table 5.2. Included in this breakdown are the cost of running the kinematic character controller, the cost of calculating the state, the cost of evaluating the deterministic policy using the state, and the cost of stepping the physics simulation. Costs are averaged over 1 second. The total cost of running the system is surprisingly low considering that the typical target framerate for interactive applications is 60 FPS. Only 340 μs are needed out of the $\Delta t = 16.67$ ms

Table 5.2: Average per timestep runtime cost breakdown.

Stage	**Runtime**
Kinematic Control	40 μs
State Construction	40 μs
Policy Evaluation	60 μs
Physics Simulation	200 μs
Total	**340** μs

Table 5.3: Comparison of performance affecting parameters.

Method	**Policy Network Size**	**Simulation Frequency**
(Ours)	2 hidden layers, 128 units each	60 Hz
Peng et al. [55]	2 hidden layers, 1024 & 512 units	1200 Hz
Chentanez et al. [56]	3-6 hidden layers, 512 units each	(not reported)

available each frame. That means only 2% of the computation time available is required each time-step. Additionally a video-game is likely to have been doing kinematic character animation and physics simulation already, so the added costs in that case are only for state construction and policy evaluation and add up to about 100 μs. This means in a typical scenario the addition of physically-based character with control uses *only an additional 0.6%* of the available budget. This is a negligible requirement so it can be said with confidence that the method presented here can comfortably fit within the performance budget of a real world production on current generation hardware.

Comparing to other work is difficult due to the lack of similar works with published performance statistics. Nonetheless information about the architecture and design of published methods is typically available, which allow a performance comparison to be made on the basis of the neural network sizes involved and the physics simulation frequencies. Such a comparison is shown in Table 5.3. The works chosen are selected based on the fact they are most similar to this work, and do not have significantly different control objectives. In general the method in this work functions with a much smaller network than those typically used in other works, and a physics simulation frequency which is very reasonable for interactive applications. The policy is also evaluated at a reduced rate, providing further runtime cost savings.

Chapter 6

Closing Remarks

The methods presented in this thesis achieve low runtime cost, responsive, and robust control of a physically simulated human character performing locomotion. This work improves upon previous methods of physically-based animation by allowing for control of style and motion via a continuously generated realistic trajectory driven by simple user friendly inputs. The resulting motion of the simulated character is natural looking and captures subtle variations in human movement. Target trajectories are generated by intelligently blending motion capture data in a manner which is constrained by user specified high level control goals. By generating feasible target trajectories in this way a large degree of medium timescale planning for achieving the high level goals can be handled kinematically. Target trajectories provide weakly unstable open-loop tracking through simple per-joint PD local pose tracking. Optimization in the form of reinforcement learning is used to fully stabilize trajectory tracking over the long term. Low level feedback driven adjustments to the open-loop tracking on a subset of the character's actuated joints are learned by the policy. The learning process is discouraged from generating behaviours which look unnatural since motion plans cannot be significantly modified by the controller without penalty. Feedback control frequency reduction combined with filtering prevents overly aggressive control strategies from being learned. Combining learned feedback corrections with open-loop control allows untrained policies to easily be initialized near well performing local optima. When possible design choices have been validated through comparative experiments, allowing the impact of the choices on performance to be measured.

Previous work utilizes reinforcement learning to reproduce motion capture derived behaviours on simulated characters, however the degree of high-level control over the motions and the ability to transition between them was somewhat limited. Previous methods have also not payed explicit attention to runtime cost, often having designs which utilize unacceptably high simulation frequencies or costly architectures that limit application in systems with realistic performance budgets. The method presented in this work is designed to have a very low runtime cost, requir-

ing only 340 μs total per time-step on typical hardware, with the majority of this cost coming from the physics evaluation. The low runtime cost and good quality results with the controller developed here are achieved without relying on high simulation frequencies. This has the implication that this method is well suited to real world applications, such as use in video-games. The trained controllers have good response time compared to methods in previous work and faithfully reproduce the commands of a user. Experimental results also show a high degree of robustness to perturbation. The three main research objectives, which were producing a controller that has high quality, high user directability, and low runtime cost, were all successfully met.

6.1 Future Work

Reinforcement Learning enabled the complex objectives and strict design requirements of this work to be met. Part of this success is owed to the generality of reinforcement learning, but a significant effort was also required to design and tune the system to point where results were satisfactory. The large number of hyperparameters makes optimization of the design difficult due to the sheer quantity of computations required. Although it may be difficult, improving performance through optimization driven design would be an interesting topic for future work.

This work has largely focused on bipedal locomotion tasks in simulation. Transfer of a controller trained to imitate a high-level data-driven kinematic plan to a physical robotic platform could be of great interest. The blending and sequencing of many weakly feasible reference trajectories using motion matching in kinematic space and subsequent use of reinforcement learning to correct the kinematic motions into achievable physical trajectories has been demonstrated here. This is a powerful idea which could extend beyond the locomotion tasks solved in this work. Control of characters other than humanoids has not been attempted, but it is reasonable to believe that the methods presented here should extend well to other animal or robot body shapes while they perform locomotion without major changes. Adapting the method presented here to such problems as well as control of non-walking systems or non-locomotion related tasks should be possible without major difficulty.

In this work the kinematic character controller does not generally receive constant feedback from the simulation. This was an explicit choice to prevent training instability, however there is nothing preventing a design where the policy can exert some control over the kinematic character motion. This would allow high level control goals to be specified in terms of a reward and could potentially help improve performance and robustness by giving the policy more agency over long term planning.

While the quality of the motion produced in this work is good, it can sometimes be distinguished from the motion capture reference in a side by side comparison. This is usually

when unexpected user control events like a direction change occur. These events introduce some "wobbling" which take time to get damped away. The reasons for this are difficult to pin down but could be related to modeling inaccuracies, poorly tuned parameters, sub-optimal controller performance or just other unknown factors. Work still remains in finding methods to control physically-based characters undergoing perturbations which are visually indistinguishable from motion capture. Great advances have been made in motion imitation, but methods for synthesis of believable human motions from scratch still have a long way to go. This is probably a question of learning to accurately model human movement and behaviour rather than purely imitate it, and will surely lead to more interesting research in the future. It can be said with confidence that this work is at least a "step" in the right direction.

Bibliography

[1] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes, "Drecon: Data-driven responsive control of physics-based characters," *ACM Trans. Graph.*, vol. 38, Nov. 2019.

[2] R. Parent, *Computer Animation: Algorithms and Techniques.* Morgan Kaufmann Publishers Inc., 2012.

[3] J. Lasseter, "Principles of traditional animation applied to 3D computer animation," *SIGGRAPH Comput. Graph.*, vol. 21, p. 35–44, Aug. 1987.

[4] L. Ciccone, C. Öztireli, and R. W. Sumner, "Tangent-space optimization for interactive animation control," *ACM Trans. Graph.*, vol. 38, July 2019.

[5] D. J. Wiley and J. K. Hahn, "Interpolation synthesis for articulated figure motion," in *1997 Virtual Reality Annual International Symposium, VRAIS '97*, pp. 156–160, IEEE Computer Society, 1997.

[6] S. Clavet, "Motion matching and the road to next-gen animation," in *Proceedings of GDC 2016.*

[7] M. Miller, D. Holden, R. Al-Ashqar, C. Dubach, K. Mitchell, and T. Komura, "Carpet unrolling for character control on uneven terrain," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games, MIG '15*, p. 193–198, Association for Computing Machinery, 2015.

[8] A. Witkin and Z. Popovic, "Motion warping," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95*, p. 105–108, Association for Computing Machinery, 1995.

[9] C. Rose, M. F. Cohen, and B. Bodenheimer, "Verbs and adverbs: Multidimensional motion interpolation," *IEEE Comput. Graph. Appl.*, vol. 18, p. 32–40, Sept. 1998.

[10] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Trans. Graph.*, vol. 23, p. 559–568, Aug. 2004.

[11] T. Mukai and S. Kuriyama, "Geostatistical motion interpolation," *ACM Trans. Graph.*, vol. 24, p. 1062–1070, July 2005.

[12] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović, "Style-based inverse kinematics," *ACM Trans. Graph.*, vol. 23, p. 522–531, Aug. 2004.

[13] S. Levine, J. M. Wang, A. Haraux, Z. Popović, and V. Koltun, "Continuous character control with low-dimensional embeddings," *ACM Trans. Graph.*, vol. 31, July 2012.

[14] D. Holden, J. Saito, and T. Komura, "A deep learning framework for character motion synthesis and editing," *ACM Trans. Graph.*, vol. 35, July 2016.

[15] D. Holden, T. Komura, and J. Saito, "Phase-functioned neural networks for character control," *ACM Trans. Graph.*, vol. 36, July 2017.

[16] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Trans. Graph.*, vol. 37, July 2018.

[17] S. Starke, H. Zhang, T. Komura, and J. Saito, "Neural state machine for character-scene interactions," *ACM Trans. Graph.*, vol. 38, Nov. 2019.

[18] K. Lee, S. Lee, and J. Lee, "Interactive character animation by learning multi-objective control," *ACM Trans. Graph.*, vol. 37, Dec. 2018.

[19] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Trans. Graph.*, vol. 21, p. 473–482, July 2002.

[20] M. Büttner and S. C. Clavet, "Motion matching - the road to next gen animation," in *Proceedings of Nucl.ai 2015*.

[21] K. Zadziuk, "Motion matching, the future of games animation... today," in *Proceedings of GDC 2016*.

[22] D. Holden, "Character control with neural networks and machine learning," in *Proceedings of GDC 2018*.

[23] G. Harrower, "Real player motion tech in EA sports UFC 3," in *Proceedings of GDC 2018*.

[24] M. Büttner, "Machine learning for motion synthesis and character control in games," in *Proceedings of i3D 2019*.

[25] F. Zinno, "Ml tutorial day: From motion matching to motion synthesis, and all the hurdles in between," in *Proceedings of GDC 2019*.

[26] M. H. Raibert and J. K. Hodgins, "Animation of dynamic legged locomotion," *SIGGRAPH Comput. Graph.*, vol. 25, p. 349–358, July 1991.

[27] K. Yin, K. Loken, and M. van de Panne, "SIMBICON: Simple biped locomotion control," *ACM Trans. Graph.*, vol. 26, July 2007.

[28] S. Coros, P. Beaudoin, and M. van de Panne, "Generalized biped walking control," *ACM Trans. Graph.*, vol. 29, pp. 130:1–130:9, July 2010.

[29] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, pp. 206:1–206:11, Nov. 2013.

[30] D. B. da Silva, R. F. Nunes, C. A. Vidal, J. B. C. Neto, P. G. Kry, and V. B. Zordan, "Tunable robustness: An artificial contact strategy with virtual actuator control for balance," *Comput. Graph. Forum*, vol. 36, no. 8, pp. 499–510, 2017.

[31] V. B. Zordan and J. K. Hodgins, "Motion capture-driven simulations that hit and react," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '02*, p. 89–96, Association for Computing Machinery, 2002.

[32] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012*, pp. 4906–4913, IEEE, 2012.

[33] M. da Silva, Y. Abe, and J. Popovic, "Simulation of human motion data using short-horizon model-predictive control," *Comput. Graph. Forum*, vol. 27, no. 2, pp. 371–380, 2008.

[34] J. D. Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018*, pp. 1–9, IEEE, 2018.

[35] A. Macchietto, V. Zordan, and C. R. Shelton, "Momentum control for balance," *ACM Trans. Graph.*, vol. 28, pp. 80:1–80:8, July 2009.

[36] S. Jain and C. K. Liu, "Modal-space control for articulated characters," *ACM Trans. Graph.*, vol. 30, pp. 118:1–118:12, Oct. 2011.

[37] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu, "Sampling-based contact-rich motion control," *ACM Trans. Graph.*, vol. 29, July 2010.

[38] L. Liu, K. Yin, and B. Guo, "Improving sampling-based motion control," *Comput. Graph. Forum*, vol. 34, pp. 415–423, May 2015.

[39] Y. Lee, M. S. Park, T. Kwon, and J. Lee, "Locomotion control for many-muscle humanoids," *ACM Trans. Graph.*, vol. 33, pp. 218:1–218:11, Nov. 2014.

[40] U. Muico, Y. Lee, J. Popović, and Z. Popović, "Contact-aware nonlinear control of dynamic characters," *ACM Trans. Graph.*, vol. 28, July 2009.

[41] K. Ding, L. Liu, M. van de Panne, and K. Yin, "Learning reduced-order feedback policies for motion skills," in *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '15*, pp. 83–92, ACM, 2015.

[42] L. Liu, M. v. d. Panne, and K. Yin, "Guided learning of control graphs for physics-based characters," *ACM Trans. Graph.*, vol. 35, pp. 29:1–29:14, May 2016.

[43] K. W. Sok, M. Kim, and J. Lee, "Simulating biped behaviors from human motion data," *ACM Trans. Graph.*, vol. 26, July 2007.

[44] J. Kober and J. Peters, *Learning Motor Skills: From Algorithms to Robot Experiments*, pp. 9–67. Springer International Publishing, 2014.

[45] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *arXiv:1808.00177(cs)*, 2018.

[46] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv:1804.10332(cs)*, 2018.

[47] J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Sci. Robotics*, vol. 4, no. 26, 2019.

[48] X. B. Peng, G. Berseth, and M. van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Trans. Graph.*, vol. 34, July 2015.

[49] X. B. Peng, G. Berseth, and M. van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Trans. Graph.*, vol. 35, July 2016.

[50] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *arXiv:1707.02286(cs)*, 2017.

[51] S. Coros, P. Beaudoin, and M. van de Panne, "Robust task-based control policies for physics-based characters," *ACM Trans. Graph.*, vol. 28, p. 1–9, Dec. 2009.

[52] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Trans. Graph.*, vol. 36, June 2017.

[53] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Trans. Graph.*, vol. 36, July 2017.

[54] L. Liu and J. K. Hodgins, "Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning," *ACM Trans. Graph.*, vol. 37, pp. 142:1–142:14, July 2018.

[55] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, July 2018.

[56] N. Chentanez, M. Müller, M. Macklin, V. Makoviychuk, and S. Jeschke, "Physics-based motion capture imitation with deep reinforcement learning," in *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games, MIG '18*, p. 1–10, Association for Computing Machinery, 2018.

[57] B. Graf, "Quaternions and dynamics," *arXiv:0811.2889(math)*, 2008.

[58] D. Eberly, "Quaternion algebra and calculus," *www.geometrictools.com*, 2010. www.geometrictools.com/Documentation/Quaternions.pdf.

[59] D. Bollo, "Inertialization: High-performance animation transitions in Gears of War," in *Proceedings of GDC 2018*.

[60] R. Mukundan, "Quaternions: From classical mechanics to computer graphics, and beyond," in *Proceedings of the 7th Asian Technology Conference in Mathematics, 2002*.

[61] K. Erleben, "Velocity-based shock propagation for multibody dynamics animation," *ACM Trans. Graph.*, vol. 26, p. 12–es, June 2007.

[62] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96*, p. 137–146, Association for Computing Machinery, 1996.

[63] M. R. Abbasifard, B. Ghahremani, and H. Naderi, "A survey on nearest neighbor search methods," *International Journal of Computer Applications*, vol. 95, pp. 39–52, 06 2014.

[64] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861 – 867, 1993.

[65] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018.

[66] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series, MIT Press, 2018.

[67] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347(cs)*, 2017.

[68] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the 4th International Conference on Learning Representations, ICLR 2016*.

[69] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018. www.github.com/openai/spinningup.

[70] Autodesk Inc., "MotionBuilder," 2015. www.autodesk.com/motionbuilder.

[71] D. Lee, M. Glueck, A. Khan, E. Fiume, and K. Jackson, "Modeling and simulation of skeletal muscle for computer graphics: A survey," *Found. Trends. Comput. Graph. Vis.*, vol. 7, no. 4, p. 229–276, 2012.

[72] Nvidia Corporation, "PhysX," 2018. www.developer.nvidia.com/physx-sdk.

[73] Erwin Coumans, Open Source Community, "Bullet physics," 2019. www.github.com/bulletphysics/bullet3.

[74] M. Mach, "Physics animation in Uncharted 4: A Thief's End," in *Proceedings of GDC 2017*.

[75] D. Holden, "Robust solving of optical motion capture data by denoising," *ACM Trans. Graph.*, vol. 37, July 2018.

[76] *Anatomy & physiology*. OpenStax College, 2013.
www.openstax.org/details/books/anatomy-and-physiology.

[77] L. Glondu, S. C. Schvartzman, M. Marchal, G. Dumont, and M. A. Otaduy, "Efficient collision detection for brittle fracture," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '12*, p. 285–294, Eurographics Association, 2012.

[78] Open Source Community, "Pytorch," 2019.
www.pytorch.org.

[79] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv:1604.00772(cs)*, 2016.

[80] E. Coumans, "Physics for game programmers: Exploring MLCP and Featherstone solvers," in *Proceedings of GDC 2014*.

[81] Aymar Cornut, Open Source Community, "Dear imgui," 2019.
www.github.com/ocornut/imgui.

[82] H. Mokhtarzadeh, F. Farahmand, A. Shirazi-Adl, N. Arjmand, F. Malekipour, and M. Parnianpour, "The effects of intra-abdominal pressure on the stability and unloading of the spine," *Journal of Mechanics in Medicine and Biology*, vol. 12, 2012.

[83] A. Kirmse, *Game Programming Gems 4*. Game Programming Gems Series, Charles River Media, 2004.

[84] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," 2017.
www.github.com/openai/baselines.

[85] M. Claypool, R. Eg, and K. Raaen, "The effects of delay on game actions: Moving target selection with a mouse," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play, Companion Extended Abstracts, CHI PLAY 2016*, pp. 117–123, Association for Computing Machinery, 2016.