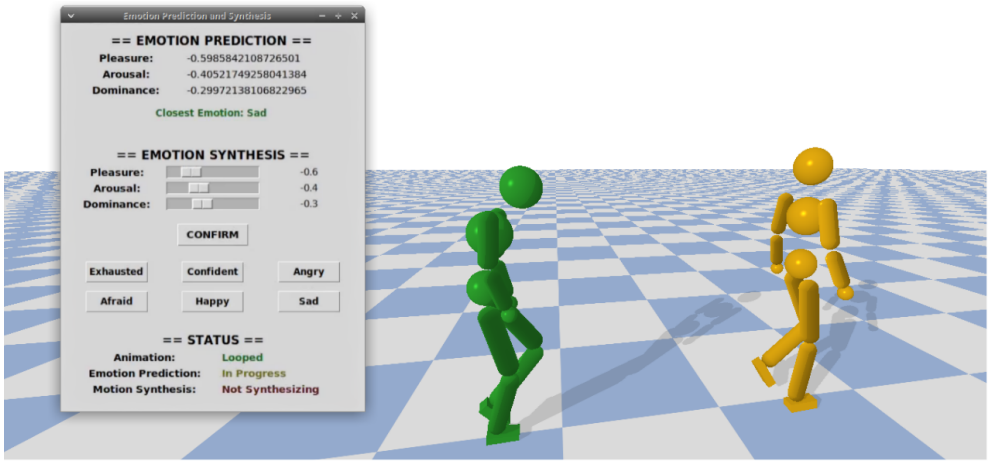
TÉCNICO LISBOA



Emotionally Expressive Motion Controller for Virtual Character Locomotion Animations

Emotional Discernment and Emotional Motion Synthesizer for both Kinematic and Policy-Based Virtual Character Animation

Diogo Gonçalves Silva

Thesis to obtain the Master of Science Degree in

Engenharia Informática e de Computadores

Supervisors: Prof. Pedro A. Santos
Prof. João Dias

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Pedro A. Santos
Member of the Committee: Prof. Carlos António Roque Martinho

November 2022

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

The work presented throughout this document represents not only the conclusion of my master's degree but also the end of an entire academic journey that began eons ago. As such, there are some people who I would like to thank and acknowledge.

First and foremost I want to thank my parents - Graça and Antero - who helped mold the person I am today. I have nothing but appreciation for all the unconditional love and support you have given me throughout my life. For all the countless hours you spent taking care of me, and the insurmountable financial help you provided throughout these years. You gave me everything I ever asked for and allowed me to delve into a multitude of hobbies and activities that shaped my interests and the person I became. You cared for me and gave me all the opportunities that you did not have. I hope to make you proud and can promise you that, wherever I go, I will never forget where I came from, and who was it that allowed me to grow into who I am today. I love you mum and dad.

I would like to thank my girlfriend Cláudia who stood by my side for the past 3 years at the time of writing. I do not know what the future holds for us, but there are no words that express my gratitude for having put up with me for as long as you have and for all the love you showed me. Thank you for being as understanding as you have been and for supporting me and my work as you did. I truly am grateful for having met you and having had the privilege of being your partner.

I also extend my thanks to my dissertation supervisors Prof. Pedro Santos and Prof. João Dias for their insight, knowledge, sharing of ideas and all the words of encouragement provided throughout the elaboration of this thesis.

Finally, I would like to thank my friends who have accompanied me throughout the years. For all the hours spent hanging out, talking or playing games in both good and bad times. Thank you for all the laughter and time we spent together. I want you to know that each and every one of you occupies a very special place in my heart.

Thank you everyone, for your support, love and affection.

Abstract

Style and emotional expressiveness are essential aspects of virtual character computer animation. For a virtual character to display different emotions, motion capture data conveying each desired style has to be recorded, even if the baseline motion is the same. Animators then have to refine and conjoin each recording in order to create the final animations making it a timely and costly process. Although there have been efforts made into the automatic generation of motions through Deep Reinforcement Learning techniques, the problem persists that, for each new desired emotion, reference data displaying said emotion has to be readily available and a new motion has to be learned from scratch. By combining Machine Learning with Emotion Analysis - in particular Laban Movement Analysis and the Pleasure, Arousal, Dominance Emotion State Model - we have developed a system that is capable of not only identifying the perceived emotion of virtual character locomotion animations but that also allows users to alter the character's expressed emotion in real time and without the need of additional data.

Keywords

Computer Animation; Kinematic Models; Physics-Based Models; Machine Learning; Sentiment Analysis; Motion Synthesis

Resumo

Estilo e expressividade emocional são aspetos essenciais da animação em computador de personagens virtuais. Para que uma personagem virtual exiba emoções diferentes, são necessários dados de captura de movimento que transmitam cada estilo desejado, mesmo que o movimento base seja o mesmo. Após a recolha destes dados, os animadores precisam ainda de refinar e juntar cada gravação para criar a animação final, sendo este um processo demorado e caro. Bastantes esforços têm sido direcionados para a geração automática de animações através de técnicas de Deep Reinforcement Learning, mas continuamos com o problema que, para cada nova emoção desejada, precisamos de dados de referência que exibam essa emoção e de gerar uma nova animação. Ao combinarmos Aprendizagem Automática com Análise de Emoções - em particular Laban Movement Analysis e o Pleasure, Arousal, Dominance Emotion State Model - desenvolvemos um sistema que é não só capaz de identificar a emoção de animações de locomoção, permitindo também que utilizadores alterem a emoção expressa pela personagem em tempo real e sem a necessidade de quaisquer dados adicionais.

Palavras Chave

Animação de Computador; Modelos Kinematicos; Modelos Baseados em Fisica; Aprendizagem Automática; Analise de Emoções; Sintetização de Movimentos

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

BVH	Bounding Volume Hierarchy
DRL	Deep Reinforcement Learning
EEMC	Emotionally Expressive Motion Controller
FBC	Feedback Controller
FFC	Feedforward Controller
GAN	Generative Adversarial Networks
GUI	Graphical User Interface
LMA	Laban Movement Analysis
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
PAD	Pleasure, Arousal, Dominance
PPO	Proximal Policy Optimization
RCM	Russel's Circumplex Model
RL	Reinforcement Learning
RSI	Reference State Initialization
SMOTE	Synthetic Minority Oversampling Technique
SSE	Sum of Squared Errors
URDF	Unified Robot Description Format
VAE	Variational AutoEncoders

1

Introduction

Contents

Throughout this chapter we provide some insight as to what problem we are aiming to tackle, what our proposed solution is and what benefits it generates. We start by explaining the motivation behind our work. We then move on to explicitly stating what the problem we're trying to solve is. Finally we lay out what our contributions are, briefly going over the proposed solution.

1.1 Motivation

Conventionally, 3D computer character animation is created by professional human artists who manually tweak a given character's body in key frames and interpolate between them [32]. This process is commonly aided by the usage of motion capture data - mocap [31, 55]. These files consist in recordings of human actors done in a way that their motions can be directly applied to a virtual character. This data, when available, can be used as the basis for an animation, heavily assisting the artist and speeding up the whole animation process.

Automatic physics-based character animation generation aims to synthesize realistic and natural-looking motions using only reference mocap files, without the need of manual animation work. Recent advancements made in Deep Reinforcement Learning (DRL) algorithms have allowed for the construction of systems [26, 39, 43] able to successfully learn and reproduce physically accurate motor skills in a plethora of motions such as dances, locomotion and other body gymnastics. These physics-based motion learning systems have major commercial value especially in the to the Film and Video-Game industries, the two currently highest grossing entertainment industries [21, 59]. They allow for overall animation costs to be cut by requiring less manual animation work. Moreover, these newer systems are capable of creating character controllers that are both robust and capable of generalizing to a wide array of situations, such as having the character automatically recover from being hit with an object, or making it so the character can keep performing the motion regardless of the inclination of the terrain.

Aside from generating the animation itself, animators are also tasked with enabling characters with the ability to express different emotions. Styles and emotions are an important aspect of generating realistic and believable virtual characters. This can be done in a manner of different ways, such as through changes in the character's voice and tonality, facial expression or through the character's body language. This latter method of emotional control has animators generate different variants of the same motion but with slight alterations to the character's body language, indicative of their current emotional state. Such expressiveness conveyed by the character's movements is paramount to properly carry a story, setting a scene's tone and generating an intended impact on the viewer.

A practical example of the usage of emotional expression through body language in virtual characters can be seen in the Video Game "The Sims 4"[1].In this game, characters are able to express a wide array

[1] https://www.ea.com/games/the-sims/the-sims-4

of emotions, with their emotional state being influenced by their surroundings and overall needs. Aside from facial animations, these emotions are expressed to the player by altering the character's baseline animations, as illustrated in Figure 1.1. For example, if a character is feeling tired or lethargic, their walking animation will slope their back forward and drop their arms to the sides to showcase their low energy. Each of these altered animations had to be manually created by human animators from scratch.

Figure 1.1: A character in the video game "The Sims 4" walking in 4 different emotional styles.

1.2 Problem

In essence, a problem present in current computer animation is the fact that, should animators want their characters to convey different emotions, different variants of the same baseline "neutral" animation have to be created. For example if an animator wants a character to walk sadly, happily and angrily, they need to create at least three different animations of the character performing the same motion but expressing a different body language. The issue is then exacerbated by the fact that if the authors now want the character to convey the same set of emotions, but in a different emotion, for example, running, a whole new animation stack has to be created from scratch. This issue persists whether the animators are using state of the art intelligent systems for automatic motion learning or conventional manual computer animation. Furthermore, this usually also means that mocap data of an actor conducting each motion-emotion pairing has to be readily available.

Having to generate an animation stack comprised of the same motion in each desirable emotional style makes creating characters capable of expressing and changing between emotions a tedious, time-consuming and expensive task. With the increase in scale of the produced content comes the need for the creation of more animations with more emotional variety and stylistic nuances, exacerbating the

amount of manual work needed and stretching the realms of what is feasible with conventional means of animation production. We believe that a tool capable of automatically altering a character's motion to convey different emotions without the need of any additional data, training or manual labour, would allow authors to more easily and quickly create emotionally expressive and impactful virtual characters.

1.3 Contributions

We propose a novel solution to this problem that combines the usage of Machine Learning (ML) models and Laban Movement Analysis (LMA) [23] for emotional classification and motion generation. Changes to the motion are applied in real time and get computed after a new desired emotion is specified. New poses are synthesized for the character at each frame, forcing the character to express the desired emotion, whilst still maintaining the baseline motion and movement.

The developed framework was named Emotionally Expressive Motion Controller (EEMC) and is shown in Figure 1.2. The system focuses on locomotive motions - walking, running and dashing - and allows users to edit the virtual character's expressed style and emotion in real-time, any number of times, without slowing down or stopping the animation and without the need for any additional mocap data or motion learning training. Moreover, our system works not only with Kinematic mocap data but also automatically generated Physics-Enabled Policy based character controllers learnt, for example, using the Spacetime Bounds DRL system [39]. The quality of emotions conveyed by the system's generated motions was validated through different user tests, revealing that the framework manages to produce motions with comparable emotional expressivity to reference, professional-grade mocap animations. An interactible Graphical User Interface (GUI) was also built to showcase the system's capabilities.

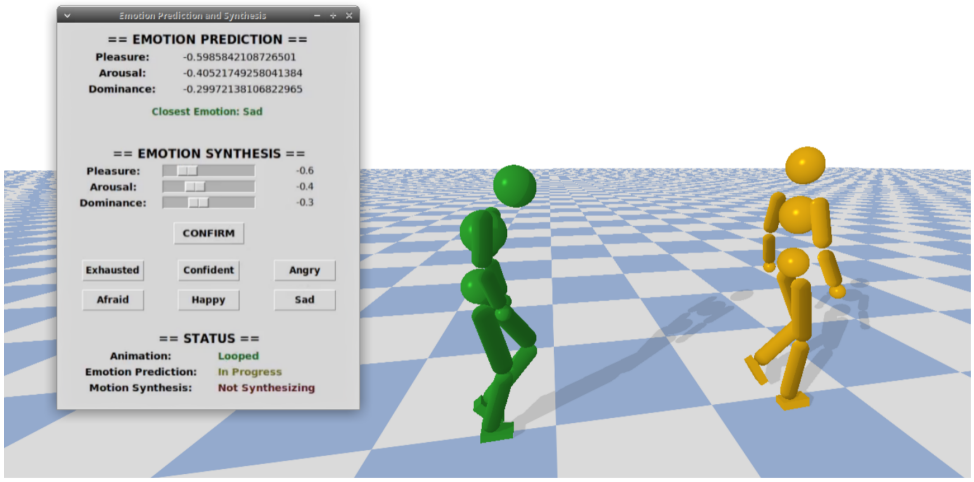


Figure 1.2: The proposed system showcasing a reference baseline motion (right) and a physics-enabled policy-controlled character (left) whose movement have been altered to showcase the desired emotion "Sad".

An article detailing the proposed framework was written and submitted to the *IEEE International Symposium on Multimedia*[2]. The paper has since been anonymously reviewed and approved for publishing during the conference. A different work was also elaborated in parallel dubbed "20 Ways To Answer Binary Questions in Virtual Reality" and is currently in the process of being reviewed and submitted for publishing in *Springer's Virtual Reality* journal[3]. Whilst not directly correlated to the EEMC, this work bootstrapped the discussion of how a character's conveyed emotion - be it through their voice's cadence, facial expression or body language - can majorly influence the impact produced on a viewer. This line of thought eventually inspired the development of the framework described throughout this document.

1.4 Document Outline

This thesis is written in conformity to the following structure:

- **Chapter 1: Introduction** presents the main motivation that lead to the creation of this work, specifies the problem that our work is attempting to tackle and provides a brief overview of the proposed system's objectives, capabilities and merits.
- **Chapter 2: Background** provides information on several topics related to our work, such as character animation models and ML techniques, in order to better help understand the system's functioning.
- **Chapter 3: Related Work** goes over previous works that heavily influenced and inspired our own.
- **Chapter 4: Emotionally Expressive Motion Controller** explains the system's design, architecture and core modules.
- **Chapter 5: Implementation** goes over implementation nuances and decisions, the dataset used and certain engineering problems that had to be surpassed.
- **Chapter 6: Results & Analysis** showcases the finalized system, how well it performs and the results of the conducted user studies.
- **Chapter 7: Conclusion** finishes up by providing a summary of our work and its accomplishments, alongside a brief reflection on possible improvements and future work.

[2] https://www.ieee-ism.org/
[3] https://www.springer.com/journal/10055/

2

Background

Contents

Our work touches upon several topics related to, not only Artificial Intelligence and ML, but also Human Movement and Emotion Analysis. This chapter provides an explanation of several background topics necessary to understand our system, from character animation and generation to specific ML models and human motion emotion analysis paradigms.

2.1 Deep Reinforcement Learning

DRL is one of the areas of ML that has seen the quickest and most promising improvements in recent years. When it comes to the field of learning to mimic animations in physics influenced environments, DRL has become the most widely used approach. New systems utilizing DRL are capable of imitating reference motion capture movements to great success, generating animations that look natural, avoid jerky motions and are robust to the influences of external forces [39, 43].

DRL consists in the combination of two paradigms. Firstly we have Reinforcement Learning (RL). This is a ML methodology based around having agents learn how to perform tasks by experimenting with actions in a controlled environment and being given rewards based on how well they performed [58, 65], as illustrated in Figure 2.1. Through trial and error agents will automatically learn what the optimal actions are given the state of the environment in the current time step. A RL problem is defined by a state space - containing all states the agent can be in -, an action space - consisting of all possible actions the agent can perform -, a transition matrix - all possible state transitions an agent can undergo upon performing an available action, and their corresponding transition probabilities - and a reward function which denotes how desirable each state is by taking as input the agent's state and outputting a value corresponding to the agent's reward for going to that state. The outcome of this type of training is an optimal **policy**, π - a mapping between the agent's state space and their action space. The policy $\pi(s)$ corresponds to the action that the agent should choose to take for every state s in the state space, meaning the action that maximizes the outcome reward for when the agent is in state s. The policy represents the agent's learned strategy, telling them what to do in any given situation.

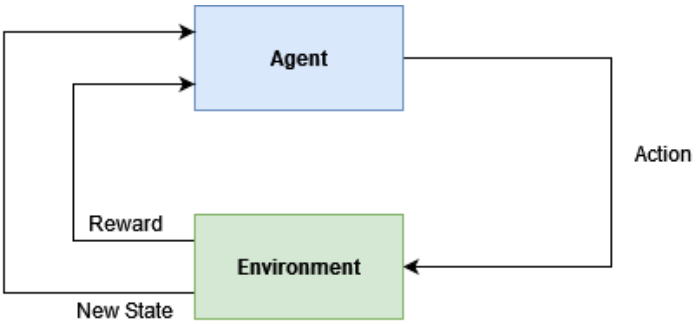


Figure 2.1: The typical RL trial and error learning loop.

There are several ways to find the optimal policy, but one technique that is worth noting is Proximal Policy Optimization (PPO) [53]. This is a form of policy gradient, a type of RL techniques that rely on updating the policy at each time-step to optimize the expected cumulative reward using gradient descent. PPO improves upon previous approaches such as Trust Region Policy Optimization [52] by ensuring that the updated policy doesn't differ too wildly from the previous iteration by clipping the update region.

Deep Learning is a family of ML methodologies encompassing algorithms based on deep neural networks [33]. They differ from other classes of ML by using multiple layered networks, each layer progressively extracting higher level features from the initial input. These types of techniques usually require a lot of data and time to train, but due to the increase in available computational resources they've become some of the most trendy ML methodologies, having seen successful applications in areas such as computer vision [62] and natural language processing [42]. By combining Deep Learning with the aforementioned RL we get DRL. A class of ML where agents learn how to optimally perform a task in their environment by trial and error by optimizing a policy [35, 41]. Rather than having the policy simply be a mapping or lookup table between state-action pairs and rewards, it instead takes the form of a Deep Neural Network tasked with learning how to predict how valuable each state-action pair is. This is especially useful for problems in which the state, action space or both are too large, as is the case with RL for animation generation.

2.2 Autoencoders

Autoencoders are unique types of ML models whose goal is to learn how to translate the initial input into a latent space, and then translate that back into the original representation. Autoencoders are comprised of two different neural networks - the Encoder and the Decoder - as represented in Figure 2.2. The Encoder is responsible for taking the initial input and learning how to compress them into a latent representation with the desired dimensionality. The Decoder does the reverse operation, of learning how to map the latent space back into the original dimensionality.

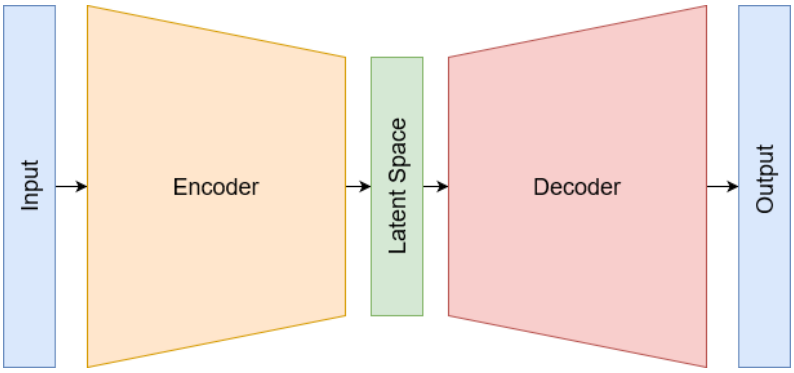


Figure 2.2: The typical Autoencoder architecture.

These types of networks have been used to great success for tasks such as Image Data Compression [14, 37] in the past, but lately, they've been mostly used in Data Analysis and Preparation steps such as Anomaly Detection [13, 69] or Noise Reduction. For our efforts in specific we're making use of the AutoEncoder's applicability in the field of Feature Dimension Reduction [63, 64]. The idea is that by using an AutoEncoder to learn a latent space representation of a set of inputs, with a smaller dimension, we can thereby reduce the dimensionality of the features and therefore the overall complexity of the problem.

2.3 Gradient Tree Boosting

Gradient Tree Boosting is a decision tree ensemble-based ML technique that can be used for regression and classification problems [12, 22]. Decision trees are usually simple models that output a prediction by sequentially partitioning data. These models can be visualized as a flowchart where each node denotes a feature of our data and each branch represents a decision. At the end of each branch we have leaf nodes representing the final prediction of the model. Figure 2.3 illustrates a simple decision tree.

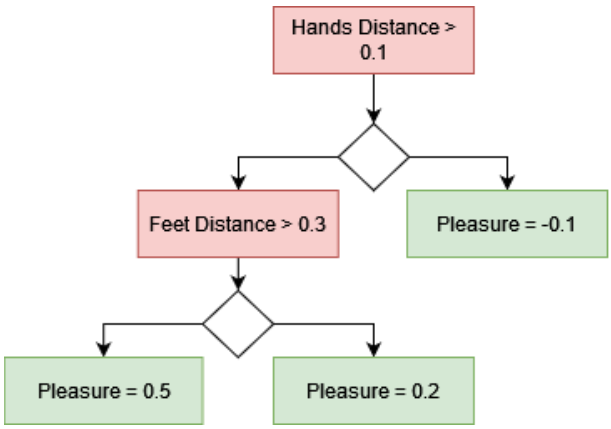


Figure 2.3: A decision tree where we use the values of the features "*Hands Distance*" and "*Feet Distance*" to predict the "*Pleasure*" value.

Ensemble methods are a class of algorithms that work by having a set of several weak models, rather than a single very powerful one [20]. A common issue with Decision Trees is that they tend to overfit to the training data. A way to counteract this is by generating several different decision trees to create an ensemble. There are two main subcategories of Ensemble Methods [45], as exemplified in Figure 2.4. Firstly, Bagging trains each model in parallel, and as such each model has no conceptualization of the others within the ensemble. Secondly we have Boosting, where each model is trained sequentially and as such will be better than the last and learn from its errors.

Gradient Tree Boosting Regression combines several weak decision trees using a Boosting Ensemble. All trees are connected sequentially and aim to optimize and minimize the previous trees' error

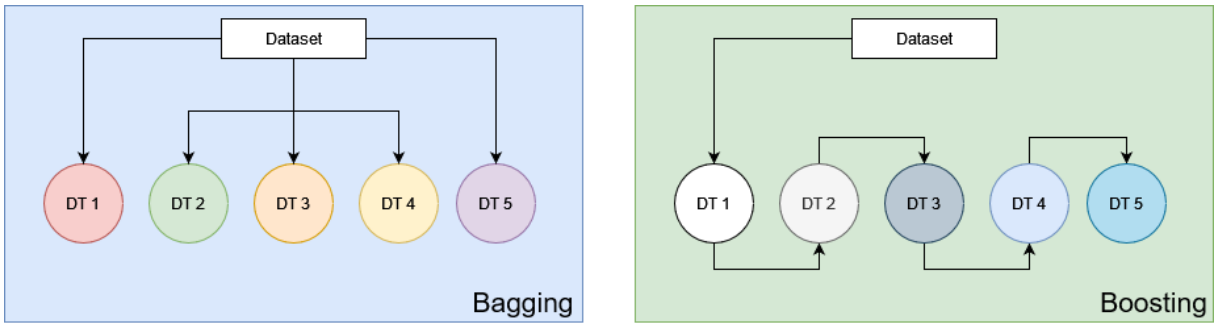


Figure 2.4: The two main types of Ensembles - Bagging and Boosting.

using a loss function such as Mean Squared Error (MSE) or Mean Absolute Error (MAE). The number of estimators is critical as too many trees can lead to a final model that overfits to the data. There are several other key hyper parameters that can be tuned such as the learning rate - how much each model can deviate from the previous one - and the maximum tree depth - which the higher it is the more complex each individual tree is allowed to be.

2.4 Animation Generation

Character Animation refers to the process of animating a virtual character to make it to perform a specific motion. This process is normally done by human artists who manually tweak a character's body, setting its joints' positions in key frames, and interpolating between them [32]. To help with this process, animators usually resort to motion capture data - recordings of human actors, done in a way that their motions can be directly applied to a virtual character or skeleton [31, 55]. Figure 2.5 shows the typical motion capture data studio recording setup. This data can be used as a reference or a baseline for a new animation, heavily aiding the artist speed up their work. However, it does not remove the need for manual human intervention in the creation of new animations.

Figure 2.5: A typical motion capture data studio recording setup [7].

There have been numerous efforts poured into creating virtual character controllers that can automatically learn how to mimic and perform animations. Unlike motion capture data by itself, these controllers aim to bypass the need to manually design and implement each animation that the character is expected to perform. These controllers can be subdivided into two different categories - Those based on **Kinematic Models** and those based on **Physics-Based Models**.

2.4.1 Kinematic Models

Kinematic Models can be defined as *"mathematical models that describe the motion of objects without consideration of forces"* [9]. These models usually structure the body they control as a set of links connected by joints. These links' motions are then constrained by the degrees of freedom of the joints they connect to, hence restricting each link's movements in relation to each other. For example, by defining an arm and a forearm as two links connected by the elbow joint, a kinematic model can restrict the forearm's motion by guaranteeing that it never rotates over 180º degrees in relation to the arm, hence preventing an unnatural positioning of the skeleton.

Lately, there have been efforts made towards automatically building Kinematic Character Controllers through data-driven approaches [34, 50] - stemming from the idea that by being fed a sufficiently large dataset of motion capture data, realistic and natural motions can be synthesized by simply picking which clip to play from the available list of animation frames at any given time and situation. The main issue with this approach comes from the fact that, for them to work, a staggeringly large amount of high quality data needs to be available, least the system risk not being able to properly generalize the motion to work in the presented environments and tasks.

2.4.2 Physics-Based Models

Physics-Based Models refer to a mathematical representation of an object and its motion. Unlike Kinematic Models, they take into consideration external forces in order to simulate the influences of the environment's physics laws over the modeled object [29]. These controllers have the benefits of generating physically accurate motions, lending themselves well to creating natural-looking character animations. However, whilst controllers operating under these physics laws can be created and tuned manually, their design can be challenging, often relying on human insight and the simplification of the underlying physics model [1, 16, 43, 67].

Implementations of Physics-Based Character Controllers have gravitated towards automatic Policy-Based, Physics-Enabled Controller generation through DRL. This is one of the areas that has seen the quickest improvement in recent years, in part due to the increase of available computational resources. A lot of investigation has been done on possible applications of Deep Learning and, more

specifically, DRL, on character animation [26]. As DRL grew to become the *de facto* methodology for generating physics-enabled character controllers, new systems have been created, capable of synthesizing controllers that learn to imitate reference motion capture movements, generating animations that look natural and managing to avoid the jerky movements of past techniques [39, 43].

2.5 Laban Movement Analysis

LMA is a methodology and language for describing human movement that draws inspiration from fields of anatomy, kinematics and psychology [6, 23]. Using this form of analysis, movement description can be broken down into 4 different categories, each possessing a different number of properties. These categories are as follows:

- **Body** - Structure and physical characteristics of the human body while in movement.
- **Effort** - Qualities and dynamics of the movement correlated to the amount of energy, effort, force and weight discharged by the body.
- **Shape** - The way the body changes shape during movement as well as the intricate relationships between the different parts of the body
- **Space** - Body's movement in relation to the environment it is in alongside other relationships with the surrounding space

The concept of LMA has been used in virtual character animation as a means to extracting features from human movements in a manner that can then be parameterized and utilized for generating realistic movements [48]. Table 2.1 showcases just some of the many LMA features that can be extracted using this concept [6].

Table 2.1: 8 examples of LMA features useful for describing human movement.

Feature	Category
Hands Distance	Body
Hip-Ground Distance	Body
Left Foot Velocity	Effort
Pelvis Acceleration	Effort
Volume (All joints)	Shape
Torso Height	Shape
Total Distance	Space
Area Per Second	Space

2.6 Pleasure, Arousal, Dominance Emotional State Model

Emotional classification is a topic related to psychology and affect computing. It involves manners of distinguishing emotions from one another through two main approaches. Emotions can either be considered discrete, meaning humans have a preset array of emotions that they discretely swap between [28, 60], or be defined in accordance to continuous values in dimensional axis, smoothly blending into each other [40, 49]. Focusing on the latter, there are several dimensional models that attempt to place emotions on a 2D or 3D scale. One such model is the widely used Russel's Circumplex Model (RCM) [49], which models emotions into a 2D circular space consisting of an Arousal and Valence axis, describing emotions alongside a Deactivated/Alert continuum and a Pleasure/Displeasure one, correspondingly.

The Pleasure, Arousal, Dominance (PAD) Emotional State Model [40] is an extension of the ideas of RCM which can also be used to describe and classify emotional states. PAD differs from RCM by categorizing emotions according to three dimensions rather than two:

- **Pleasure** - An axis which describes how pleasurable or unpleasant the actor feels.
- **Arousal** - An axis measuring how alert or soporific the actor feels.
- **Dominance** - An axis representing how in control or submission the actor feels.

While RCM is a good, simple model for pinpointing core affect emotions [49], PAD has the advantage of being able to take into account the influence and emotional impact of external forces upon the actor's feelings, through the implementation of the dominance axis. Using this model, an emotion e is represented as a 3-dimensional set of coordinates (p, a, d) corresponding to each of the axis. Image 2.6 showcases a 3D visualization of the PAD model, alongside giving examples of discrete emotions described using this model.

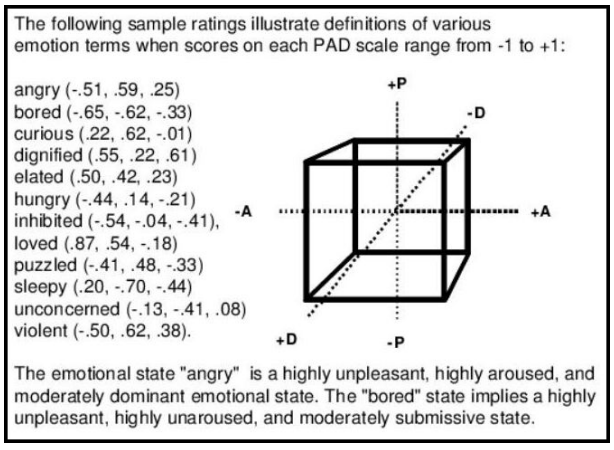


Figure 2.6: Explanation of the Pleasure, Arousal, Dominance 3-Dimensional model [10].

3

Related Work

Contents

There have been lots of recent developments in fields that aim to tackle and solve problems similar to our own. As such, this chapter will serve to mention some of the most prevalent, state of the art works. The efforts described throughout these sections laid the groundwork and served as inspiration for our own system.

3.1 Deepmimic

DeepMimic [43] is a DRL system for physics-based virtual character motion learning. This system is capable of learning complex behaviours, enabling a character to perform movements such as walking and running, alongside harder gymnastics like cartwheels and spins as displayed in Figure 3.1. Furthermore it does it while avoiding generating unnatural movements, jerky motions and other artifacts that previous works struggled with [26, 43].

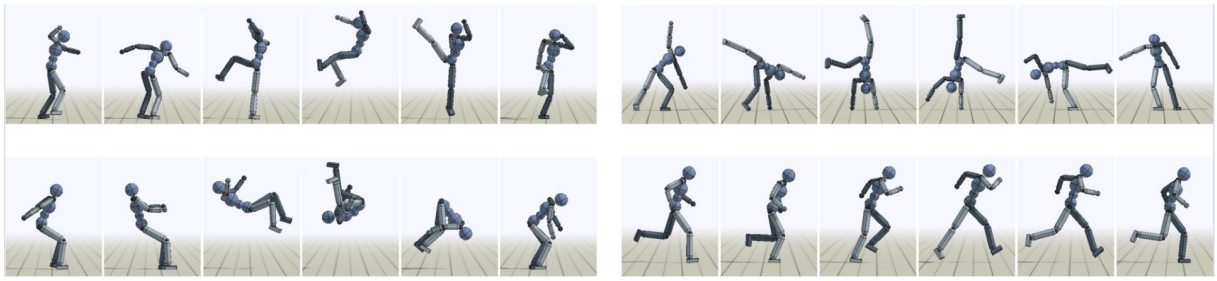


Figure 3.1: An example of motions learnt using DeepMimic [43].

The DeepMimic system is based around the idea of directly rewarding the character controller for having outcome motions that resemble the reference animation whilst also accomplishing additional task objectives. These addition goals are used to indirectly curate the look and feel of the final outcome animation. This *goal-oriented* approach has the benefit of making the motion learning process more robust and allows the character to learn how to recover from external perturbations, like, for example, a ball being thrown at the character while it is walking. The motion will still manage to remain natural with the recovery strategies exhibiting a high degree of robustness.

In terms of how it functions, DeepMimic receives as input a baseline animation in the form of motion capture data or animation keyframes. These motion files have to be provided in a specific Deepmimic-friendly format. The system then uses this data to generate a **control policy**. This policy is a mapping between the state of the character and task-specific goal pair, and the action that needs to be performed to keep up with the reference motion at any given time-step. States describe the configuration of the character's body in terms of the relative positions of each link and joint with respect to the character's pelvis (the root joint), their rotations and their linear and angular velocities. Additionally the state also includes a phase variable denoting whether the character is at the start or at the end of the animation.

The output actions specify the target angles for the character controllers that produce the final torques that need to be applied to the character's joints.

To learn the control policy the system uses an imitation reward derived from how closely the policy controlled character resembles the reference motion and a task-specific reward defined by the additional goals. The policy is modelled as a neural networks in accordance to the DRL paradigm. The used policy network architecture can be seen in Figure 3.2. PPO [53] is used to train and find the optimal policy. Layers 1 through 4 of the policy network are utilized to process the environments heightmap. This is done so that the virtual character is able to perform its motion regardless of slopes and changes in the ground's altitude. Layers 5 through 7 combine the previous layers with the current state and goal of the character. For tasks that don't require a heightmap the policy consists only of layers 5 through 7. The system also uses Reference State Initialization (RSI) allowing the agent to learn a certain timestamp of the animation before they move on to another. For example, for a backflip, the agent first needs to learn how to land, before they learn how to jump, meaning they need to learn the end of the motion before the start. Early Termination is also made use of in the sense that a learning episode is terminated should certain links make contact with the ground, giving the agent a minimal reward of 0. Which body parts are allowed to touch the ground depends on the reference animation and can be specified before the training begins. This provides a means of shaping the reward function to discourage undesirable behaviours. It also provides a curating mechanism that biases the data distribution in favour of samples that may be more relevant for the task, disregarding the others.

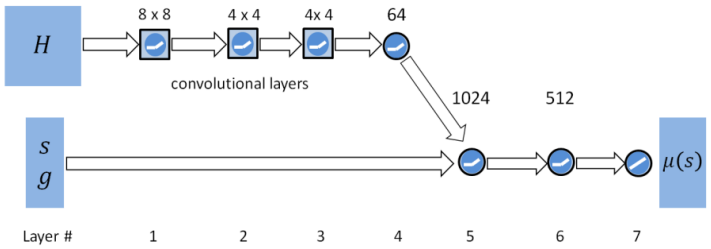


Figure 3.2: DeepMimic's policy network [43].

3.2 Spacetime Bounds

The Spacetime Bounds framework [39] is an extension of DeepMimic [43] which allows for, not only, stylistic exploration via heuristic terms, but also robustness against bad reference data. This system is based around the notion of *Spacetime Bounds* - loose space-time constraints used to limit the search space in a fashion akin to early termination. They bind the character's states in space and time during the RL training process based only on the given reference motion [39]. They also allow for the simplification

of the reward function used during the policy training. With Spacetime Bounds we can use a simple binary survival reward (0 if the character violates the spacetime bound, 1 otherwise) and still get well performing, robust policy character controllers.

Formally, a spacetime is the space of all possible events. These events consist in a state-time tuple representing the system's state at any given time. A trajectory is the sequence of events in a spacetime. We can say that a trajectory is causal if all points on it obey applicable physics laws and dynamic constraints. For example, if a character is falling and suddenly gains height then its trajectory is not causal. With this in mind a **Spacetime Bound** is a subset of a given spacetime and has associated with it a *Feasible Region*.Feasible Regions are a mathematical optimization concept. Traditionally they correspond to the region of a graph that contains all the points that satisfy a given problem's constraints. For Spacetime Bounds the Feasible Region is the set of points on causal trajectories that fall within the constraints specified by the bound. The more Spacetime Bounds we specify the slimmer our Feasible Region will be, since less points of the causal trajectory will befit the imposed constraints. Figure 3.3 exemplifies how imposing more and more Spacetime Bounds - $b1$ and $b2$ - will slim down a Feasible Region of an animation moving from event $e1$ to $e2$.

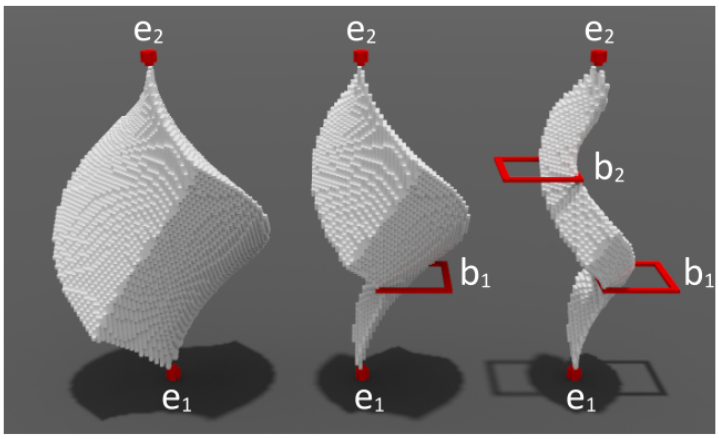


Figure 3.3: A feasible region and the effects of applying spacetime bounds to it [39].

Complex motions have intrinsically smaller feasible regions due to the fact that there are inherently less possible points composing the movement's causal trajectory. For example, more complex gymnastics feats such as a backflip have slim feasible regions even without the usage of Spacetime Bounds. Something like a simple walk animation, however, will have higher volume Feasible Region since there are more ways this motion can be achieved without violating any physics constraints. As such we can generate good policy controllers for complex motions by defining a loose set of Spacetime Bounds since their Feasible Regions are slim by nature.

In order to construct Spacetime Bounds we need to look at the reference trajectory. From it we can define a Spacetime Bound at any given time-step by restricting the state of the character to be within

a certain distance of the reference character's state. For example, a possible spacetime bound could be written as "*the root orientation should be within 50 degrees of the reference root's angle*". As we progress through training each episode will get progressively longer since, as the policy improves, it begins producing trajectories that violate the spacetime bounds less. This results in less episodes being terminated early. Figure 3.4 showcases how, by adding more and more Spacetime Bounds - represented by the red dots and bars - we can slim down the trajectory's feasible region - coloured in blue - and force the policy to produce outcomes that more closely resemble the original line of movement - illustrated by the green line.

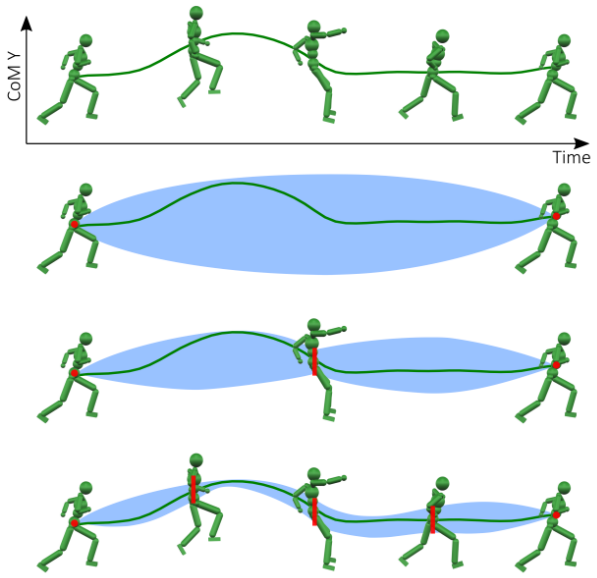


Figure 3.4: How progressively specifying spacetime bounds can restrict the feasible region of an animation and force the policy to more accurately resemble the reference animation [39].

Spacetime Bounds also differs from DeepMimic by making usage of a different policy network architecture, as shown in Figure 3.5. This policy contains two main components. First we have the Feedforward Controller (FFC) which stores the joint angles of the reference motion and linearly interpolates between them during run time to output the default reference joint angles - $\widehat{q}$. Secondly we have the Feedback Controller (FBC) consisting of a two-layer fully connected neural network which takes in the full state vector and outputs the offset joint angles - Δq - that it thinks should be applied to the character at the given time-step. The input state vector is comprised of the phase index - ϕ - and the position, orientation, linear velocity and angular velocity of each of the character's joints alongside the positions of the end-effectors. The final output of the policy comes in the form of joint angles computed by summing the previous two results, $q = \widehat{q} + \Delta q$.

The introduction of Spacetime Bounds to the DeepMimic system provides several benefits. Firstly, there's the fact that Spacetime Bounds are basically just a more refined early termination technique. In

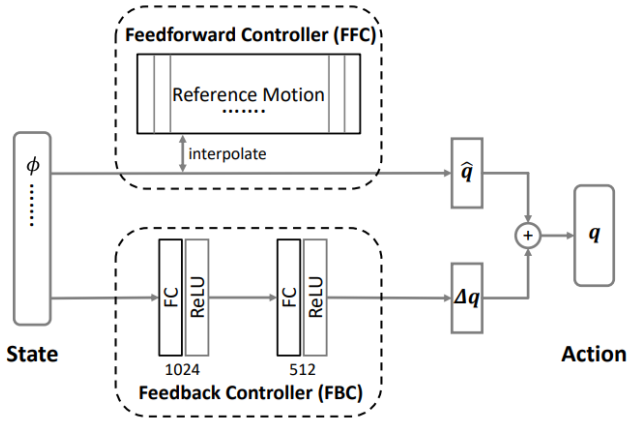


Figure 3.5: Spacetime Bound's Policy Network.

fact, we can specify the same early termination techniques that DeepMimic used through Spacetime Bounds. However, they are much easier to visualize, understand and generalize for different motions. Alongside this, rather than having to audit and design appropriate reward functions for policy learning we can simply use binary rewards - 0 if the character breaks the restrictions, 1 if it does not - and still achieve a high degree of success in imitating our reference animations. Through the definition of looser Spacetime constraints we can also allow our policy to perform *stylistic exploration*. By having less restrictive Spacetime Bounds or through the usage of additional heuristic rules we can conduct our policies to learn how to control the character to exhibit certain types of behaviours rather than just imitating the reference motion. For example, if we added a restriction on the total volume that the character's body can express, we can have our character walk more gallantly or confidently. A downside of this type of stylistic exploration is that it is only possible for less complex motor skills with intrinsically large feasible regions. This is due to the fact that more complex movements possess naturally thinner feasible regions giving the character less leeway to deviate from the reference and preventing it from finding different styles.

3.3 Emotion Control of Unstructured Dance Movements

The Emotion Control of Unstructured Dance Movements system [6] is a program able to analyse a mocap animation's emotion and automatically tweak it to express a different one. This system is capable of extracting a mocap's LMA Features and mapping them into emotional coordinates in the RCM. They also accomplished the inverse process of mapping emotional coordinates into LMA Features. Using these generated features, paired with a set of heuristic rules, they can then edit the mocap data interactively, allowing users to alter the character's displayed emotion with changes being applied within a 10 second interval.

The authors began by extracting 37 spatio-temporal LMA features from their training data. Some of these features include but are not limited to *Pelvis Acceleration*, *Feet Velocity*, *Hand-Hip Distance* and *Torso Height*. The training data that was used consisted of several labeled motion captures of dance performances by different actors in different emotional styles. They then reduced these features by identifying and keeping only those that were both *effective*, in the sense that they presented different values for different emotions, and *consistent*, meaning they had similar values for the same emotion regardless of the actor. After this Feature Selection step they ended up with a total of 31 effective-consistent LMA features which was used to train their system.

Regression was used for emotional prediction by mapping the 31 LMA Features into the 2 dimensional RCM coordinates. The authors began by formulating the Gaussian Radial Basis Function

$$\phi(r) = exp(-\frac{r^2}{2\sigma^2}) \tag{3.1}$$

where σ represents the average distance between the LMA features in the baseline animation's current feature vector and their corresponding counterparts in the feature vector - $\widehat{f_k}$ - of emotion k from the list of 12 unique emotions the system was trained with. The r parameter is a constant used in Gaussian Radial Basis Functions. Function (3.1) was then used in the equation

$$v = w_0 + \sum_{i=1}^{31} w_i \widehat{f^i} + \sum_{k=1}^{12} \lambda_k \phi(\left\| \widehat{f} - \widehat{f_k} \right\|) \tag{3.2}$$

responsible for computing the predicted emotional valence coordinate - v. In terms of parameters, $\widehat{f}$ is the baseline animation's current LMA feature vector containing the 31 extrapolated LMA Features. The weight parameters - w_0, w_i, λ_k - were obtained by fitting Equation (3.2) to the mean values of the training data's LMA features for each of the 12 unique emotions - $\widehat{f_k}$. Note that whilst the shown equations correspond to the *valence* coordinate in the RCM space, the exact same regression was used to compute the *arousal* coordinate - a - with the weights being fitted to the training data's arousal values instead.

For the process of motion synthesis the authors began by creating a way to generate new LMA feature values. This was done by directly mapping from the RCM emotional coordinates back into the LMA feature space, once again, through regression. They used another Gaussian Radial Basis Function

$$\phi(r) = \sqrt{1 + \frac{r^2}{\sigma^2}} \tag{3.3}$$

where σ represents the average distance between the desired emotion's coordinates - e - and their counterparts in emotion e_k, where k corresponds to each of the 12 emotions in the training data. This

function is used as a parameter for equation

$$\widehat{f^i} = w_0 + w_1 v + w_2 a + \sum_{k=1}^{12} \lambda_k \phi(\|e - e_k\|) \tag{3.4}$$

where $\widehat{f^i}$ corresponds to each of the 31 LMA features of the feature vector the equation aims to generate. v and a are the RCM valence and arousal coordinates of the provided desired emotion coordinate tuple e and the weight parameters - w_0, w_1, w_2, λ_k - are obtained by fitting Equation 3.4 to the training data.

To synthesize motion changes from a given desired emotion it wasn't enough to simply generate new LMA Features since the generated features are not guaranteed to be consistent with each other. For example, given the emotion coordinates representing "Sad" in the RCM space, the regression outputs 31 LMA features to express said feature such as "Hand-Hand" distance and "Hand-Hip" distance. These desired features, however, may conflict with each other making it impossible for the character to showcase them both at the same time. As such, the generated features had to be converted into feasible motion changes applicable to the character. Four heuristic rules, shown in Figure 3.6, were devised for this. Each rule aims to modify the position of one of four major control joints - Head, Left Hand, Right Hand and Pelvis. Rules are applied sequentially to avoid conflicts between the generated LMA features. Each rule has associated with it a subset of the LMA Features and one or more coefficients. These coefficients aim to minimize the Sum of Squared Errors (SSE) between the current animation's subset of LMA Features and the corresponding generated ones at each keyframe of the animation.

Feature Modification Rules	**Affected Features**
$g_1(c_1)$: modifies the position of the pelvis p_r: if $c_1 > 1.0$ if $\|d_{pelvis}\| > \|d_{pelvis,T-pose}\|$ p_r+= $(d_{pelvis} - d_{pelvis,T}) \times c_1$ else p_r += $(d_{pelvis,T} - d_{pelvis})(1.0 - 1.0/c_1)$ else p_r+= $d_{pelvis} \times (c_1 - 1.0)$ where d'_{pelvis} is the vector from the pelvis to the ground, and $d_{pelvis,T}$ is the same vector in the T-pose.	$\hat{f}^4, \hat{f}^{18}, \hat{f}^{19}, \hat{f}^{27}, \hat{f}^{28}, \hat{f}^{29}$.
$g_2(c_2)$: modifies the position of the head p_h: p_h+= $d_{torso,T} - [(d_{torso,T} - d_{torso})/c_2] - d_{torso}$ where d_{torso} is the vector from the head to the pelvis, and $d_{torso,T}$ is the same vector in the T-pose.	$\hat{f}^{18}, \hat{f}^{19}, \hat{f}^{20}, \hat{f}^{26}, \hat{f}^{28}, \hat{f}^{29}$.
$g_3(c_{L/R}^{head}, c_{L/R}^{chest}, c_{L/R}^{pelvis}, c_{L/R}^{ground})$: modifies the position $p_{L/R}$ of the left (L) and right (R) hand respectively: $p_{L/R}$ += $[d_{L/R}^{head} \times (c_{L/R}^{head} - 1.0) + d_{L/R}^{chest} \times (c_{L/R}^{chest} - 1.0) + d_{L/R}^{pelvis} \times (c_{L/R}^{pelvis} - 1.0) + d_{L/R}^{ground} \times (c_{L/R}^{ground} - 1.0)]/4$ where $d_{L/R}^{joint}$ is the vector from each hand to the referred joint (head, chest, pelvis, ground).	$\hat{f}^1, \hat{f}^2, \hat{f}^3, \hat{f}^{18}, \hat{f}^{19}, \hat{f}^{21}, \hat{f}^{22}, \hat{f}^{23}, \hat{f}^{26}, \hat{f}^{28}, \hat{f}^{29}, \hat{f}^{30}, \hat{f}^{31}$.
$g_4(c_4)$: modifies the frame rate v: v ×= c_4	$\hat{f}^6, \hat{f}^7, \hat{f}^8, \hat{f}^9, \hat{f}^{10}, \hat{f}^{11}, \hat{f}^{12}, \hat{f}^{13}, \hat{f}^{14}, \hat{f}^{15}, \hat{f}^{16}, \hat{f}^{17}, \hat{f}^{24}, \hat{f}^{25}$.

Figure 3.6: Heuristic rules for Emotional Motion synthesis [6].

Users are allowed to interact with the system to visualize the predicted perceived emotion and change it through the provided GUI. As shown in Figure 3.7, one or more characters can be loaded into the environment and each of their emotions are represented as coloured dots in the RCM diagram on the bottom left. A user can then change each of the character's emotions by simply dragging the corresponding dot into different coordinates, at which point the system will compute and apply changes in 12 seconds (on average). The proposed system, however, has some shortcomings. Most notably, the fact that it was limited to specific types of animations - dances - which present a very unique use case rather than a more generalized one with a larger relevance to broader professional applications - such as locomotions. Moreover, this system works solemnly with kinematic character controllers which directly apply mocap data to a virtual skeleton. As such, the system lacks the ability to be paired with the newer systems for automatic motion generation based on physics-enabled policy-based character controllers.

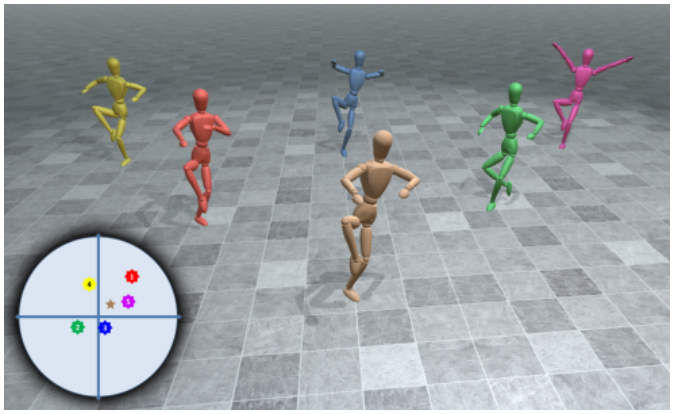


Figure 3.7: A GUI allowing users to change and visualize virtual character's expressed emotions [6].

Although similar in overall purpose, our system diverges from this one in several aspects. Firstly, rather than using the RCM emotional model we resort to the more recent and extensive PAD model [40] which adds a whole new dimension to our emotional classification in the form of the Dominance axis, allowing for a more granular emotional identification. Secondly, instead of using Linear Regressions for the motion-emotion mappings we utilized a more refined and robust technique in Gradient Tree Boosting Regression. We also proposed an alternative methodology for our PAD to LMA mapping based around the combination of an AutoEncoder and the aforementioned regression models. Furthermore our set of LMA features differed, resulting from the mixture and our own experimentation of different combinations of features from several works [2, 4–6, 46]. The EEMC framework was also implemented in a manner that allows it to work with, not only Kinematic controllers reading directly from an animation file, but also with automatically learned Policy-Based, Physics-Enabled character controllers which generate poses at every given frame. Finally the dataset we used consisted of a multitude of emotionally expressive locomotion type animations, which tend to have a more general commercial usage than dances.

4

Emotionally Expressive Motion Controller

Contents

This chapter describes the EEMC System's overall architecture and structure. An explanation of each of the core modules, the way they function and how they're connected to the rest of the system is provided.

4.1 Architecture

The EEMC system can be subdivided into several core sub modules. Figure 4.1 illustrates the connections between the modules and the system's overall architecture. At the core of the system lies a character controller used to make a character execute the intended baseline animation. This controller can either be Kinematic, driven directly by provided mocap, or Policy-Based Physics-Enabled learned, for instance, using the Spacetime Bounds [39] or DeepMimic [43] system.

For Emotion Classification, the system begins by computing a set of LMA features from the frame data extracted from the character. These features are then given to the Emotion Classifier module which, being empowered with a set of ML models, outputs the predicted PAD coordinates.

Emotional Motion Synthesis is triggered whenever new desired PAD coordinates are specified. Firstly, the system converts the new coordinates into a set of LMA features using ML. These features, alongside all of the baseline animation's LMA features, are then given to the Motion Synthesis module which computes new desired joint positions. These, plus the character's current pose, are then provided to the Inverse Kinematics Solver module to generate a new pose for the character.

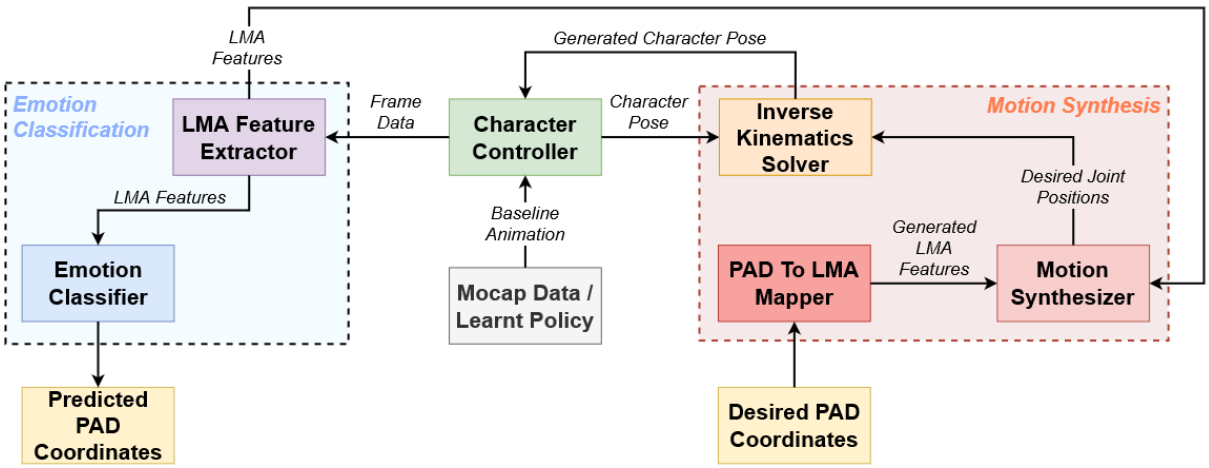


Figure 4.1: An overview of the EEMC's architecture.

4.2 Emotional Classification

The Emotional Classification subsystem is responsible for analyzing the character's movements in order to output the emotion that they're currently expressing, in the form of a set of PAD coordinates. These values are continuous and each range from -1.0 to 1.0, as is customary of the PAD model [40]. In order to achieve this emotional discernment, the Emotional Classification subsystem contains 2 main modules - The **LMA Feature Extraction** module and the **Emotion Classifier**.

4.2.1 LMA Feature Extraction

The **LMA Feature Extraction** module is capable of receiving Frame Data in the form of joint positions and rotations and transforming it into a set of LMA Features. The extracted Frame Data is provided directly by the character controller, which has access to the character's body pose at any given frame. LMA Features are computed using the data received over the previous five frames, which corresponds to our animation's keyframe interval. It should be noted, however, that this extraction rate, as well as other parameters, is configurable and can be adapted to different keyframe intervals and frame-rate. The LMA Features are then passed along to, primarily, the Emotion Classifier. Additionally, this module can also be used independently from the EEMC system to extract and store a mocap file's LMA Features.

Each extracted LMA Feature Set is composed of **25 different LMA Features**, shown in Table 4.1. These features were inspired by previous LMA emotional discernment efforts [2, 4–6, 46]. In order to reach the final chosen set of features data analysis steps were conducted. In particular, for feature selection, we evaluated feature correlations and overall performance on the LMA to PAD regression tasks. Different feature combinations, such as the inclusion of joint jitter and joint heights to ground, were considered and experimented with. The finalized set of features were those that produced an overall better, more consistent output on our ML models, proving to be sufficient to discern between emotions whilst avoiding being too highly correlated and redundant amongst each other.

Table 4.1: Our set of 25 LMA Features.

Feature	f	Category
Max Hand Distance	f_1	Body
Avg. Left Hand - Hip Distance	f_2	Body
Avg. Right Hand - Hip Distance	f_3	Body
Max Stride Length	f_4	Body
Avg. Left Hand - Chest Distance	f_5	Body
Avg. Right Hand - Chest Distance	f_6	Body
Avg. Left Elbow - Hip Distance	f_7	Body
Avg. Right Elbow - Hip Distance	f_8	Body
Avg. Chest - Pelvis Distance	f_9	Body
Avg. Neck - Chest Distance	f_{10}	Body
Avg. Total Body Volume	f_{11}	Shape
Avg. Lower Body Volume	f_{12}	Shape
Avg. Upper Body Volume	f_{13}	Shape
Avg. Area between Hands and Neck	f_{14}	Shape
Avg. Area between Feet and Hip	f_{15}	Shape
Left Hand Speed	f_{16}	Effort
Right Hand Speed	f_{17}	Effort
Left Foot Speed	f_{18}	Effort
Right Foot Speed	f_{19}	Effort
Neck Speed	f_{20}	Effort
Left Hand Acceleration Magnitude	f_{21}	Effort
Right Hand Acceleration Magnitude	f_{22}	Effort
Left Foot Acceleration Magnitude	f_{23}	Effort
Right Foot Acceleration Magnitude	f_{24}	Effort
Neck Acceleration Magnitude	f_{25}	Effort

4.2.2 Emotion Classifier

The **Emotion Classifier** is the module that actually converts the character's current LMA Features into PAD coordinates, representative of the expressed emotion. This module is equipped with pretrained LMA to PAD regression models and algorithms to analyze both motion's emotion at its current time-step, and the overall emotion showcased throughout the animation in its entirety.

To classify the motion's perceived emotion a set of Gradient Tree Boosting Regressors [22] was first trained to map our LMA Features into PAD coordinates. Three different regressors were used - one for each emotional coordinate. Each regressor took as input our set of 25 LMA Features and output the corresponding predicted coordinate. Figure 4.2 illustrates this LMA to PAD mapping process. Regression was used in lieu of classification due to the usage of the PAD Model [40]. As such, rather than simple discrete emotions like "Sad" or "Happy", the system instead tries to infer continuous Pleasure, Arousal and Dominance values ranging in an uninterrupted spectrum between -1.0 and 1.0.

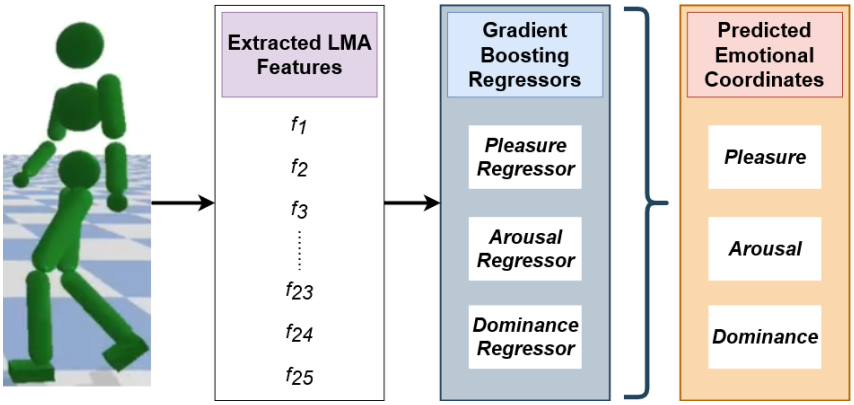


Figure 4.2: The process of using Gradient Boosting Regressors to predict the PAD coordinates of a set of LMA features extracted from a motion.

Using the trained predictors it is then possible to identify a given motion's perceived emotion in real time. Emotional Coordinate prediction is done automatically during an animation's run time. The prediction is done in parallel to the animation's main display loop through multi-threading so as to not slow down the motion's visualization. During an animation's playtime, LMA Feature Sets are computed by the LMA Feature Extraction module and stored internally. Rather then starting a new coordinate prediction process every time a new feature set is computed, we instead opted to buffer the sets into batches. This was done to avoid starting too many emotional identification processes, which would have hindered the program's performance due to the process initialization overhead and the fact that too many processes would be running in parallel at the same time. This would then lead to stutters and slowdowns on the animation's display window. Several batch sizes were experimented with. Each batch ended up consisting of ten feature sets. Whenever one of the prediction processes finishes, the most up to date emotional coordinates are given to the GUI manager for display. At the end of the animation, a

final prediction process is called in order to predict the animation's overall emotional coordinates, taking into account all predictions made up to that point. This entire process is described in Algorithm 4.1.

Algorithm 4.1: Main Emotion Identification Loop.

```
begin
    currentLMAFeatureSets, storedFrames, processes ⟵ {}
    while animationDisplayLoopRunning do
        if not animationHasFinished then
            currentFrame ⟵ GetCurrentFrame()
            storedFrames.insert(currentFrame)
            if IsKeyframe(currentFrame) then
                currentLMAFeatureSets.insert(GetLMAFeatureSet(storedFrames))
                storedFrames ⟵ {}
            if Count(currentLMAFeatureSets) >= 10 then
                newProcess ⟵
                  CreateNewThread(PredictEmotionalCoordinates(currentLMAFeatureSets))
                processes.insert(newProcess)
                StartNewThread(newProcess)
                currentLMAFeatureSets ⟵ {}
        else
            if Count(currentLMAFeatureSets) > 0 then
                newProcess ⟵
                  CreateNewThread(PredictEmotionalCoordinates(currentLMAFeatureSets))
                processes.append(newProcess)
                StartNewThread(newProcess)
                currentLMAFeatureSets ⟵ {}
            foreach process ∈ processes do
                if notHasFinished(process) then
                    WaitForProcess(process)
            pleasure, arousal, dominance ⟵ PredictFinalEmotionalCoordinates()
```

The Emotion Classifier module contains two main functions described in Algorithm 4.2. The first one is used during the animation's playtime. This function is provided with the current list of LMA features sets. It standardizes these features and then uses the regression models to compute the pleasure, arousal and dominance coordinate values for each feature set. It then computes the average value of each coordinate and returns them, whilst also storing each prediction internally. The second function is called when the animation is done playing and is used to compute the final overall coordinate predictions. This function begins by getting the largest stored absolute value and the number of positive and negative values for all pleasure, arousal and dominance coordinates predicted up to that point. If the largest value is positive and the majority of predictions is also positive, or vice versa, we compute the final prediction as a weighted average, giving more importance to the largest value comparatively to the rest. The idea for this comes from the assumption that the intensity of the animation's intended emotional expression

can vary throughout the motion's course. At some point, however, each of the coordinate's intensity will reach a maximum absolute value, indicative of the true feeling the character is aiming to express. If the aforementioned condition is not verified the final coordinate prediction is simply the average of all predictions made throughout the animations playtime.

Algorithm 4.2: Emotional Coordinate Prediction Functions.

$allPleasurePredictions, allArousalPredictions, allDominancePredictions \longleftarrow \{\}$

Function `PredictEmotionalCoordinates(`$currentLMAFeatureSets$`)`**:**

$standardizedFeatures \longleftarrow Standardizer(currentLMAFeatureSets)$

$pleasurePredictions \longleftarrow PredictPleasure(standardizedFeatures)$
$arousalPredictions \longleftarrow PredictArousal(standardizedFeatures)$
$dominancePredictions \longleftarrow PredictDominance(standardizedFeatures)$

$pleasure \longleftarrow \sum(pleasurePredictions)/Count(pleasurePredictions)$
$arousal \longleftarrow \sum(arousalPredictions)/Count(arousalPredictions)$
$dominance \longleftarrow \sum(dominancePredictions)/Count(dominancePredictions)$

$allPleasurePredictions.add(pleasurePredictions)$
$allArousalPredictions.add(arousalPredictions)$
$allDominancePredictions.add(dominancePredictions)$

return $pleasure, arousal, dominance$

Function `PredictFinalEmotionalCoordinates()`**:**

$finalCoordinatePredictions \longleftarrow \{\}$

foreach $predictions \in \{allPleasurePredictions, allArousalPredictions, allDominancePredictions\}$ **do**

$largestValue \longleftarrow GetHighestAbsoluteValue(predictions)$

$positiveValueCount \longleftarrow CountPositiveValues(predictions)$
$negativeValueCount \longleftarrow CountNegativeValues(predictions)$

if $(largestValue < 0$ **and** $(positiveValueCount > negativeValueCount))$ **or** $(largestValue > 0$ **and** $(positiveValueCount < negativeValueCount))$ **then**

$largestValue \longleftarrow \emptyset$

if $largestValue \neq \emptyset$ **then**

$finalPrediction \longleftarrow \sum(predictions)/Count(predictions) * 0.75 + largestValue * 0.25$

else

$finalPrediction \longleftarrow \sum(predictions)/Count(predictions)$

$finalCoordinatePredictions.add(finalPrediction)$

return $finalCoordinatePredictions$

The emotional coordinates, alongside the closest discrete emotion, are shown to the users in our GUI as shown in Figure 4.3. These predictions are updated throughout the animation's display. The final prediction takes into account all of the classifications made previously, therefore providing an overall

prediction of the entire animation, rather than just the previous ten key frames.The Emotional Classifier module is able to correctly identify the emotions of locomotion animations - walking, dashing and running. We also tested this module with other types of animations, such as dances. In these instances our predictions struggled to pinpoint the exact emotional coordinates but nevertheless tended to land on adjacent emotions within the same PAD model octant. For example, when tested with a “Happy” dance, the predicted coordinates gravitated towards the emotions of “Confidence” and “Pride” meaning that we landed on the correct octant of the PAD model, but were overestimating the motion’s Dominance.

Figure 4.3: Example of 5 different motions and our system’s predicted PAD coordinates.

4.3 Emotionally Expressive Motion Synthesis

The Emotionally Expressive Motion Synthesis subsystem is capable of taking in a new set of desired PAD coordinates and altering the character's motion in real time. Specifying individual PAD coordinate values was chosen over selecting from preset discrete emotions in order to give a more granular control over the produced motion's emotion. Moreover, any discrete emotion can be specified by inputting its corresponding PAD coordinates. Our motion synthesis changes the character's motion in order to make it convey the provided input emotional coordinates. Motion editing is done in real time and can be performed any number of times without slowing down or stopping the animation's display. This subsystem is comprised by the **PAD To LMA Mapper**, the **Motion Synthesizer** and an **Inverse Kinematics Solver**.

4.3.1 PAD to LMA Mapper

The PAD to LMA mapper is the first step of the Motion Synthesis pipeline and is used for LMA Feature Generation. LMA Feature Generation pertains to the process of converting PAD coordinates into representative values for our 25 LMA Feature set. This was a necessary step for our EEMC since its Motion Synthesis subsystem receives, as input, new desired emotional coordinates. These values have to then be converted into LMA Features that can be used by the proceeding Motion Synthesis module.

The LMA Feature Generation process was accomplished through ML using two distinct methodologies - the *Direct* and the *AutoEncoder* approach - as exemplified in Figure 4.4. The Direct approach directly transforms PAD coordinates into LMA features using a set of 25 Gradient Tree Boosting Regressors [22]. The Autoencoder approach, on the other hand, adds an additional step, first transforming the input PAD coordinates into a Latent Feature space and then using an Autoencoder to decode the generated latent features into our set of LMA feature values. Further detail on each of the methodologies' implementation and performance is provided in Chapter 5.

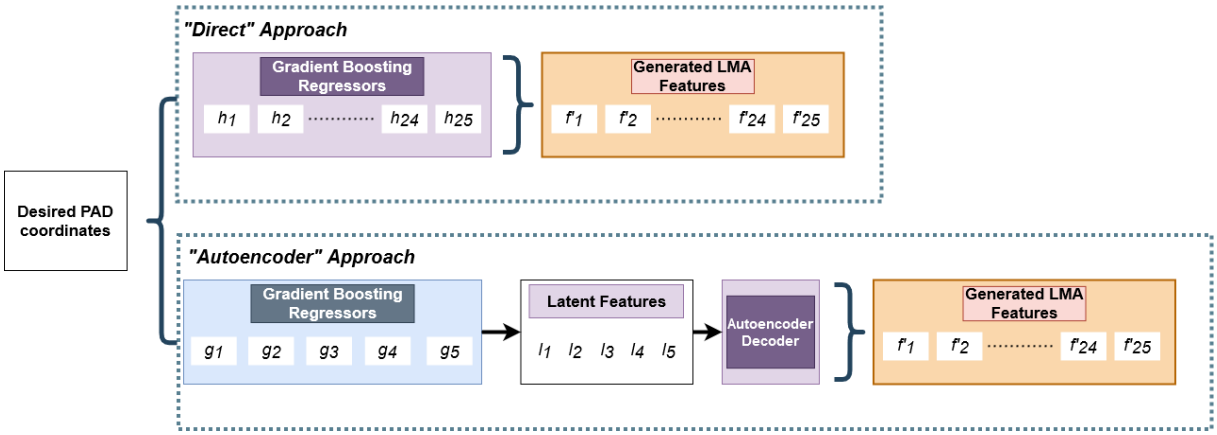


Figure 4.4: The two processes of generating a new LMA feature set given PAD coordinates.

4.3.2 Motion Synthesizer

The **Motion Synthesizer** module is used to compute new desired positions and rotations for the character's core joints. The module receives both the generated LMA feature set from the PAD To LMA Mapper, and the baseline motion's stored LMA feature sets, extracted using the LMA Feature Extractor module. It should be noted that we consider the LMA Feature Extractor as belonging to the Emotional Classification subsystem because it is actively computing and sending it features. The Motion Synthesis subsystem, on the other hand, simply uses it to retrieve the LMA Features that have already been computed during the animation's first playback.

To generate motion changes, the Motion Synthesis module uses a set of six Heuristic Rules, each responsible for tweaking the position or rotation of a core joint - Hips, Chest, Hands, Elbows, Feet and Neck. Each of these rules works by taking into account the current state of the joint its trying to change - its position and rotation - and one or more associated coefficients. The rules are primarily focused on changing upper body joints as these tend to have the most impact on the conveyed emotion, while lower body joints are more important for balance and motion integrity, rather than expression [6]. Each rule was designed with locomotion type motions in mind, following inspiration and emotional movement insights from previous works [3, 6, 24, 46, 51]. For example, higher Dominance emotions tend to have the character broaden their shoulders and increase their overall body volume, lower Arousal, on the other hand, is usually expressed by the character slumping their back forward and swaying their arms to exhibit their low energy. The designed rules attempt to translate this knowledge into motion changes. They each attempt to modify a character's joint in order to make it so the character's LMA Features more closely resemble the generated ones. To do so, each rule uses a set of coefficients representative of the value difference between its associated subset of baseline and generated LMA Features. The subset of LMA Features associated with each rule includes all features that are directly impacted by the joint it is trying to change. Table 4.2 shows each rule, their joint, goal and subset of LMA Features.

Table 4.2: Each of Heuristic Rule and their associated LMA Features.

Rule	Associated LMA Features
g1: Modifies Hip Height *Raises or lowers the character's Hip, changing the body volume.*	Avg. Chest-Pelvis Distance (f_8) ; Avg. Total Body Volume (f_{10}) ; Avg. Lower Body Volume (f_{11}) ; Avg. Area Feet-Hips Triangle (f_{14}) ;
g2: Modifies the Chest's Position *Raises or lowers the character's Chest, making their back appear slumped over or straight.*	Avg. Chest-Pelvis Distance (f_8) ; Avg. Total Body Volume (f_{10}) ; Avg. Upper Body Volume (f_{12}) ;
g3: Modifies the Hands' Positions *Pulls each Hand towards or away from the character's body. Also raises or lowers each Hand towards the character's chest.*	Max Hand Distance (f_0) ; Avg. Left Hand-Hip Distance (f_1) ; Avg. Right Hand-Hip Distance (f_2) ; Avg. Left Hand-Chest Distance (f_4) ; Avg. Right Hand-Chest Distance (f_5) ; Avg. Total Body Volume (f_{10}) ; Avg. Upper Body Volume (f_{12}) ; Avg. Area Hands-Neck Triangle (f_{13}) ;

Rule	Associated LMA Features
g4: Modifies Elbows Positions *Pulls each Elbow towards or away from the character's body, changing their upper volume.*	Avg. Left Elbow-Hip Distance (f_6) ; Avg. Right Elbow-Hip Distance (f_7) ; Avg. Total Body Volume (f_{10}) ; Avg. Upper Body Volume (f_{12}) ;
g5: Modifies the Feets' Positions *Increases or decreases the distance between each Foot, changing the stride length.*	Max Stride Length (f_3) ; Avg. Total Body Volume (f_{10}) ; Avg. Lower Body Volume (f_{11}) ; Avg. Area Feet-Hips Triangle (f_{14}) ;
g6: Modifies Neck Tilt *Tilts the character's Neck towards or away from their chest.*	Avg. Neck-Chest Distance (f_9) ; Avg. Total Body Volume (f_{10}) ; Avg. Upper Body Volume (f_{12}) ;

The coefficients associated with each rule are computed by finding the value that minimizes the distance between the corresponding subset of recorded and generated LMA features. For example, rule g_1 aims to modify the position of the character's Hip joint. To compute c_1, the coefficient associated with rule g_1, we find the value that minimizes the difference between the values of all recorded and generated LMA features that pertain to the hips. For coefficient c_1, these features include $f_9, f_{11}, f_{12}, f_{13}$ and f_{15}. All coefficients are initialized at 1.0 and the objective function we want to minimize is the SSE between the current feature values and the generated ones. Equation

$$\sum_{t} \parallel \hat{f} - f_t c \parallel^2 \tag{4.1}$$

is what we're trying to minimize, where $\hat{f}$ is the generated LMA Feature vector, f_t the reference animation's LMA Feature vector at keyframe t and c the coefficient that we want to optimize for.

In terms of functioning, the Motion Synthesis module begins by collecting all of the baseline animation's LMA Features, computed by the LMA Feature Extraction Module while the animation is playing. Besides these features, the module also stores each of the individual frames' data, in the form of the character's joint positions, rotations and velocities. Whenever new PAD coordinates are specified, a new process is started in a thread parallel to the main motion display. Using the new specified PAD coordinates the Motion Synthesis module calls the PAD to LMA Mapper to generate a new set of 25 LMA Feature values. After generating a set of LMA feature values, the module uses them, in conjunction with the baseline animation's recorded LMA Features, to compute the heuristic rule's coefficients. Each rule is then called sequentially to compute and output a set of positions/rotations for each of the core joints. These changes can then be passed along to the Inverse Kinematics Solver in order to determine the character's new pose.

Joint changes produced by the Motion Synthesis module are synthesized at each frame when the character controller is Policy-Based or for all keyframes as a batch when the controller is Kinematic. This is because when using a Kinematic controller we have access to the character's exact pose at each keyframe, since we are reading directly from a mocap file, meaning we can compute all changes at once and then interpolate between the altered keyframes. For a Policy-Based controller this is not the case since each pose is generated at each frame, so changes have to be computed on a per-frame basis.

4.3.3 Inverse Kinematics Solver

The **Inverse Kinematics Solver** module is the last step of the Motion Synthesis subsystem pipeline and is the one that actually synthesizes the new pose for the character. It takes as input the generated core joints positions and rotations provided by the Motion Synthesis module and the character's current pose. A new pose is then generated using Inverse Kinematics by trying to place the character's joints as close

as possible to their desired positions, whilst maintaining the baseline pose. The Inverse Kinematics Solver also keeps in mind the character's body restraints in order to avoid generating unnatural poses.

The synthesized poses returned by the Inverse Kinematics Solver can be used in lieu of the baseline animation's regular pose at their corresponding time-step. By superimposing these generated poses over the reference ones we are effectively altering the character's motion and, by proxy, its expressed emotion therefore completing the motion synthesis portion of the EEMC system. Figure 4.5 shows an example of the result of our motion synthesis, with the rightmost character displaying the baseline motion in a "Neutral" emotion and the leftmost using the altered motion for the emotional coordinates corresponding to "Tired".

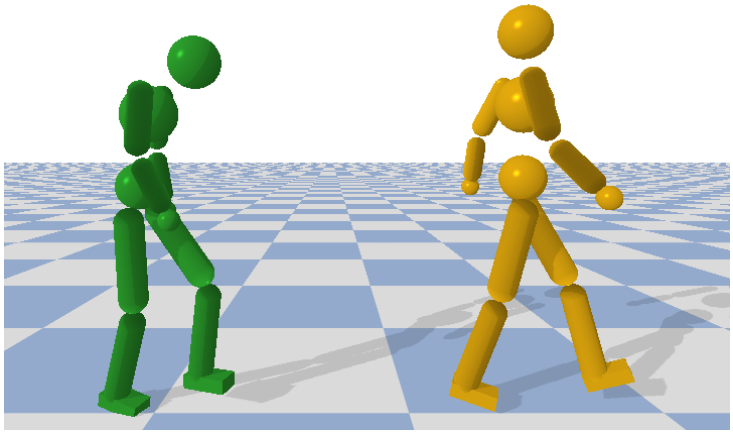


Figure 4.5: Example of an altered motion to convey the feeling "Tired" (left) and the baseline motion (right).

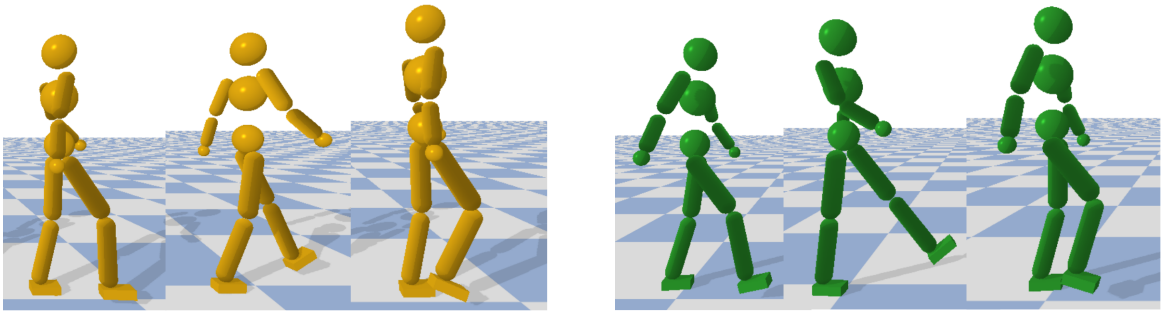
4.4 System Showcase Modules

To prototype and showcase the system's capabilities additional sub-modules were developed. Whilst these are not the main focus of the EEMC system they are still important to mention since they provide a way for users to interact with the underlying framework. The included showcase modules consist of an automatic **motion learning module** and a motion display and **user interaction interface**.

4.4.1 Motion Learning

The SpaceTime Bounds **Motion Learning** module [39] is used to generate new character controller policies in order to showcase the system's ability to work with physics-enabled non-kinematic character controllers. This module can generate character controllers in the form of policies, able to control a character, making it mimic the given reference motion, provided via a mocap file, in a physics-enabled environment. Figure 4.6 compares 3 frames from a learned policy-controlled motion to their counterparts

at the same time-step in the reference animation. It should be noted that this module can be bypassed entirely by simply providing the system with a mocap file directly. By doing so the system will be initialized with a kinematic character controller instead of a policy-based one.



(a) Frames from a kinematic-controlled reference character

(b) Frames from a learned policy-controlled character

Figure 4.6: Frames from a character performing the same motion with different controllers.

4.4.2 User Interface

The User Interface aims to provide a visual representation of the EEMC system, showing a virtual character performing a motion, outputting the Emotional Classification results and allowing users to input new desired PAD coordinates for Motion Synthesis. It consists of 2 parts - the Motion Display window and the Motion Controller & Emotion Classifier GUI as displayed in Figure 4.7.

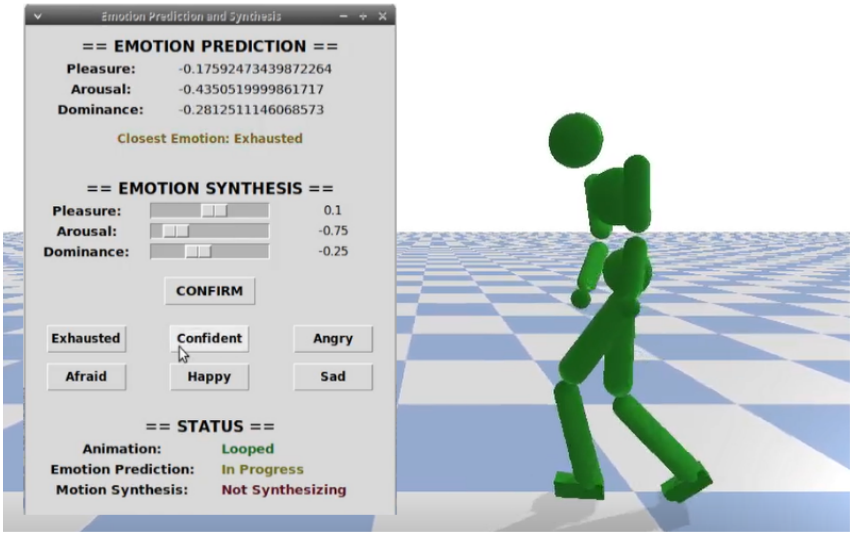
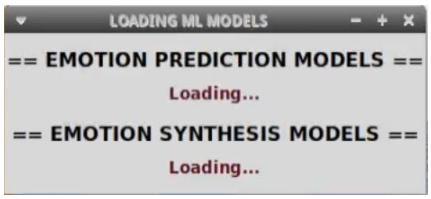
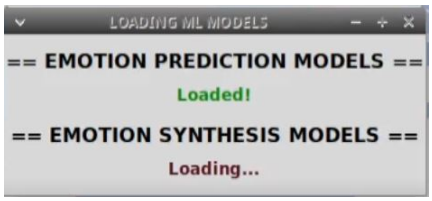


Figure 4.7: A character performing a motion and the GUI used to trigger the EEMC.

Firstly, when starting a program the user is informed that the pretrained Emotion Classification and LMA Feature Generation ML models are being loaded, as shown in Figure 4.8. While this happens, the animation is deterred from starting and the remaining User Interface components sit idly awaiting for the models to finish loading. As such, all user interaction over the Main Display window and GUI is disabled.



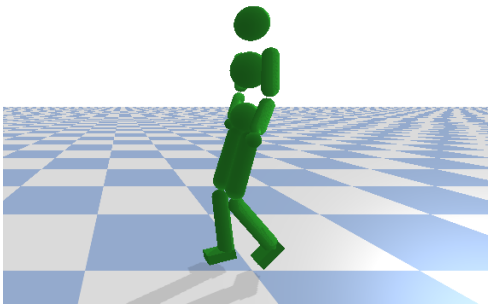
(a) The loading window when the program first starts



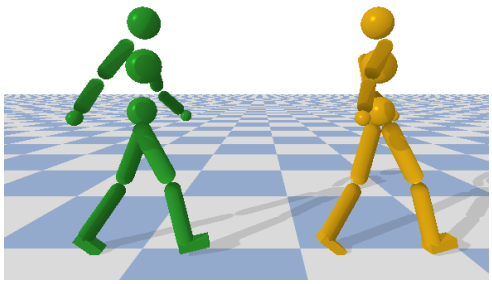
(b) The loading window after the emotion prediciton models have finished loading

Figure 4.8: Our ML model loading display.

The main Motion Display window, presented in Figure 4.9, contains an environment and a character displaying the animation the system was initialized with. Additionally, we provide the option of starting the environment with two characters, with one being the target for our Emotional Motion Synthesis and the other being used to display the baseline reference motion. In terms of interaction, users can fully rotate the camera and zoom in and out. This window, alongside the rest of the system, stays open until manually closed by a user, even if the animation has finished playing.



(a) Motion Display with one character



(b) Motion Display window with a main (left) and a reference (right) character

Figure 4.9: The Motion Display window.

The second half of the User Interface consists of the Motion Controller & Emotion Classifier GUI. This is an additional window, detached from the Main Display one, which the user can move and position at will. The GUI - shown in Figure 4.10 - has three main functions. First, it automatically gets the most up to date emotional predictions from our animation's main loop and shows them to the user. It individually

presents the predicted values for the Pleasure, Arousal and Dominance coordinates. It also uses these coordinates to find the closest discrete emotion from our dataset's list of emotions. The discrete emotion is colour coded green, yellow or red depending on how closely its emotional coordinates approximate the predicted ones. Secondly, at the bottom of the GUI we can find useful system state information. The user is informed of whether the animation is running for the first time, is looping or has finished playing, whether emotion prediction is ongoing or stopped and whether our motion synthesizer is computing changes to the character's movements or in standby. Thirdly, and most importantly, this window allows users to specify new desired emotional coordinates. This can be done through the three provided emotional coordinate sliders. For ease of use we also provided six template emotion buttons. Clicking any of these will automatically change the sliders into the coordinates corresponding to the clicked button's emotion. In order to start synthesizing the motion changes the user has to click the "Confirm" button - which only becomes available after the animation has looped once - at which point the Motion Synthesis module gets activated and starts changing the character's motion in real time.

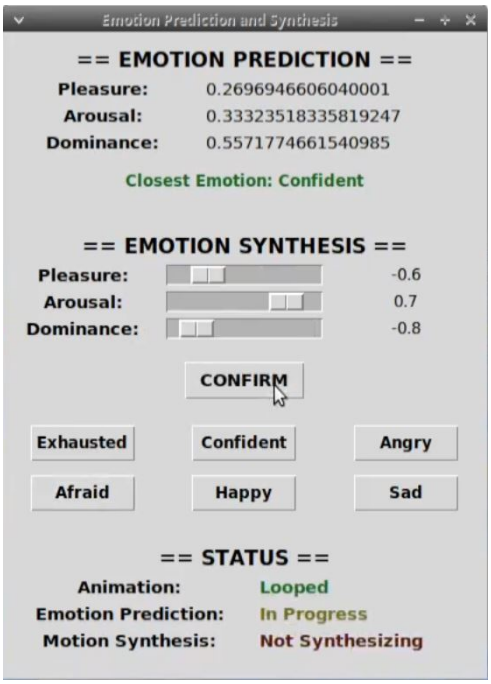


Figure 4.10: The main GUI.

5

Implementation

Contents

This chapter goes over how we went about implementing our system. We go into detail on how the system's main modules were implemented, what the dataset consisted of and what some of the major engineering challenges faced were.

5.1 Dataset

The **Bandai-Namco-Research Motion Dataset** [7] was utilized to train each of the system's ML models. This data consists of Bounding Volume Hierarchy (BVH) files describing a wide array of motions such as walking, running, kicks and dances running at 30 frames per second. Each animation was performed in order to convey a specific style like proud or masculine. The dataset is organized into two sub datasets, with *Bandai-Namco-Research Motion Dataset-1* containing 17 different types of motions in 15 different expression styles and *Bandai-Namco-Research Motion Dataset-2* having 10 different motions, focused around locomotion and hand actions (such as waves), in 7 unique styles. Figure 5.1 shows some of the animation styles included in the Bandai-Namco-Research Motion Dataset.

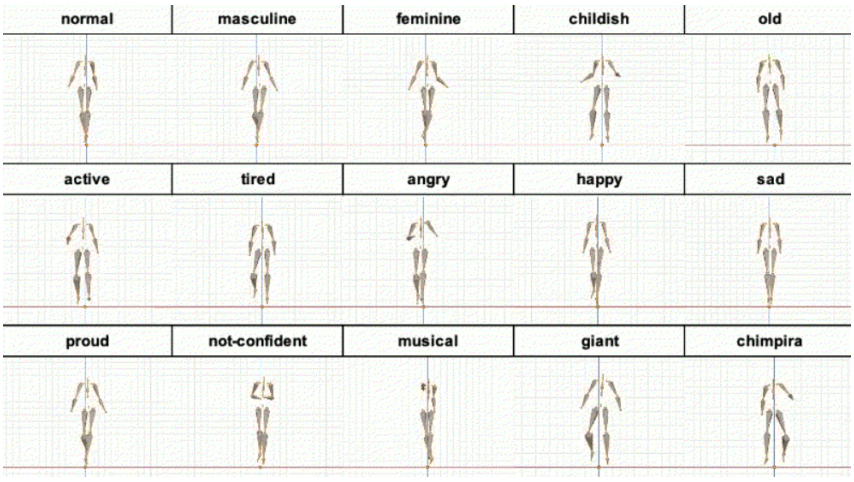


Figure 5.1: Animations from the Bandai-Namco-Research Motion Dataset [7].

This dataset was chosen for a multitude of reasons. For one it was publicly available and free to use under the Creative Commons Public Licenses. Secondly it provided us with a plethora of high quality animations, having been recorded at Bandai Namco's own high-end motion capture studio with motions performed by professional actors. The data was also post-processed to remove any artifacts. Thirdly, despite not covering the entire emotional spectrum of the PAD model, the dataset still contains a fair amount of emotional variety which was necessary to train the EEMC's ML models.

Before usage the dataset was first culled and prepared using the process described in Figure 5.2. As our efforts were primarily focused on locomotion type animations, only Walking, Running and Dashing animations were kept and converted into a Deepmimic-friendly format [36, 43].

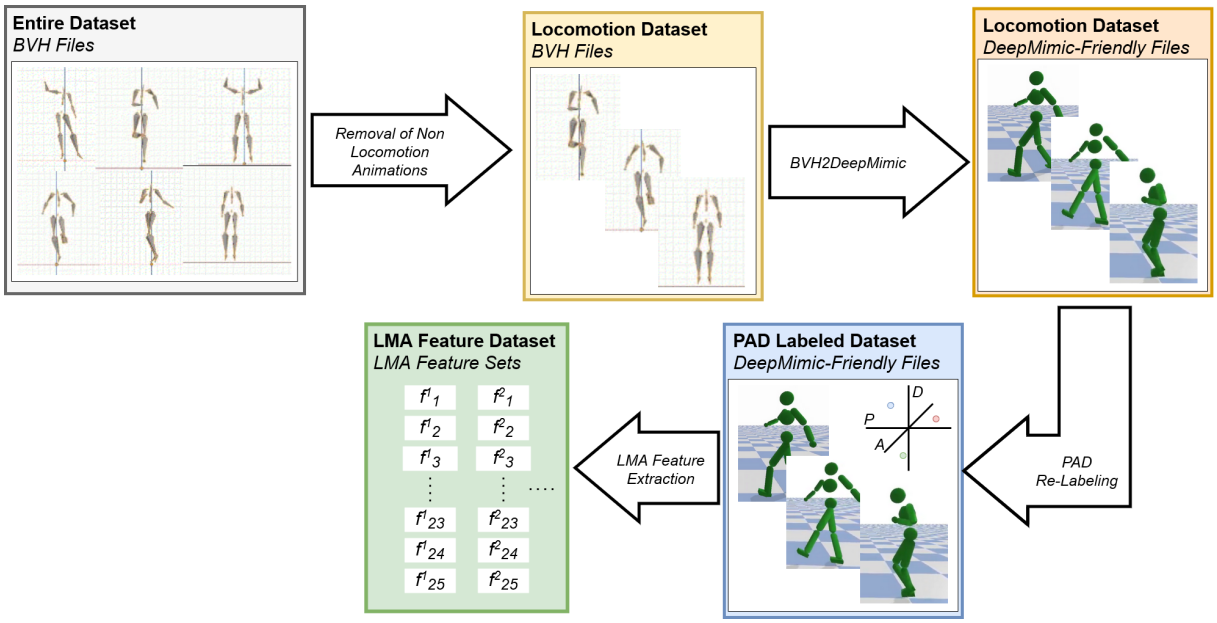


Figure 5.2: The process of converting the Dataset's BVH files into a collection of usable LMA Feature sets.

The original dataset's labels were mapped into corresponding emotions and PAD coordinates [25]. The values chosen for the emotional coordinates were inspired by previous discrete emotion to PAD mapping efforts [6, 25, 68] with minor adjustments to better fit the dataset's animations. Table 5.1 specifies each of the label-emotion mappings, coordinates and number of samples per emotion. The subscript present in certain labels indicates whether that label originally belonged to the Bandai 1 or 2 dataset. This resulted in **468 different animations in 14 emotions**, exemplified in Figure 5.3.

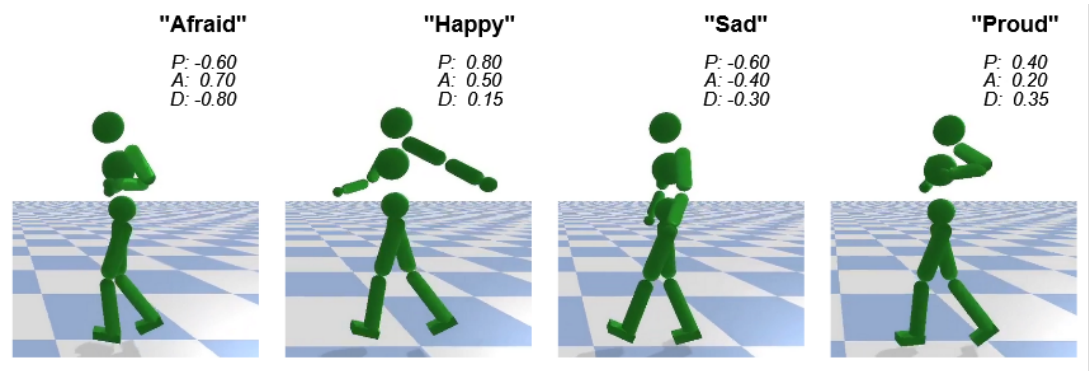


Figure 5.3: Example of 4 motions from our dataset expressing 4 different emotions.

After having been labeled, each of the animation's LMA Features were then extracted using the LMA Feature Extractor Module. In the end, a total of **78551 LMA Feature Sets** were retrieved and labeled according to their corresponding animation's PAD coordinates. These feature sets comprised the data that was then used to train our system's ML models.

Table 5.1: Conversion from the original motion styles to emotions and emotional coordinates.

Original Style	Emotion	PAD Coordinates	No. Samples
Normal	Neutral	(0.05, -0.05, 0.0)	28324
Tired_1	Tired	(0.1, -0.7, -0.2)	1749
Exhausted_2	Tired 2	(-0.1, -0.75, -0.15)	28457
Old_1 / Elderly_2	Exhausted	(-0.1, -0.6, -0.15)	31651
Angry_1	Angry	(-0.5, 0.8, 0.9)	1371
Happy_1	Happy	(0.8, 0.5, 0.15)	1531
Youthful_2	Happy 2	(0.6, 0.4, 0.1)	25939
Sad_1	Sad	(-0.6, -0.4, -0.3)	2406
Proud_1	Proud	(0.4, 0.2, 0.35)	2174
Giant_1	Confident	(0.3, 0.3, 0.9)	1430
Masculine_1	Confident 2	(0.25, 0.15, 0.4)	1239
Masculine_2	Confident 3	(0.3, 0.4, 0.6)	26089
NotConfident_1	Afraid	(-0.6, 0.7, -0.8)	2200
Active	Active	(0.1, 0.6, 0.4)	28564

5.2 LMA Feature Extraction

The LMA Feature Extraction module is responsible for taking in frame data from the main motion display loop and extrapolating the animation's LMA Feature values. This module is primarily used while the system is running to provide the Emotion Classifier and Motion Synthesizer modules with features to use in their corresponding ML models. The extractor can also be used separately from the main system, having been utilized to extract the Bandai-Namco-Research Motion Dataset [7] animation's LMA features, which were, in turn, used to train the LMA to PAD and PAD to LMA ML models.

This module provides several configurable parameters that alter its behavior, from defining whether features should be saved onto a file or stored in memory while the program is running, to whether or not there should be an initial buffer of ignored frames, useful for animations that have a windup time before starting. The full list of parameters is showcased in Table 5.2. The LMA extraction rate is a particularly relevant setting. By default the module extracts features at every fifth frame interval, which for our motion dataset corresponded to every keyframe. A different extraction rate can be defined to fit a different frame-rate or key frame interval by specifying a new pooling rate in seconds. When this happens, the module computes how many frames are displayed per second using the frame's duration parameter and uses it to compute the corresponding frame interval between LMA feature extractions.

In terms of implementation, the LMA Feature Extraction's main function has to be called each time a new animation frame is displayed or whenever a new pose is applied to the virtual character. The module stores the current frame's data internally. After a number of frames equal to the defined extraction interval - by default every fifth frame - has been stored the module takes the cached data and uses it to compute the values of the 25 LMA Features. Feature computation is done through several internal math functions including, but not limited to, velocity and acceleration estimation and triangle area, box volumes and joint

Table 5.2: Configurable parameters of the LMA Feature Extractor module.

Parameter	Type	Description
append_to_outfile	boolean	Whether or not the computed features should be saved to a file.
outfile	string	Path to the file features should be saved to. Only relevant if *append_to_outfile* is set to True.
pool_rate	float	Rate at which LMA features should be extracted, in seconds. If set to -1 LMA Features will be extracted every 5 frames. If set to -2 LMA Features will be extracted every 15 frames.
label	tuple of 3 floats	PAD Label associated with the extracted LMA Features.
ignore_amount	integer	Number of frames that should be ignored at the start before LMA Feature extraction can begin.
round_values	boolean	Whether or not the computed feature values should be rounded.
write_mocap	boolean	Whether or not the provided frame data should be written to a file.
write_mocap_file	string	Path to the file provided frame data should be saved to. Only relevant if *write_mocap* is set to True.

distance calculation. The generated features are then joined together into a set and output to be used by the system's other modules. This output comes in the form of the set described in Listing 5.1.

Listing 5.1: The LMA Feature Extraction output.

```
{
    "frame_counter": index of the frame at which LMA features were computed,

    "label": PAD Emotional Coordinates (3D),

    "lma_features": [
        max hand_distance (1D),
        average l_hand_hip_distance (1D),
        average r_hand_hip_distance (1D),
        max stride length (distance between left and right foot) (1D),
        average l_hand_chest_distance (1D),
        average r_hand_chest_distance (1D),
        average l_elbow_hip_distance (1D),
        average r_elbow_hip_distance (1D),
        average chest_pelvis_distance (1D),
        average neck_chest_distance (1D),
        average total_body_volume (1D),
        average lower_body_volume (1D),
        average upper_body_volume (1D),
        triangle area between hands and neck (1D),
        triangle area between feet and root (1D),
        l_hand speed (1D),
        r_hand speed (1D),
        l_foot_speed (1D),
        r_foot_speed (1D),
        neck speed (1D),
        l_hand acceleration magnitude (1D),
        r_hand acceleration magnitude (1D),
        l_foot acceleration magnitude (1D),
        r_foot acceleration magnitude (1D),
        neck acceleration magnitude (1D)
    ]
}
```

A complementary program for mass feature extraction was also included. This script - Mass Kinematic LMA Extractor - allows for the sequential extraction of the LMA features from several mocap files

and was primarily utilized to extract the Bandai-Namco-Research Motion Dataset [7] animations' LMA Features. By specifying an input directory - path to the directory containing mocap files - and output directory - path to the directory where files with the LMA Features will be stored - users can easily extract the LMA features of several BVH files in a row. Users can also specify paths to Meta Files, which will store a list of every output file's names and emotions. The name and emotion of each output file is inferred from the original's name which should follow the nomenclature *"<emotion>_<file name>.bvh"*.

5.3 LMA to PAD Mapping

To perform the task of Emotional Coordinate Prediction a set of Gradient Tree Boosting Regressors was trained to map the motion's LMA Feature set values into corresponding PAD coordinates. To train each of the LMA to PAD Gradient Tree Boosting Regressors a dataset of LMA Features extracted from the *Bandai-Namco-Research Motion Dataset* [7] was used. Models were trained and tested with both LMA Features extracted at every fifth frame - which corresponded to each animation's keyframe - and features extracted uniformly every half a second - which corresponds to every 15 frames, since the dataset's animations all ran at 30 frames-per-second.

Prior to training the models, data analysis and preparation steps were performed. By looking at the data's sample distribution over emotions we decided to experiment with data balancing using undersampling and Synthetic Minority Oversampling Technique (SMOTE) [11]. As mentioned in the previous section, feature selection was also performed, based on both variance and correlation analysis, resulting in the finalized set of 25 LMA Features. The correlation between each feature and the three individual target PAD coordinates was also analyzed, noting that some features correlated very little to certain coordinates, but very highly to others. For example, the average triangle area between the hands and neck had a fairly high correlation value of 0.33 with the Dominance coordinate but only a 0.04 with Arousal. Taking this into consideration we experimented with creating subsets of the LMA features for each regressor, using only features with a high correlation to the corresponding target coordinate. Finally we did outlier detection using Z-Score evaluation [57] and standardized the data [54].

We generated several datasets for every possible combination of each of the aforementioned data preparation pipeline steps using either the 5 or 15 frame LMA Feature extraction rate. In the end we noted that our models tended to perform better with **just Standardization** and no other additional preparation steps, using LMA Features extracted every fifth frame. We believe data balancing through undersampling was removing too many samples from the dataset leading to a lack of data, whilst SMOTE was generating too many repeated samples. The usage of different subsets of features for each regressor seemed to harm the performance possibly due to the fact that, despite not correlating as highly to the target variable directly, the features removed from each subset nonetheless helped compose the full set

of inputs, providing vital information for granular emotional discernment. Outlier detection neither hindered nor improved the results for the most part as only a very small amount of samples ended up being removed. As to why the features extracted every keyframe performed better than every 0.5 seconds, which in our animations corresponds to roughly every 15th frame, we believe that, aside from generating less overall samples, this quicker extraction frequency failed to capture the slight nuances between shifts in the character's pose as well as the smaller, every keyframe extraction rate.

The models were built using XGBoost [12]. Features were shuffled and split into a Train/Validation and Test set. In total, 80% (62841 samples) of our data was used for Training and Validation and 20% (15710 samples) was left for for Testing. In order to find the optimal hyper parameters for each of regressor we used **Random Search 10-Fold Cross Validation** [8]. Table 5.3 shows which hyper parameters were tuned and their experimented values. The final trained models were evaluated using the Test set and managed to accomplish a MAE of 0.02, 0.06 and 0.03 using the Test set for the Pleasure, Arousal and Dominance coordinates correspondingly. Each coordinate can range between $[-1.0, 1.0]$ meaning that, even on our worst performing coordinate, we still managed to achieve a model with a MAE under 5% of the total spectrum. In terms of MSE all 3 of our models managed to score values under 0.02 over the Test set, with Pleasure scoring 0.002, Arousal 0.013 and Dominance 0.004.

Table 5.3: All hyper parameters that were tuned using Random Search Cross Validation for each regressor.

HyperParameter	Tested Values	Description
learning rate	[0.01, 0.05, 0.1, 0.3, 0.5] *Default: 0.3*	Controls how much the model is allowed to change at each step in response to the estimated error. The lower the value the more conservative the model is, reducing the odds of overfitting but slowing the learning process
subsample	[0.75, 1] *Default: 1*	Subsample ratio of the training samples. A 0.75 subsample value means that prior to training the trees at each iteration 75% of training data is sampled and the rest is discarded.
max depth	[3, 6, 10, 15] *Default: 6*	Maximum depth of the trees. The higher the depth the higher the complexity increasing the odds of overfitting.
alpha	[0.0, 0.25] *Default: 0*	L1 regularization value.

Argument	Type	Description
min child weight	[1, 5, 11, 21] *Default: 1*	Minimum sum of instance weight needed in a child node. When the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight then the parent node won't be further partitioned.
gamma	[0, 0.001, 0.01] *Default: 0*	Minimum estimated error reduction required to partition a leaf node of the tree. The higher the value, the more conservative the model will be as a higher error reduction will be required to make the model more complex.
colsample bytree	[0.75, 1] *Default: 1*	Subsample ratio of features when constructing each tree.
lambda	[1.0, 1.25] *Default: 1*	L2 regularization value.

The line graphs in Figure 5.4 present the predicted and true values of 100 random samples from the Test set, ordered by ascending value. As can be seen, regardless of model and coordinate, the silhouette of the predicted values closely follows the true values' one showcasing how close our predictions

manage to get to the correct values for each sample. Figure 5.5, on the other hand, shows the predicted emotional coordinates of 1000 random samples from our Test set for each emotional pairing and on the PAD 3 Dimensional coordinate system. Each sample is coloured according to its true emotion. As we can see, some predictions do stray slightly from their real emotional coordinates, but they still fall into well defined emotion clusters. Moreover, they seldom stray from the correct octant in the 3D model.

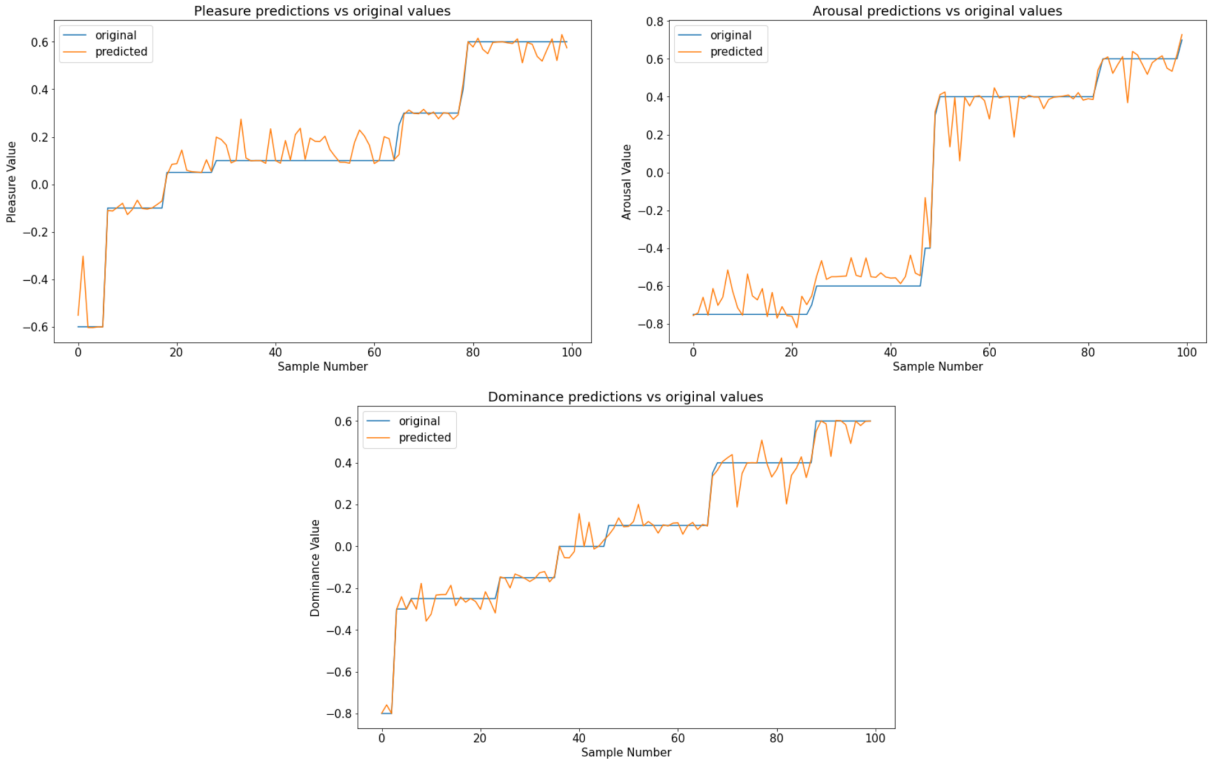


Figure 5.4: The predicted and true values of the Pleasure, Arousal and Dominance of 100 random sorted samples from the Test set.

It should be noted that mocap data consists of high-frequency "continuous" time series, in the sense that frames from the same animation are neighbours of each other and may present some form of sequential similarities. This same line of thought can be extended to our LMA Feature dataset. This may lead to an issue where, when data is randomly split as aforementioned, the train and test sets end up containing neighbouring LMA feature sets belonging to the same animation. This in turn could mean that the final results obtained over the test set could be good, solemnly because the models are overfitting to the train data, and the test set is comprised of similar features. To counteract this, and to make sure that the PAD regressors were not performing well simply due to dataset overfitting, we experimented with splitting animations directly into either the train and test set, rather than doing the aforementioned LMA Feature-level split. This means that the LMA Feature sets in the train set come from entirely different animations from those in the test set, effectively removing the "frame neighbour

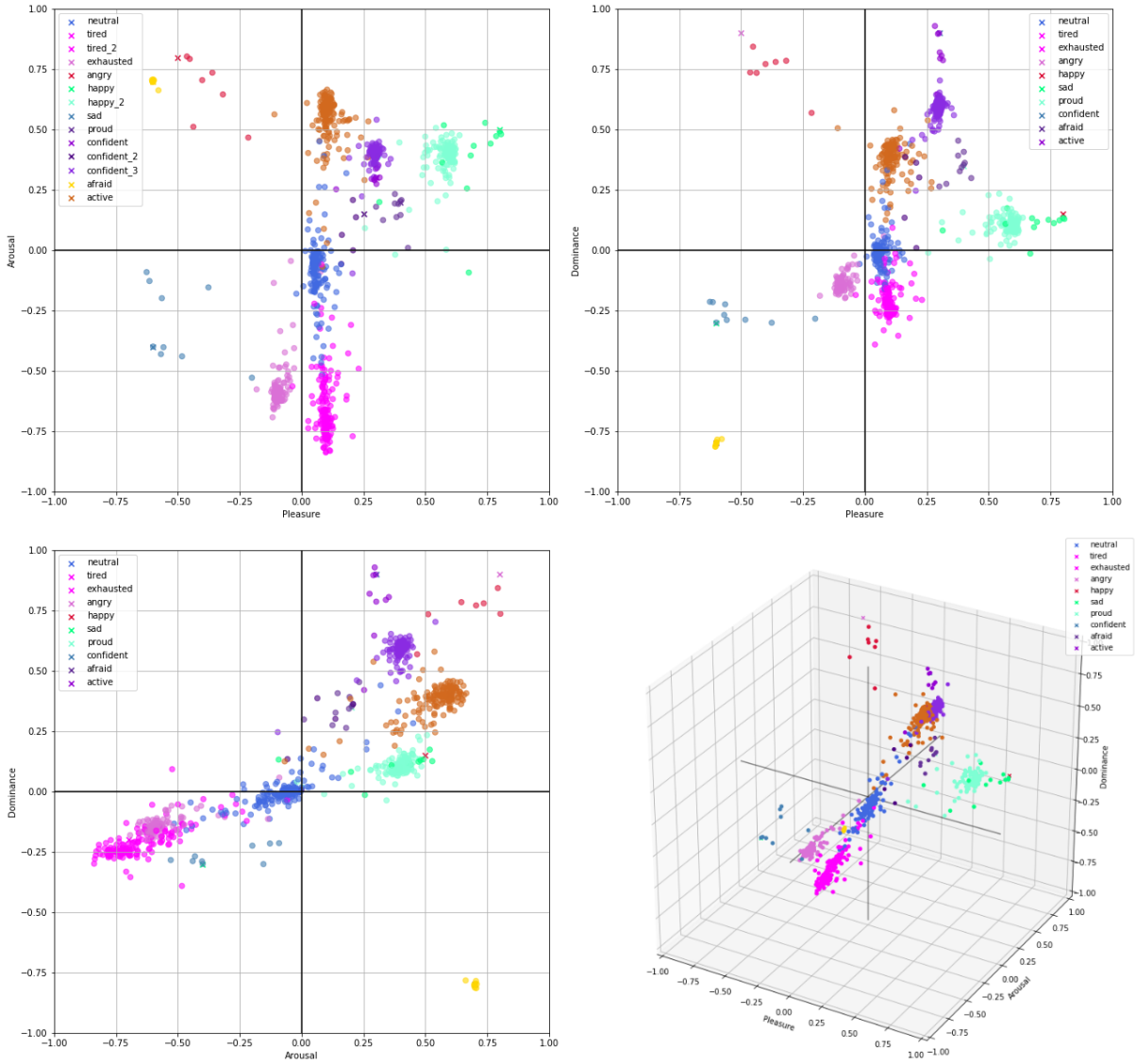


Figure 5.5: Prediction results of samples from our Test set. Each sample is coloured according to their real emotion and placed in the coordinate space according to its predicted emotional coordinate.

similarity" issue. The regressors were then trained and tested using these new dataset splits. Effectively, there was no apparent major performance hit, with the models achieving MAEs of approximately 0.02, 0.07 and 0.05 and MSEs of 0.01, 0.025 and 0.02 for the Pleasure, Arousal and Dominance coordinates, correspondingly. This proves that the initially presented results weren't caused by an overfitting to the dataset. The reason as to why the sequential similarity nature of frame data seems to be a non-issue may be due to the fact that our models are not being trained with frame data directly, but instead using LMA Features extracted every fifth frame. As such, two neighbouring LMA features represent a fairly high time difference making them different enough from each other. This may have become an issue had we used LMA features were to be extracted over a smaller frame-step, for example, every two frames.

5.4 PAD to LMA Mapping

The PAD to LMA ML models are responsible for converting the input PAD coordinates into a set of representative LMA feature values. This was implemented in two distinct manners - a "Direct" approach where the PAD coordinates are directly converted into LMA feature values using a set of regressors, and an "AutoEncoder" approach which adds an intermediate step, first mapping the input coordinates into Latent Features, and then decoding them into the LMA values. Other methodologies were also experimented with, notably the usage of Generative Adversarial Networks (GAN) [18] and Variational AutoEncoders (VAE) [30] for LMA feature generation, but the aforementioned techniques were the ones that netted the best PAD to LMA conversion results.

Each of the ML models, regardless of methodology, were trained with the same dataset. Rather than using the *Bandai-Namco-Research Motion Dataset* [7] with the PAD coordinates as input and the LMA Feature sets as output, we instead built a new dataset using our LMA to PAD Gradient Tree Boosting regressors. First the dataset was standardized. Then we removed the PAD coordinates from the dataset and fed it to our LMA to PAD regressors, generating new predicted coordinates and adding them to the dataset. The idea behind this is that, although each animation in the dataset was labeled according to its emotion and corresponding emotional coordinates, not all motions display the same emotional intensity. For example, all "Sad" walks were labeled with the exact same PAD values - $(-0.6, -0.4, -0.3)$ - but in reality, some of them might present different intensities, like an even lower Pleasure value, or a slightly higher Arousal. By using our predicted coordinates rather than the original labels we hope to capture these nuances and introduce more varied data to train our PAD to LMA models with. It should be noted that whilst the final models were trained using this new dataset, we also experimented with using the original labels, but realized that it produced worse results for either approach.

5.4.1 Direct LMA Feature Generation

The *Direct* approach was our first attempt at mapping PAD coordinates into LMA Feature Sets. In this methodology we simply trained a set of 25 Gradient Tree Boosting regressors to map our 3 PAD coordinates into each of our LMA features. To generate new features the input PAD coordinates are fed to each regressor individually. The regressors then output their corresponding LMA feature. This process is described in Figure 5.6. The Gradient Tree Boosting regressors were built using XGBoost [12]. Our data was shuffled and split into a Train/Validation and Test set, each containing 80% (62841) and 20% (15710) samples accordingly. The optimal hyper parameters for each regressor were found using Random Search 10-Fold Cross Validation [8] in a manner similar to the one used to find the optimal hyper parameters for the LMA to PAD regressors.

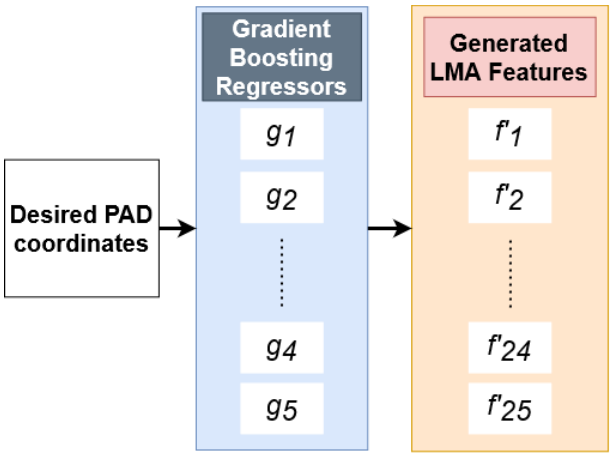


Figure 5.6: An overview of the Direct approach.

After training, the models' performance was evaluated by computing the MAE and MSE of each regressor over the Test set. We noted that for a majority of LMA Features, the reported MAE values were smaller than 0.1. The LMA Features which performed the worst were "Max Hand Distance" and "Max Stride Length", managing an absolute error of 0.17 and 0.15 accordingly. In terms of MSE, no regressor presented an error over 0.03. Additionally, we also used our LMA to PAD to convert the generated LMA Features back into PAD coordinates in order to evaluate how truly representative the generated features were of their intended emotional coordinates. Doing so and comparing the predicted PAD coordinates of the generated features with the original PAD coordinates that were used to generate them we managed to achieve an MAE under 0.20 for both Pleasure (0.20) and Dominance (0.18) coordinates and 0.30 for the Arousal. All in all, the features generated using this approach present some quality. Nevertheless we hoped to achieve better results than this, and as such the second approach, based around AutoEncoders, was created.

5.4.2 AutoEncoder LMA Feature Generation

The second approach we implemented was based around the usage of AutoEncoders. The idea was that the problem of mapping PAD coordinates into LMA presented a fair share of complexity due to the fact that we're trying to infer 25 different outputs from just 3 inputs. As such, using an AutoEncoder to reduce the output dimensionality would reduce the overall complexity of the problem and improve the PAD-LMA mapping performance [63, 64].

The entire process of going from PAD features into Latent Space features and how it then decodes these features into our usable LMA Feature set is shown in Figure 5.7. An *Autoencoder* was used to convert the 25 LMA Feature space into a 5 dimensional Latent Feature space - l_1, l_2, l_3, l_4, l_5 - and vice-versa. We then trained a set of 5 Gradient Tree Boosting regressors to convert our PAD coordinates into each of these Latent Features. To generate new features we first convert the PAD coordinates into our Latent Features. We then decode these Latent Features into a generated set of LMA Feature values representing the desired emotional coordinates.

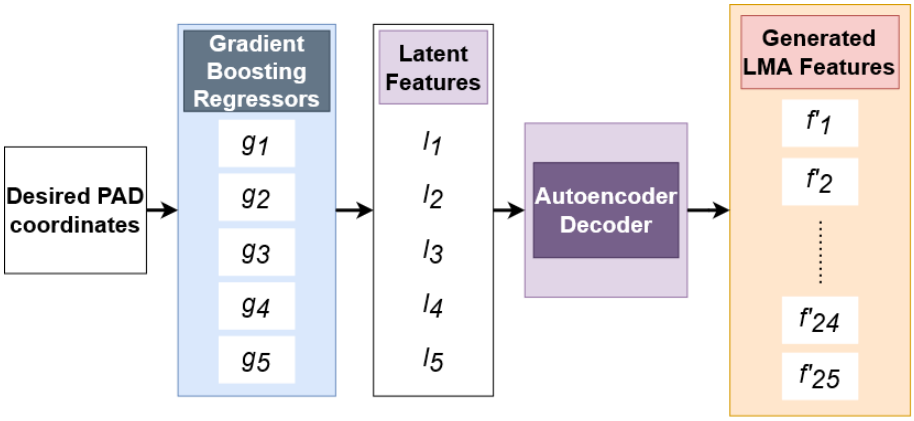


Figure 5.7: An overview of the AutoEncoder approach.

The reason behind using an additional set of regressors to go from PAD into our AutoEncoder's Latent Space comes from the fact that our goal is to convert PAD coordinates into LMA features, representative of these coordinates. AutoEncoders, however, are specialized networks whose input and output is the same. This means that the autoencoder converts LMA features into a Latent Representation, and then back into LMA, meaning that we had to have some form of mapping the PAD coordinates into the Latent Space. We attempted with forcing our Latent Space to be 3 dimensional and a one to one representation of the PAD space but there is little to no way of directly influencing and manipulating the Encoder network's output to behave in this manner. Furthermore the Decoder's performance suffered from having such a small dimensional Latent Space. We also attempted to use Deep Neural Networks to map directly from PAD coordinates into LMA features, and even to map PAD into the Latent Space features generated by our Encoder, but results weren't as good.

We began by training our Autoencoder Neural Network, built using Keras [15], with the architecture illustrated in Figure 5.8. After training for 1024 epochs, we accomplished a mean absolute reconstruction error of 0.17 on our Test set. We then generated a new labeled dataset using our PAD coordinates as input and the latent features created by the Autoencoder as output. Using this new dataset we trained the five regressors built using XGBoost [12]. The regressors were trained in a manner similar to those for Emotional Classification and tuned using Random Search 10-Fold Cross Validation.

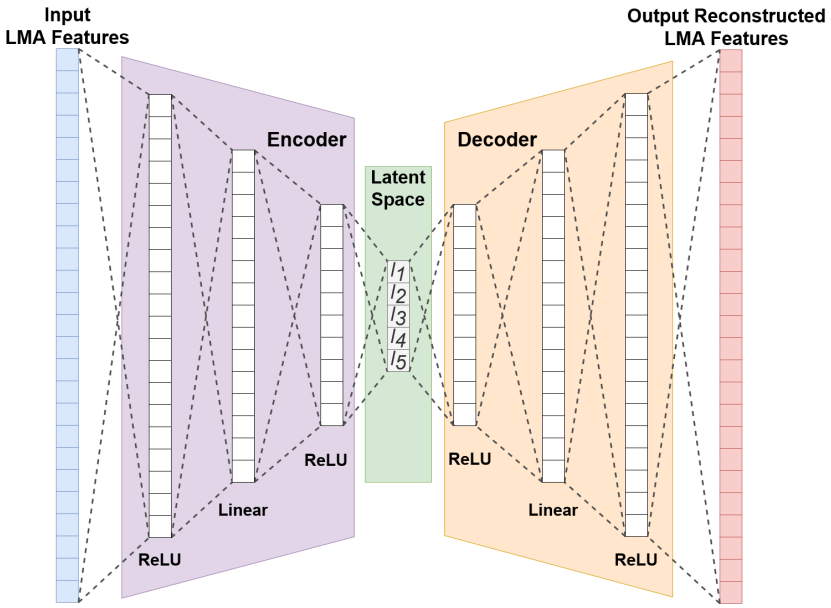


Figure 5.8: The Autoencoder architecture.

Through this AutoEncoder-based method we managed to achieve an overall mean absolute error of 0.19 between the predicted emotional coordinates of the generated LMA Feature set and the original ones, with Pleasure presenting an MAE of 0.19, Arousal 0.24 and Dominance 0.14. Comparing these error values to those from the Direct approach we can conclude that this more refined AutoEncoder approach manages to outperform it by a significant margin. Nevertheless both approaches were kept in the system with users being allowed to pick which LMA Feature Generation approach they wish to use when initializing the program. This was done in order to compare the quality of the generated motions of either approach through user testing as is further explained in Chapter 6.

5.5 Motion Synthesis

Given a new set of desired PAD coordinates we are then able to synthesize and apply motion changes to the character in real time. Pose changes are computed in a multi-threaded process to avoid interrupting or slowing down the running animation. Tweaking is done through a combination of LMA feature

generation and an algorithm based around a set of six heuristic rules used to output new desired core joint positions - Hips, Chest, Hands, Elbows, Feet and Neck.

For motion synthesis to take place the animation has to have looped at least once. The reason for this is due to the fact that our motion synthesis takes into consideration the baseline motion, as such, we need to store the animation's entire frame data. If the system was meant to only work with a Kinematic Controller reading directly from an animation file this would not have to be the case. since all data would be readily available from the start. Because we also wanted the system to be able to apply motion changes to a learnt policy-based controller in a physics-enabled environment we need to first run the animation at least once to store the necessary data.

The full set of Heuristic Rules, alongside their coefficients and associated LMA Features, can be seen in Table 5.4. Coefficient minimization is done right after the LMA Features corresponding to the input PAD coordinates are generated. Minimization is solved using Powell's method [44] provided by the SciPy [61] optimization library. The underlying idea behind each heuristic rule is that, by comparing the baseline animation's LMA features - which reflect the character's current emotion - with the generated ones - which correlate to the new desired emotion - we can then use coefficients to decide in which manner each joint should be altered. To exemplify, rule 1 - g_1 - changes the hips' height. The coefficient associated with rule 1 represents a comparison between relevant baseline and generated LMA Features. If this coefficient value is larger than one then that means that the current animation's associated LMA Features are smaller than their corresponding generated counterparts - chest-pelvis height, body volumes and so on - and as such, we want to increase them by increasing the hip's height and vice-versa. The new height is computed simply by adding the coefficient minus 1 to the current height. We subtract by one since if the coefficient is larger than one that will keep the value positive, hence increasing the height, otherwise the value becomes negative, lowering the current height. The coefficient's value also gets reduced in order to avoid generating extreme poses where the hips would be on the ground or too high up. A similar thought process was used to design the remaining rules.

5.6 Inverse Kinematics Solver

Whenever new joint position and rotations are synthesized, a new pose that respects these changes and that can be applied to the character has to be generated. To do this we utilize an Inverse Kinematics Solver module. This module runs a separate instance of PyBullet [17], without its graphical display enabled, containing a similar character model to the one in the main display loop. After generating the desired joint positions the Inverse Kinematics Solver is provided with the character's current pose so that it can update its own character to be synchronized with the baseline. We then provide it with the newly computed desired joint positions and rotations and use PyBullet's integrated Inverse Kinematics

Table 5.4: Motion Synthesis rules.

Rule	Coefficients
g1(c_1): Modifies the hips height **r'x** = rx **r'y** = ry + (c1 - 1.0) * 0.08 **r'z** = rz Where **r** is the current pelvis position and **r'** is the new desired pelvis position	**c1**: f_9, f_{11}, $f12$, f_{15}

g2(c_2): Modifies the chest position
 	
if c2 >1.0:
w = 0.025
else:
w = 0.1
 	
n'x = nx - (c2 - 1.0) * w
n'y = ny + (c2 - 1.0) * w
n'z = nz
 	
Where **n** is the current chest position
and **n'** is the new desired chest position
	c2: f_9, f_{11}, f_{13}

g3($c_{3.1}, c_{3.2}$): Modifies the hand's positions **l'x** = lx + dl-p x * (c3.1 - 1.0) * 0.5 **l'y** = ly + dl-p y * (c3.1 - 1.0) * 0.5 **l'z** = lz + dl-p z * (c3.1 - 1.0) * 0.5 **r'x** = rx + dr-p x * (c3.1 - 1.0) * 0.5 **r'y** = ry + dr-p y * (c3.1 - 1.0) * 0.5 **r'z** = rz + dr-p z * (c3.1 - 1.0) * 0.5 **l'x** = l'x - dl-h x * (c3.2 - 1.0) * 0.5 **l'y** = l'y - dl-h y * (c3.2 - 1.0) * 0.5 **l'z** = l'z - dl-h z * (c3.2 - 1.0) * 0.5 **r'x** = r'x - dr-h x * (c3.2 - 1.0) * 0.5 **r'y** = r'y - dr-h y * (c3.2 - 1.0) * 0.5 **r'z** = r'z - dr-h z * (c3.2 - 1.0) * 0.5 Where: - **l** and **r** are the current left/right hand positions, - **l'** and **r'** are the desired left/right hand positions, - **dl-p** and **dr-p** are unit vectors going from the current left/right hand positions to the pelvis - **dl-h** and **r-h** are unit vectors going from the current left/right hand positions to the chest	**c3.1**: f_1, f_2, f_3, f_{11}, f_{13}, f_{14} **c3.2**: f_1, f_5, f_6, f_{11}, f_{13}, f_{14}

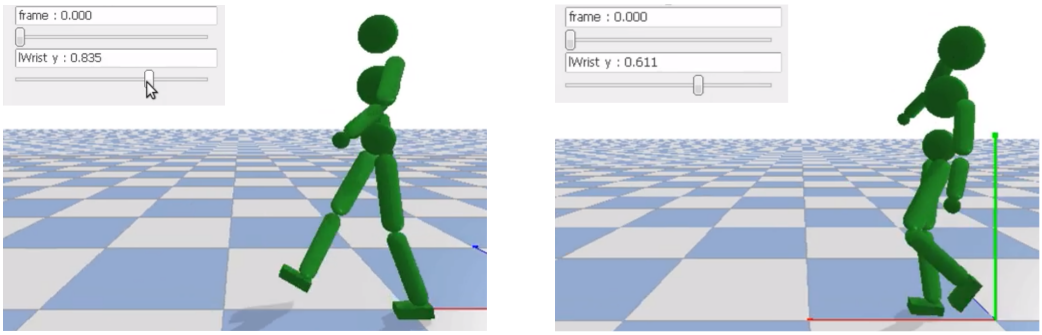
Rule	Coefficients

g4(c_4): Modifies the elbow positions
 	
le'x = lex + dle-p x * (c4 - 1.0) * 0.5
le'y = ley + dle-p y * (c4 - 1.0) * 0.5
le'z = lez + dle-p z * (c4 - 1.0) * 0.5
 	
re'x = rex + dre-p x * (c4 - 1.0) * 0.5
re'y = rey + dre-p y * (c4 - 1.0) * 0.5
re'z = rez + dre-p z * (c4 - 1.0) * 0.5
 	
if $\|$le'z - lez$\|$ $<$ 0.15:
le'z = lez - 0.15
if $\|$re'z - rez$\|$ $<$ 0.15:
re'z = rez + 0.15
 	
Where:
- **le** and **re** are the current left/right elbow positions,
- **le'** and **re'** are the desired left/right elbow positions
- **dle-p** and **dre-p** are unit vectors going from the current left/right elbow positions to the pelvis
	c4: f_7, f_8, f_{11}, f_{13}

g5(c_5): Modifies the feet positions
 	
if c5 >1.5:
c5 = 1.5
if c5 <0.5:
c5 = 0.5
 	
lf'x = lex + dlf-rf x * (c5 - 1.0) * 0.2
lf'y = ley
lf'z = lez + dlf-rf z * (c5 - 1.0) * 0.2
 	
rf'x = rex + drf-lf x * (c5 - 1.0) * 0.2
rf'y = rey
rf'z = rez + drf-lf z * (c5 - 1.0) * 0.2
 	
Where:
- **lf** and **rf** are the current left/right foot positions,
- **lf'** and **rf'** are the desired left/right foot positions
- **dlf-rf** and **drf-lf** are unit vectors going from the current left foot to right foot positions and vice versa
	c5: f_4, f_{11}, f_{12}, f_{15}

g6(c_6): Modifies the neck tilt **nr'x** = nrx **nr'y** = nry + (c6 - 1.0) * 1.5 **nr'z** = nrz Where: - **nr** is the current neck rotation in Euler - **nr'** is the desired neck rotation in Euler	**c6**: f_{10}, f_{11}, f_{13}

functionalities to output a new synthesized pose, which can then be applied to the main character. This new pose tries to get the core joints as close as possible to their desired synthesized counterparts, while still respecting the character's body restraints to avoid unnatural postures. Figure 5.9 provides a demonstration of the usage of Inverse Kinematics to change the position of a character's joint. In this particular example we tasked the Inverse Kinematics Solver with lowering the height of the character's left wrist. The output pose had the character shift its weight and curve to the left in order to naturally achieve the desired left wrist position.



(a) A character's baseline pose

(b) A character's altered pose

Figure 5.9: Showcase of the usage of Inverse Kinematics to alter a character's joint (left wrist) position.

There were a few problems that had to be solved for the Inverse Kinematics to work. Firstly, it should be mentioned that the humanoid model that SpaceTime Bounds [39] and DeepMimic [43] use, structured as a Unified Robot Description Format (URDF) file, defines its joints as spherical joints. These joints possess 3 Degrees of Freedom, meaning they are allowed to rotate in all 3 axis and their positions are specified as quaternions. The problem is that, while PyBullet is compatible with these joints, its Inverse Kinematics solver function is not. As such we had to create a copy of our main humanoid model where we replaced each spherical joint with a set of three linear ones. Each of these joints is only allowed to rotate in a single axis and their positions are represented as an Euler angle. We now had the issue that our main character's model was inherently different from our our Inverse Kinematics one. Because of this, all joint position specifications in the extracted frame data came in the form of quaternions, which had to be converted into a set of 3 Euler angles - one for each axis. These angles could then be applied directly to the Inverse Kinematics character's linear joints. The order in which these rotations are applied also matters [19]. For our set of linear joints to perfectly correspond to its spherical joint counterpart we had to apply rotations to the Z (Yaw), Y (Pitch) and X (Roll) axis, in this order.

5.7 Support Modules

Aside from the main modules responsible for performing Emotional Classification and Motion Synthesis our system counts with additional modules that are used to support and showcase the functioning of the core components.

5.7.1 BVH To Deepmimic Converter

DeepMimic [43] utilizes a unique motion file structure. The issue with this is that most available mocap data files are stored in the BVH standard and as such are incompatible with the DeepMimic system out of the box. Our system was built over Spacetime Bounds [39], which in turn was built on top of DeepMimic, and, as such, suffers from the same problem. DeepMimic's mocap format are stored in JSON files as can be seen in Listing 5.2. The first argument - "Loop" - can either be "none" or "wrap" depending or whether or not the animation loops or stops after its done playing. The "Frames" list contains all each of the keyframes data comprised of the character's joint position and rotations and the duration of each keyframe.

Listing 5.2: The DeepMimic motion file format.

```
{
    "Loop": "none" or "wrap",
    "Frames": [
        [
            duration of frame in seconds (1D),
            root position (3D),
            root rotation (4D),
            chest rotation (4D),
            neck rotation (4D),
            right hip rotation (4D),
            right knee rotation (1D),
            right ankle rotation (4D),
            right shoulder rotation (4D),
            right elbow rotation (1D),
            left hip rotation (4D),
            left knee rotation (1D),
            left ankle rotation (4D),
            left shoulder rotation (4D),
            left elbow rotation (1D)
        ],
    ]
}
```

In order to utilize the Bandai-Namco-Research Motion Dataset [7] we had to convert each of its BVH mocap files into DeepMimic's motion file format. To do so we built our BVH To Deepmimic Converter module by adapting the preexisting Bvh2Deepmimic library [36] to work with our dataset's character skeleton and structure.

5.7.2 User Interface

The User Interfaced built to showcase the EEMC System's capabilities consists of two components. The first is a display window, which contains a virtual character performing the baseline animation the

system was started with. The second is the main GUI, which shows the current output of the Emotional Classification and allows for the input of new desired PAD coordinates for Motion Synthesis. The GUI also displays System State information regarding whether the animation has looped or ended, whether the emotional classification is still ongoing or finished, and whether the system is synthesizing new motion changes or not.

The Motion Display window was built using the PyBullet [17] graphical and physics engine. Upon starting our system, this window is automatically open. After the ML models have finished loading, a character is then shown performing the provided baseline animation in a virtual environment. This environment is either Kinematic or Physics-Enabled depending on whether the EEMC system was started with a mocap file or a learned policy character controller. An additional reference character can also be spawned. This character is impervious to the system's motion changes, always displaying the baseline motion using a Kinematic character controller. As shown in 5.10, the reference character is always coloured in orange, whilst the main one is always painted green. This was done to make it easy to distinguish between the two characters. Originally this reference character was only used to showcase the similarities between a Policy-Based character controller and the original motion it learned from. Its usage was then extended since it also proved useful at highlighting the impact of our Emotional Motion Synthesizer, comparatively to the baseline animation.

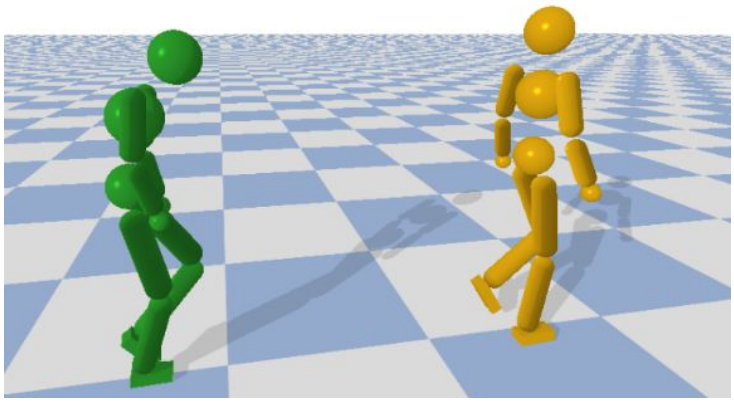


Figure 5.10: A main (left) and reference (right) character's performing an animation in the Main Display window.

The Motion Controller & Emotion Classifier GUI was developed using TKinter [38] and is used to bridge users to the underlying EEMC framework. All of this window's interactions and updates are handled by a GUI Manager, which connects to the rest of the system both to collect Emotional Prediction's results, but also to inject new PAD coordinates into the Motion Synthesis subsystem. Whenever new emotional coordinates are output by the Emotion Classification subsystem, the manager captures them and shows them to the user, displaying each predicted coordinate individually, alongside the closest discrete emotion. To find the closest emotion we compute the SSE between the PAD coordinates of a preset list of emotions and the output predicted coordinates. The emotion with the smallest error is

shown and colour coded in accordance with how small this error was, as shown in Figure 5.11. The closest emotion's name is coloured green for errors under 0.03, yellow for under 0.05 yellow and red otherwise. The preset list of emotions includes the emotions "Neutral", "Tired", "Exhausted", "Angry", "Happy", "Sad", "Proud", "Confident", "Afraid" and "Active", covering a wide portion of the PAD model.

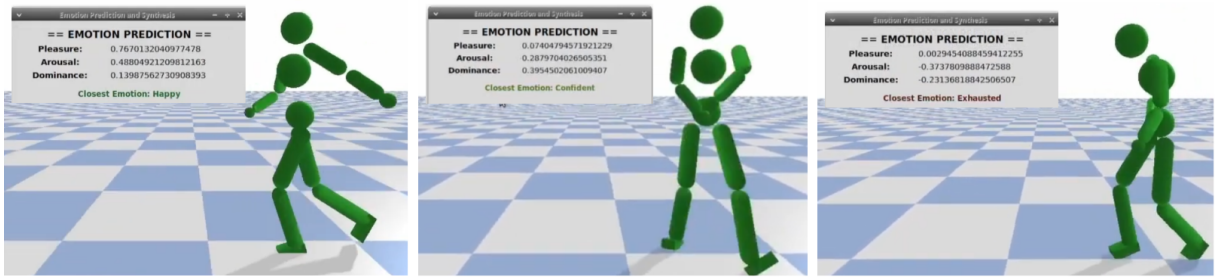
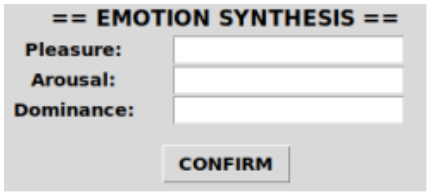
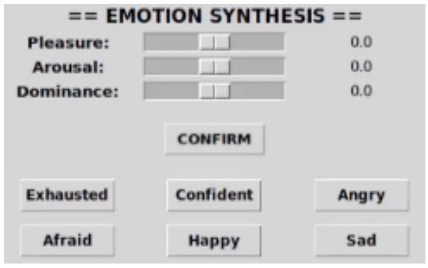


Figure 5.11: The closest discrete emotion, colour coded according to how close it is to the predicted coordinates.

Motion Synthesis can be triggered through the GUI using its available sliders. For ease of use we also included a set of predefined emotions. Clicking any of these simply inputs the emotion's coordinates into the corresponding coordinate sliders. An initial implementation had users specify coordinates by inputting the desired values into text fields. These fields were then replaced with sliders to make the interaction easier and to ensure only valid values, between -1.0 and 1.0 are inserted into the system. The old and new interfaces for coordinate specification are shown in Figure 5.12.



(a) The old interface where coordinates were specified via text



(b) The new interface where coordinates are specified using sliders

Figure 5.12: The old and new way of specifying emotional coordinates for motion synthesis.

5.7.3 Motion Learning

The Motion Learning module is capable of generating a character controller in the form of a policy learnt using the Spacetime Bounds [39] system. Unlike the other modules, this one is entirely independent of the core system and is meant to be used to generate animations. These animations can then be provided to the rest of the system to be used as a baseline motion for emotional tweaking.

Animations generated using this module come in the form of policies which take into account the current state of the character, time step and reference motion in order to generate the character's desired joint rotations. Should the system be initialized with one of these learnt policies, characters will be placed into a physics-enabled environment. In this instance, two characters are shown to the user, as seen in Figure 5.13, one using a Kinematic controller simply showcasing the reference motion the policy was trained with, and the other entirely controlled by the policy and subject to the environment's external physics forces, such as gravity.

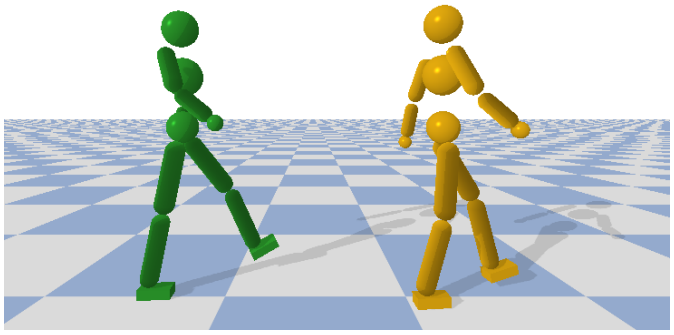


Figure 5.13: Two characters being displayed with one (left) being controlled by a policy that learned to mimic the reference animation showcased by the other (right) in a physics-enabled environment.

This module was entirely built over Spacetime Bounds [39] with no further modifications. As such the policy controller consists of a neural network as explained in Chapter 3. In order to train a new policy mocap data with the desired animation has to be provided to the module and certain training parameters have to be specified through a JSON file containing all desired arguments. Table 5.5 contains all possible training parameters and a description of what they do. The reference mocap file, which the policy will aim to mimic, is specified through the *task* and *model* arguments. These reference files have to be in a Deepmimic friendly format [36] and follow the naming scheme "*model_task.txt*". For example for a walking animation using the default humanoid model, the reference mocap should be named "*humanoid3d_walk.txt*". Listing 5.3 showcases an example of a complete arguments file used to train a policy to control the *humanoid3d* character model to perform the reference running animation.

The Spacetime Bounds system has the capability of booting up several different types of environments aside from the regular "Spacetime" one, each with varying arguments [39]. For example the "Energy Style" environment imposes additional constraints on the amount of energy the character is allowed to display in order to audit the generated animation's outcome. We, however, focused our efforts solemnly towards the default "Spacetime" environment since additional tweaking of the animation was to be performed in real time using our main EEMC system. Nevertheless, our system is able to function with any type of character controller generated using the Spacetime Bounds system, regardless of environment type.

Table 5.5: All possible training arguments for the Motion Learning module.

Argument	Type	Description
task	env_args	The name of the motion task (e.g Walk, Dance, Run).
model	env_args	The name of the virtual character's model.
engine	env_args	The name of the physics engine used during training.
self_collision	env_args	Whether or not the body can collide with itself.
enable_draw	env_args	Whether or not to provide a graphical display during training.
heading_vec	env_args	The XYZ vector specifying which axis the character is considered "front".
record_contacts	env_args	Whether or not to record all registered contacts.
record_torques	env_args	Whether or not to record all registered torques.
use_global_root_ori	env_args	Whether or not to use the global root orientation.
rel_endeffector	env_args	Whether or not to compare the policy controlled character's end effector positions and orientations to the reference character's when checking by how much the generated and reference pose's differ.
rel_root_pos	env_args	Whether or not to compare the policy controlled character's root height to the reference character's when checking by how much the generated and reference pose's differ.
rel_root_ori	env_args	Whether or not to compare the policy controlled character's local root orientation to the reference character's when checking by how much the generated and reference pose's differ.
use_spacetime_bounds	env_args	Whether or not to use Spacetime Bounds.
bound	env_args	File specifying the Spacetime Bounds to be used.
contact	env_args	Specifies which joints are allowed to make contact with the ground.

Argument	Type	Description
id	meta	Name of the training.
workers	meta	Number of workers used to sample data. Corresponds to the number of environments that will be initialized and used to train the policy in parallel.
ckpt	meta	Path to a previously paused training. Used to resumetraining rather than starting from scratch.

iter_num	train_args	Number of iterations the training should run for.
gamma	train_args	The discount factor. The closer the value is to zero the more immediate rewards are prioritized over possible future rewards.
use_importance_sampling	train_args	Whether or not to use Importance Sampling.
num_segments	train_args	Number of samples in each sample pool used in Importance Sampling.

noise	model_args	The standard deviation value of the actor policy's network's noise layer in the Actor-Critic architecture.
with_ffc	model_args	Whether or not to use the FeedForward Controller portion of the Policy network architecture. This controller contains the joint angles from the reference motion stored and linearly interpolates them during run time according to the current animation timestep.

Listing 5.3: Example of a JSON file containing training arguments to learn to mimic a running animation using spacetime bounds with the default humanoid character.

```
{
  "env_name": "spacetime",
  "env_args":
  {
    "task": "run",
    "model": "humanoid3d",
    "engine": "pybullet",
    "contact": "walk",
    "self_collision": true,
    "enable_draw": false,
    "record_contact": false,
    "record_torques": false,

    "use_global_root_ori": true,
    "heading_vec": [1, 0, 0],
    "use_spacetime_bounds": true,
    "bound": "data/bounds/default.txt",
    "rel_root_pos": false,
    "rel_root_ori": false,
    "rel_endeffector": true
  },

  "model_args":
  {
    "noise": 0.1,
    "with_ffc": true
  },

  "train_args":
  {
    "iter_num": 6000,
    "gamma": 0.95,
    "use_importance_sampling": true,
    "num_segments": 10,
  }
}
```

After a policy training is complete we can use the generated character controller with the rest of the system by specifying a new arguments file. This file simply contains the name of the task in order to load the correct reference animation and the path to the generated policy, stored in the form of a compressed *tar* file. Listing 5.4 contains an example of such an arguments file used to load the policy stored in the file "*walk_training_1*" trained to mimic the reference mocap file containing the "*Walk*" motion.

Listing 5.4: Example of a JSON file containing arguments to start our system with a learnt physics-enabled policy-based character controller.

```
{
  "env_name": "spacetime",
  "env_args":
  {
    "task": "walk"
  },
  "ckpt": "data/policies/bound_scale/walk_training_1tar"
}
```

6

Results & Analysis

Contents

This chapter goes over the finalized system, showcasing how users can interact with it and what the final outcome of the EEMC is. We also present the results of the user tests that were conducted to evaluate the synthesized animation's quality.

6.1 Final System

The EEMC framework was built in Python 3.8, using PyBullet [17] as the underlying engine. All ML models were trained offline in a dedicated external server with two NVidia GeForce RTX 2080 Ti GPUs, 32GB RAM and a six-core AMD Ryzen 5 2600X 3.6GHz CPU. Emotional Classification and Motion Synthesis is done in multithreaded processes and takes, on average, less than 3 seconds to execute and apply, running in real time. The system's code, results and other resources are publicly available[1]. A video was also created and made available showcasing an overview of the system's capabilities[2]. The framework can be initialized with several configurable parameters as shown in Table 6.1. These parameters can be used to alter some of the system's behaviors such as which motion synthesis methodology is used or whether an additional reference character should be spawned.

Table 6.1: The EEMC system's boot arguments.

Parameter	Type	Description
mocap	string	The path to the MoCap file or Learned Policy describing the baseline animation that will be displayed.
ms	string Options:["ae", "direct"]	Specifies the type of Motion Synthesis methodology to be used. By default the AutoEncoder approach is used.
record_lma	string	File name LMA Features should be recorded onto. If no name is specified then features won't be stored in a file.
record_mocap	string	File name frame data should be recorded onto. If no name is specified then this data won't be stored.
show_reference	boolean	Whether or not to spawn an additional reference character.

Aside from allowing for the real time Emotional Discernment and Emotionally Expressive Motion Synthesis, the finalized work also includes complementary GUI and Motion Learning components to showcase the system's capabilities and allow users to easily interact with the underlying framework without the need of any additional domain knowledge. Users can provide the system with a baseline motion either via DeepMimic-friendly mocap files or using a learned character controller policy. The GUI and motion display window are shown in Figure 6.1. The GUI shows the current results of the Emotional Classification by displaying the predicted PAD coordinates alongside whichever discrete emotion is closest to the predicted coordinates. To specify new desired PAD coordinates users can freely tweak the corresponding sliders or select one of the available presets and hit the "Confirm" button.

[1] https://heroufenix.github.io/expressive_animations_web/
[2] https://youtu.be/QOfVkoZa5HM

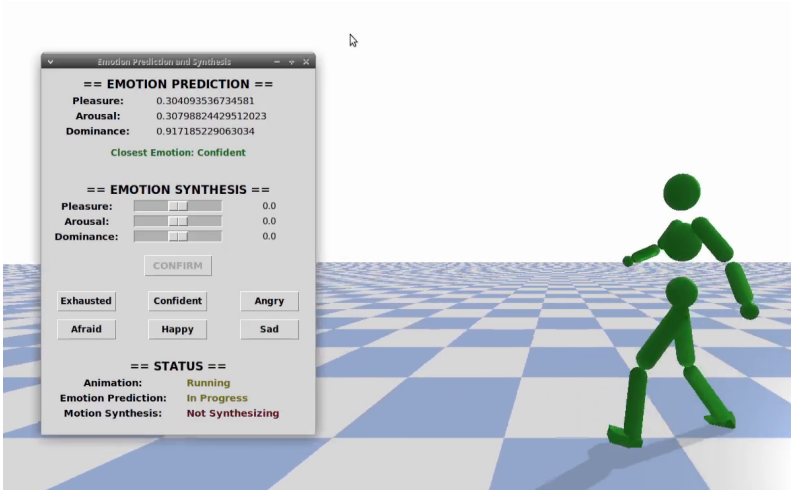


Figure 6.1: The finalized GUI used to showcase the system.

Triggering the Motion Synthesis module alters the character's motion in real time. Figure 6.2 showcases 4 generated motions synthesized from the same baseline animation. The "Confident" character, for example, highly elevates their shoulders, widens their upper body volume and exposes their neck, while the "Afraid" character raises their arms to protect their torso and slumps down, reducing its body volume. A playlist of clips containing motions generated using the EEMC framework has been published online[3]. Motion Synthesis can be performed using LMA Features generated either through the Direct or AutoEncoder approach. Both methodologies work in real time and are done in a parallel process so there is no performance impact on the motion display. The time it takes the models to load during the system's boot up does differ. The Direct approach takes much longer to load since it uses 25 different Gradient Tree Boosting regressors, as opposed to the 5 of the AutoEncoder approach.

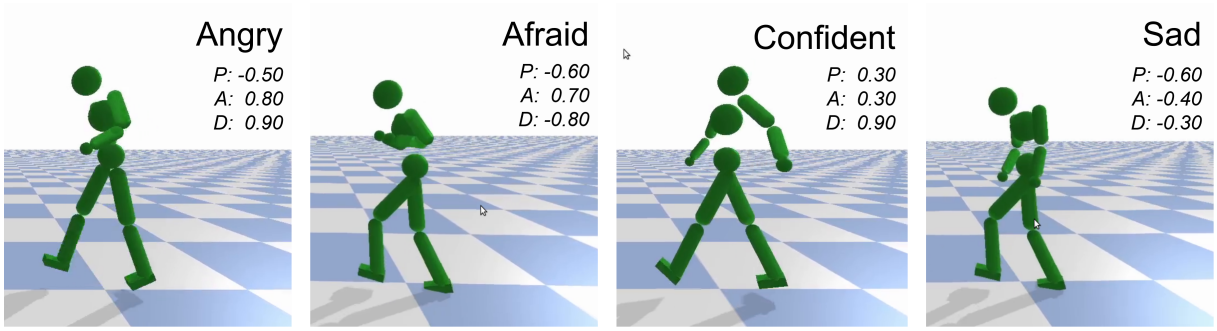


Figure 6.2: Four motions synthesized using the same baseline motion and 4 different desired emotions.

Our system works best when applied to a neutral baseline movement but it nevertheless works with different base emotions. Both emotional classification and synthesis were trained and designed for locomotion-type motions. Whilst they can still be applied to other types of animations without additional

[3] https://youtube.com/playlist?list=PLBchdrsdyMe_y7oUAzumcPc3P5ft3SMi4

changes, the results won't be as consistent. Motion Synthesis seems to suffer the most from this in the quality of the generated motions, mostly due to the fact that our heuristic rules were purposefully tweaked for locomotion type animations. Emotional Classification also suffers in the accuracy of its predictions but still mostly manages to predict the motion's correct octant within the PAD 3D space.

6.2 User Testing & Validation

User tests were conducted in order to evaluate the performance of the Emotionally Expressive Motion Synthesis. We wanted to evaluate how the generated motions would compare against the ones from the Bandai-Research Motion Dataset [7]. These reference motions were recorded with professional paid actors performing motions in very specific emotional styles. Our aim was to infer the synthesized motions' quality by checking if there were any major statistical differences between the answers users gave when presented with reference animation clips versus the generated ones. Furthermore we also wanted to compare the two LMA Feature generation approaches to understand if one outperformed the other in terms of emotional expression quality as perceived by users.

A set of video clips was created using our motion generation over both a Kinematic and a Policy-Controlled physics enabled character and generating features with either the Direct or AutoEncoder approach. Motions were generated to convey a subset of the Bandai-Research Motion Dataset's [7] emotions covering a wide spectrum of the PAD model - "Sad", "Confident", "Tired", "Afraid", "Angry" and "Happy". The overall intent was to check whether the generated motions managed to convey their intended emotions as well as the reference mocap. To infer these, two distinct tests were conducted with 40 anonymous, paid participants [4] each. The tests consisted of online forms containing the aforementioned recorded animation clips mixed together and sorted randomly. Clips had their names redacted and participants were never informed or able to tell the type of clip - reference mocap or synthesized, kinematic or policy-based, generated using the Direct or AutoEncoder approach. The retrieved answers were then compiled and studied through statistical tests performed using IBM SPSS [27].

6.2.1 Emotion Identification Task

The first set of participants were asked to view each clip and select which emotion they thought the character was trying to express from a given list. The goal was to provide an initial insight towards how easy the generated motions' emotions were to identify. As such, a clip's performance is better the higher the percentage of participants that manage to correctly guess the character's intended emotion. For example, if a character is attempting to convey the feeling "Angry", the more participants that answer with

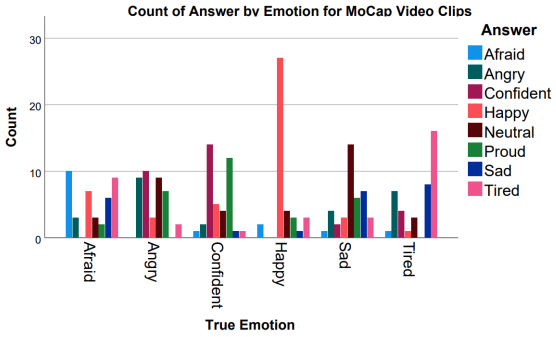
[4]This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference FCT: UIDB/50021/2020 and Project SLICE reference: PTDC/CCICOM/30787/2017

“Angry”, the better that clip performed. The quality of each generation technique was then ascertained by comparing its performance against each other and, more importantly, against the reference mocap.

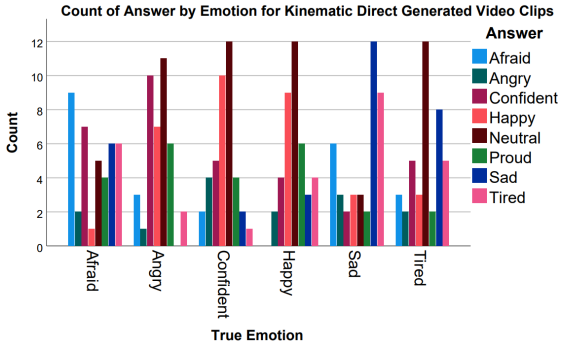
The test’s results were gathered in the clustered bar charts shown in Figure 6.3. Looking at the reference mocap we can see that most participants managed to correctly identify the emotions “Afraid”, “Confident”, “Happy” and “Tired”, although not by a vast margin in most cases. “Happy” seemed to be the most easily identifiable emotion with an overwhelming majority of participants being able to correctly classify it. Moving on to our synthesized motions, using the AutoEncoder approach applied to a Kinematic character yielded results that were as good, and in some cases like for the emotion “Sad”, slightly better than the reference mocap. These clips managed to have most participants select the correct emotion in all cases but “Tired” which was the second most selected answer in its category, nearly tying with “Sad” for first place. Even when paired with a Policy-based physics-enabled character, motions generated with this method presented decent results. The best performing emotions became “Confident”, “Sad” and “Tired”. The other emotions were usually the second or third most guessed in their categories. In general, motions generated with the AutoEncoder approach performed well comparatively to the reference mocaps, regardless of the type of character controller being used. The Direct generation method didn’t perform as well, however. When applied to a Kinematic character, participants tended to believe most clips aside from “Afraid” and “Sad” were “Neutral”. The same happened when using a policy controlled character, although with slight improvements to the “Tired” emotion’s performance. This seems to indicate that the motions generated using this methodology are too generic and lack the expressivity of the AutoEncoder approach, causing users to believe that the character is in a “Neutral” style.

6.2.2 Primed Emotion Agreement Task

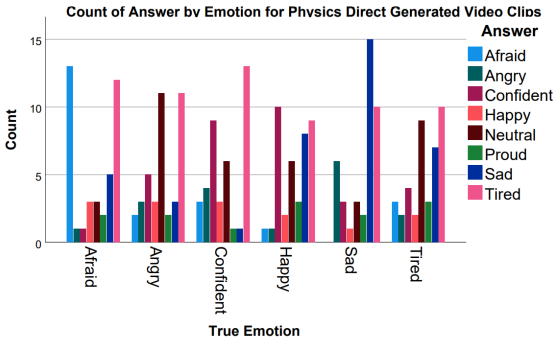
Certain emotions have intrinsic ambiguity when lacking context [47], which might have influenced the answers given in the first test, explaining some of the obtained results. “Tired” and “Sad”, for example, are both very low energy emotions and the way they get conveyed is somewhat similar, especially in the reference mocaps. As such, when presented with just the clip with no further context or information about what the character’s intentions are, it becomes easy for users to mix these emotions. To counteract this, a second test was conducted where participants were explicitly told which emotion the character was trying to express. They were then asked to rate how much they agreed that the character was in fact expressing said emotion. Participants could answer using a Likert scale from 1 (Completely Disagree) to 5 (Completely Agree). The overall goal of this test was to infer how accurately each clip managed to convey their intended emotion, as perceived by the participants. As such, clips perform the better the more participants agree that the presented emotion accurately matches the one showcased by the character in its motion.



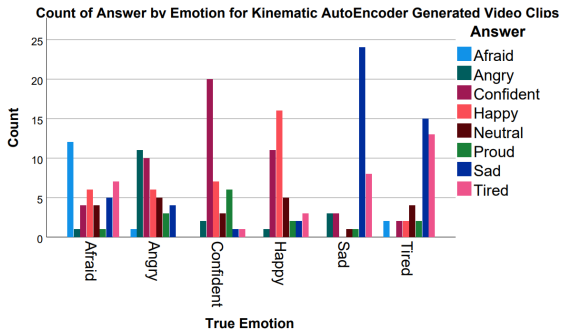
(a) Baseline Mocap performance.



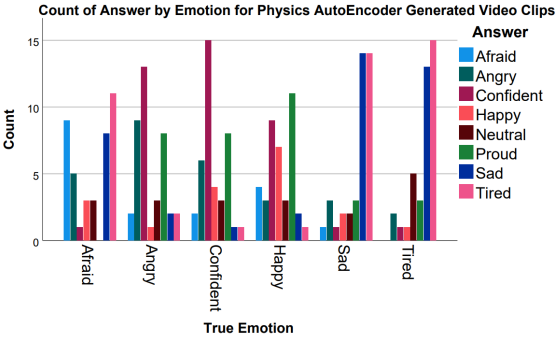
(b) Direct approach on a Kinematic Controller performance.



(c) Direct approach on a Policy Controller performance.



(d) AutoEncoder approach on a Kinematic Controller performance.



(e) AutoEncoder approach on a Policy Controller performance.

Figure 6.3: Clustered bar charts showing the count of answers compared to the correct emotion for each of our set of clips.

An initial Friedman test [56] was performed on the retrieved data in order to determine if there was any statistically significant difference between the answers given for the same emotion using different clip types - reference or generated animation, kinematic or policy-based character, AutoEncoder or Direct generation. The analysis conducted over each emotion showed that there were statistically significant differences between the responses given for each clip type for all emotions except for "Afraid" ($p = 0.493$). This means that participants provided similar answers for the "Afraid" emotion, regardless of whether the viewed clip presented the reference mocap, or had been generated using our system applied to either a Kinematic or Policy-based character. Every other emotion, however, reported, to a varying extent, a statistically significant difference amongst clip types. The full list of results is presented in Table 6.2.

Table 6.2: Reported Friedman Test Significance Levels per emotion.

Emotion	Significance Level (Asymp. Sig.)
Happy	<0.001
Afraid	0.493
Tired	0.002
Angry	0.003
Confident	<0.001
Sad	<0.001

A post hoc analysis using a Wilcoxon Signed Rank Test [66] was conducted over every emotion with a significance level under $p = 0.005$ on the Friedman test - "Happy", "Tired", "Angry", "Confident", "Sad". This was done to directly compare each clip type pair in order to more granularly identify the presence of statistically significant differences between clip type pairs. The results of this test can be found in Appendix B. The "Sad" emotion was the only one for which there was a significant difference between every generated motion type and the mocap ($p < 0.001$ for every generated-reference pair). For all other emotions, most generated motions proved to have a statistically significant difference compared to the reference, but there tended to be at least one generation method per emotion for which this didn't hold true. For example, on the "Tired" emotion there was a difference for all generated clips other than the Direct approach applied to a Kinematic character ($p = 0.472$), and for "Confident" the same happened but with the AutoEncoder approach applied to a Policy controlled character ($p = 0.835$).

After having determined the presence of statistically significant difference between the answers given for each clip type, we then moved on to analyzing the provided responses. Knowing that there was a difference between the answers given depending on clip type the goal was now to infer which type performed the best in our test. To do so the dispersion of answers per type of clip for each emotion were graphed into the boxplots shown in Figure 6.4. For the emotion "Happy" the baseline mocap outperformed all of our motion synthesis models, although they still reported decent results on all generation method-controller type pairings other than Direct on a Policy controlled character. This, however, was the only emotion where our generated motions didn't perform as well or better than the reference mo-

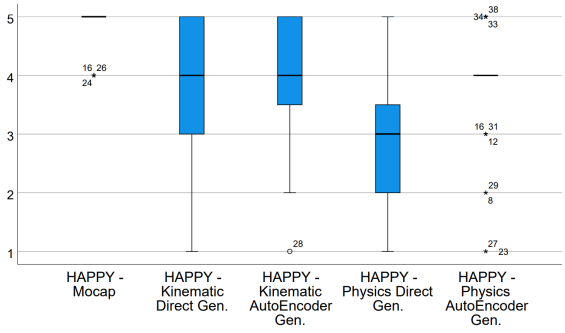
cap. We can see that for the “Sad” and “Tired” emotions, both types of generation actually outperform the reference mocap meaning that for these particular emotions, our generated motions are more easily identified as their corresponding emotions. For the “Angry” emotion the results were similar regardless of the clip being of a mocap or a generated motion. On the “Afraid” emotion we can see that the generated motions applied to the Policy-based characters performed slightly worse, but when applied to a Kinematic characters they still presented results equitable to the reference mocap. For “Confident” our AutoEncoder generated motions managed to perform slightly better than the reference but the Direct approach generated ones didn’t, performing a bit worse, especially when applied to a Kinematic character. Comparing our generation methods, the AutoEncoder approach seems to beat the Direct approach in all instances although by how much the latter approach gets outperformed varies between emotions.

6.2.3 Discussion

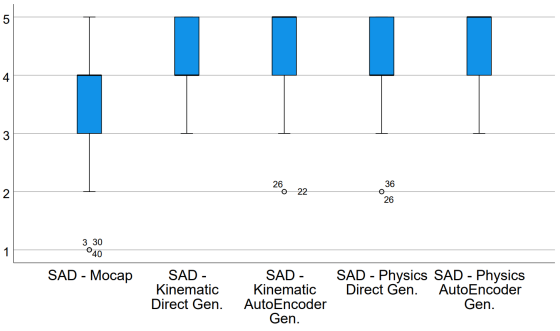
Looking at the results of both tests, participants, for the most part, managed to correctly identify and tended to agree with, the emotions that the generated motions were trying to convey. Moreover, certain emotions were more easily identified comparatively to the reference mocap. This showcases the efficacy of our system, as it proves that we can achieve results with similar emotional expressiveness to professional-grade mocap without the need and costs of recording several actors performing each of the desired emotions.

In terms of character controller type, both Kinematic and Policy-based character controllers seemed to present comparable performances. The obtained results didn’t seem to deviate much within generation method as there was never an instance where results drastically changed depending on the character controller’s type. This seems to indicate that the EEMC system can be effectively used regardless of controller type broadening its range of application to not only conventional mocap-based kinematic animations but also automatically generated policy-controlled learned motions. By comparing the test’s results obtained using either a Kinematic or Policy-based character controllers we managed to deduce that there

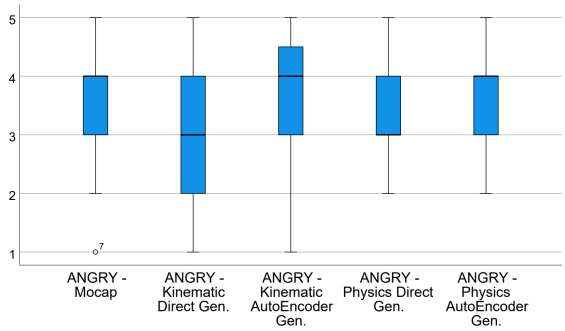
The AutoEncoder generation approach presented the best results out of the two implemented methodologies. Whilst the Direct approach still managed to achieve respectable results, it was always outperformed by the AutoEncoder generation and it seldom managed to beat the reference mocaps performance, usually doing as good as, or just slightly worse than it. This, paired with the fact that the AutoEncoder approach takes less time to load, speeding up the system’s initialization time, has us believe that this is the *de facto* better LMA Feature generation approach implemented, showcasing the efficacy of the usage of an AutoEncoder to reduce the complexity of the PAD to LMA mapping problem.



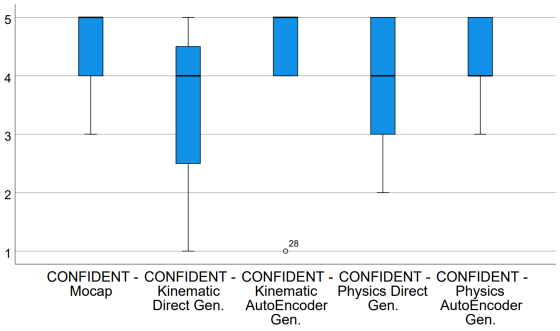
(a) Performance for the emotion “Happy”.



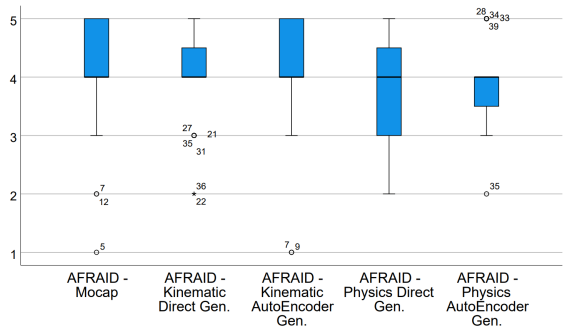
(b) Performance for the emotion “Sad”.



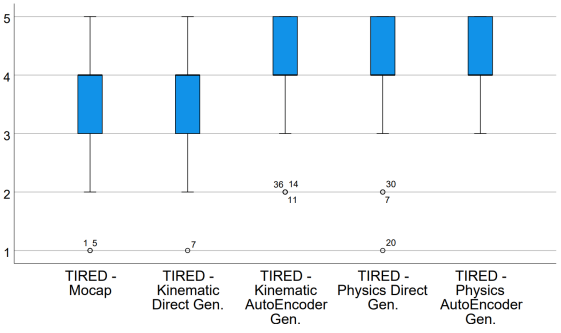
(c) Performance for the emotion “Angry”.



(d) Performance for the emotion “Confident”.



(e) Performance for the emotion “Afraid”.



(f) Performance for the emotion “Tired”.

Figure 6.4: Boxplot charts showing the value distribution for each of 6 of our tested emotions.

7

Conclusion

Contents

To conclude this document this chapter summarizes what the proposed system aimed to do and what it managed to accomplish in the end. We also provide some insight into possible improvements that could be done to the system in the future.

7.1 Conclusions

We set out to tackle an issue plaguing current computer animation techniques - the fact that to make a character display different emotions over the same baseline motion, an animator has to manually generate a new animation to add to a stack. Each emotion that the character should be able to convey equates to an additional animation that needs to be created, even if the underlying motion is the same. Furthermore, mocap of an actor performing the same exact movements in different emotional styles has to be readily available to be used as reference. This issue is repetitive, time-consuming, costly and prevalent regardless of whether animators are manually creating their animations or using an automatic motion learning system. To counteract this issue we created our Emotional Classification and Emotionally Expressive Motion Control system for Locomotion animations.

The developed system is not only able to correctly identify the emotion that a character is attempting to convey throughout its motion, but is also capable of tweaking the character's movements in real time in order to change its expressed emotion, specified through a set of PAD emotional coordinate values. Motion synthesis can be performed multiple times with changes being applied to the character's movements almost instantly and without breaking the underlying motion. Through the usage of select LMA features this system can accurately identify a character's expressed emotion in the 3D PAD Emotional space. Two distinct methodologies for generating sets of LMA Features with desired emotional values were designed and implemented, with one of them simply mapping the coordinates into LMA features, and the other using an AutoEncoder to reduce the mapping problem's complexity. Using these generated LMA Feature values we can then alter a character's motion in real time without the need for any additional data or training. Furthermore, our system works not only on Kinematic controllers driven by mocap, but also on physics-enabled characters controlled by learnt policies. The EEMC system was validated through a set of user tests to infer the quality of its synthesized motions. Furthermore an article describing the framework has been reviewed and was already accepted for publishing in an international conference - the *IEEE International Symposium on Multimedia*.

Our system's value lies in the fact that we can alter a motion's emotion in real time without the need for any further data or training. The system bypasses the need of having to record a mocap or train a character controller policy for each emotion that the character is meant to express over the same motion by managing to change the character's emotion instantaneously while its still performing the baseline movement. The fact that the system can be used interactively and that changes and predictions are

output in real time means that users can be used not only to create new animations that could then be extracted and used just like conventionally generated ones, but could also be integrated with applications that require modifications to be done during run time. For example, it would be possible to integrate our system with a video-game where character's need to alter their baseline motions in order to convey different emotions to react to an ever-changing environment, evolving story, or decisions made by the player on the spot. To showcase our emotional classification and expressive motion editing we also designed an easy to use and interactable GUI that allows users to alter a baseline motion's emotion by specifying new desired PAD coordinates in real time and without the need for specific domain knowledge.

7.2 Future Work

In terms of future work there are several possible avenues that could be followed to improve our system. Firstly the dataset we used grouped animations into preset styles rather than emotional coordinates. As such, all animations that aimed to express the same emotion were labeled with the exact same Pleasure, Arousal and Dominance values. In reality not all animations with the same emotion express it with the same intensity, and the emotional coordinate values are subject to change even during the course of the same animation. It would be interesting to explore our approaches using an enriched dataset that further split each animation's labels into chunks, adding more granularity to the emotional expression of the data. Increasing the overall animation and emotional variety, alongside tweaking our motion generation heuristic rules further may also improve our system's performance on non-locomotion animations. It would also be worth exploring different avenues for LMA Feature generation. More specifically, we believe that GANs [18] or VAEs [30] could have the baseline generative capabilities to accomplish the feature generation task [46] and possibly outperform our own PAD to LMA mapping methodologies. Whilst we did try to create such models, the results yielded were far from ideal, outputting worse results than the proposed techniques. We believe this may have been due to lack of data, salienting the need for more and more varied emotionally expressive mocap animation data. Finally, motions generated by our system can be somewhat exacerbated. Whilst the usage of the PAD model was chosen to allow for a granular definition of emotions, it would be interesting to see the inclusion of an additional "Intensity" slider to provide even more control on how exaggerated the synthesized motion's emotion is conveyed. A possible way to make this work would be to take the baseline neutral pose and the generated one and perform some form of interpolation between them to find a "middle-ground". The intensity slider could then be used as a form of weight given to the baseline pose over the generated one, changing how close we want the final output to be to its more neutral stance, over the generated emotion's.

Bibliography

[1] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. Diverse motion variations for physics-based character animation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 37–44, 2013.

[2] Insaf Ajili, Malik Mallem, and Jean-Yves Didier. Human motions and emotions recognition inspired by lma qualities. *The Visual Computer*, 35(10):1411–1426, 2019.

[3] Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from motion. In *Graphics interface*, volume 96, pages 222–229. Toronto, Canada, 1996.

[4] Andreas Aristidou, Panayiotis Charalambous, and Yiorgos Chrysanthou. Emotion Analysis and Classification: Understanding the Performers' Emotions Using the LMA Entities. In *Computer Graphics Forum*, volume 34, pages 262–276. Wiley, 4 2015.

[5] Andreas Aristidou and Yiorgos Chrysanthou. Motion indexing of different emotional states using lma components. In *SIGGRAPH Asia 2013 Technical Briefs*, pages 1–4. Association for Computing Machinery, 2013.

[6] Andreas Aristidou, Qiong Zeng, Efstathios Stavrakis, KangKang Yin, Daniel Cohen-Or, Yiorgos Chrysanthou, and Baoquan Chen. Emotion control of unstructured dance movements. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 1–10, 2017.

[7] Bandai Namco Research Inc. Bandai-Namco-Research-Motiondataset. https://github.com/BandaiNamcoResearchInc/Bandai-\Namco-Research-Motiondataset, 2022.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[9] Christoph Bregler. *Kinematic Motion Models*, pages 437–440. Springer US, Boston, MA, 2014.

[10] Joost Broekens and Doug DeGroot. Scalable and flexible appraisal models for virtual agents. In *Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education (CGAIDE)*, pages 208–215, 2004.

[11] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[12] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[13] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless telecommunications symposium (WTS)*, pages 1–5. IEEE, 2018.

[14] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Deep convolutional autoencoder-based lossy image compression. In *2018 Picture Coding Symposium (PCS)*, pages 253–257. IEEE, 2018.

[15] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[16] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel Van De Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics (TOG)*, 30(4):1–12, 2011.

[17] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016.

[18] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[19] Aditya Deshpande. Rotation using Euler Angles. https://adipandas.github.io/posts/2020/02/euler-rotation/, 02 2020.

[20] Thomas G Dietterich et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.

[21] Entertainment Software Association. 2022 Essential Facts About the Video Game Industry. https://www.theesa.com/resource/2022-essential-facts-about-the-video-game-industry/, 6 2022.

[22] Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[23] Ed Groff. Laban movement analysis: Charting the ineffable domain of human movement. *Journal of Physical Education, Recreation & Dance*, 66(2):27–30, 1995.

[24] Muhammad Arslan Hashmi, Qaiser Riaz, Muhammad Zeeshan, Muhammad Shahzad, and Muhammad Moazam Fraz. Motion reveal emotions: identifying emotions from human walk using chest mounted smartphone. *IEEE Sensors Journal*, 20(22):13511–13522, 2020.

[25] Holger Hoffmann, Andreas Scheck, Timo Schuster, Steffen Walter, Kerstin Limbrecht, Harald C. Traue, and Henrik Kessler. Mapping discrete emotions into the dimensional space: An empirical approach. *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3316–3320, 2012.

[26] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

[27] IBM Corp. IBM SPSS Statistics for Windows. https://www.ibm.com/spss, 2021.

[28] Carroll E Izard. Basic emotions, relations among emotions, and emotion-cognition relations. *Psychological Review*, 99(3):561–565, 1992.

[29] Ioannis A Kakadiaris. Physics-based modeling, analysis and animation. *Technical Reports (CIS)*, page 274, 1993.

[30] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.

[31] Midori Kitagawa and Brian Windsor. *MoCap for artists: workflow and techniques for motion capture*. Routledge, 2020.

[32] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 35–44, 1987.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[34] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. In *ACM SIGGRAPH 2010 papers*, volume 29, pages 1–8. Association for Computing Machinery (ACM), 7 2010.

[35] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[36] Bart Moyaers Logan King. Bvhtodeepmimic. https://github.com/BartMoyaers/BvhToDeepMimic, 2019.

[37] Kin Gwn Lore, Adedotun Akintayo, and Soumik Sarkar. Llnet: A deep autoencoder approach to natural low-light image enhancement. *Pattern Recognition*, 61:650–662, 2017.

[38] Fredrik Lundh. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[39] Li-Ke Ma, Zeshi Yang, Xin Tong, Baining Guo, and KangKang Yin. Learning and Exploring Motor Skills with Spacetime Bounds. In *Computer Graphics Forum*, volume 40, pages 251–263. Wiley, 5 2021.

[40] Albert Mehrabian. Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14(4):261–292, 1996.

[41] Chris Nicholson. A beginner's guide to deep reinforcement learning. https://wiki.pathmind.com/deep-reinforcement-learning, 2020.

[42] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[43] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[44] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

[45] J Ross Quinlan et al. Bagging, boosting, and c4. 5. In *Aaai/Iaai, vol. 1*, pages 725–730, 1996.

[46] Tanmay Randhavane, Uttaran Bhattacharya, Kyra Kapsaskis, Kurt Gray, Aniket Bera, and Dinesh Manocha. Identifying emotions from walking using affective and deep features. *arXiv preprint arXiv:1906.11884*, 2019.

[47] RM Reynolds, E Novotny, J Lee, D Roth, and G Bente. Ambiguous bodies: The role of displayed arousal in emotion [mis] perception. *Journal of Nonverbal Behavior*, 43(4):529–548, 2019.

[48] Kamrad Khoshhal Roudposhti, Luís Santos, Hadi Aliakbarpour, and Jorge Dias. Parameterizing interpersonal behaviour with laban movement analysis—a bayesian approach. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 7–13. IEEE, 2012.

[49] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[50] Alla Safonova and Jessica K Hodgins. Construction and optimal search of interpolated motion graphs. In *ACM SIGGRAPH 2007 papers*. ACM Press, 2007.

[51] Sabrina Schneider, Andrea Christensen, Florian B Häußinger, Andreas J Fallgatter, Martin A Giese, and Ann-Christine Ehlis. Show me how you walk and i tell you how you feel—a functional near-infrared spectroscopy study on emotion perception based on human gait. *Neuroimage*, 85:380–390, 2014.

[52] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[54] Murali Shanker, Michael Y Hu, and Ming S Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996.

[55] Shubham Sharma, Shubhankar Verma, Mohit Kumar, and Lavanya Sharma. Use of motion capture in 3d animation: motion capture systems, challenges, and recent trends. In *2019 international conference on machine learning, big data, cloud and parallel computing (comitcon)*, pages 289–294. IEEE, 2019.

[56] Michael R Sheldon, Michael J Fillyaw, and W Douglas Thompson. The use and interpretation of the friedman test in the analysis of ordinal-scale data in repeated measures designs. *Physiotherapy Research International*, 1(4):221–228, 1996.

[57] Ronald E Shiffler. Maximum z scores and outliers. *The American Statistician*, 42(1):79–80, 1988.

[58] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[59] Dean Takahashi. SuperData: Games grew 12% to $139.9 billion in 2020 amid pandemic, 1 2021.

[60] Silvan S Tomkins. Affect theory. *Approaches to emotion*, 163(163-195):31–65, 1984.

[61] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[62] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[63] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[64] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[65] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[66] Robert F Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.

[67] KangKang Yin, Kevin Loken, and Michiel Van de Panne. Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (TOG)*, 26(3):105–es, 2007.

[68] Zerrin Yumak, Maher Ben Moussa, Parag Chaudhuri, and Nadia Thalmann. Making them remember—emotional virtual characters with memory. *IEEE computer graphics and applications*, 29:20–9, 05 2009.

[69] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017.

A

Project Code & Useful Links

Table A.1 contains all links pertaining to our project, including the code-basis, article and website.

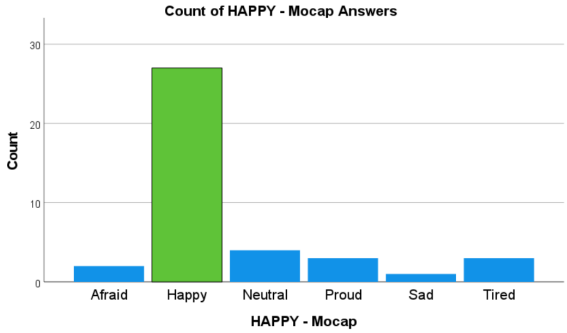
Table A.1: Project links.

Name	Link	Last Access Date
Code (GitHub)	https://github.com/HerouFenix/Emotionally-Expressive-Motion-Controller-for-Virtual-Characters	24/10/2022
Website	https://heroufenix.github.io/expressive_animations_web/	24/10/2022
Paper	https://heroufenix.github.io/expressive_animations_web/ISM_2022_Emotionally_Expressive_Controller_FINAL.pdf	24/10/2022
Video (Youtube)	https://youtu.be/QOfVkoZa5HM	24/10/2022
User Test Video Clips (Youtube)	https://youtube.com/playlist?list=PLBchdrsdyMe_y7oUAzumcPc3P5ft3SMi4	24/10/2022
User Test Reports (Download)	https://heroufenix.github.io/expressive_animations_web/results.zip	24/10/2022
Devlogs (Download)	https://heroufenix.github.io/expressive_animations_web/Devlogs.zip	28/10/2022

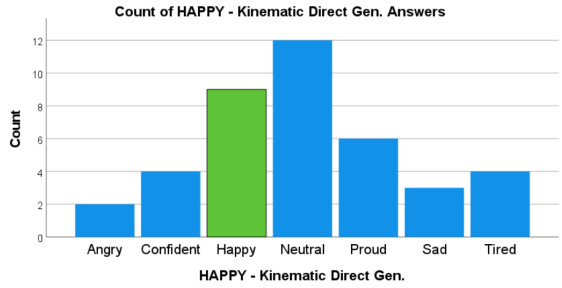
B

User Test Reports

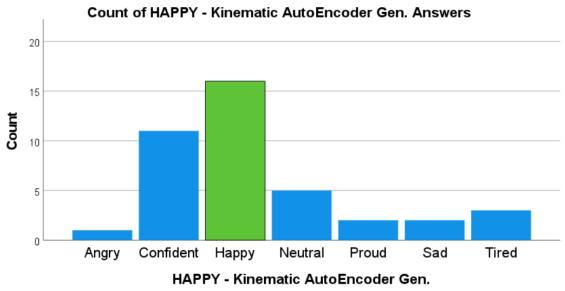
Following are several results from the tests done over the User Test Tasks described in Chapter 6. These tests and graphs were obtained using the IBM SPSS [27] tool. Figures B.1, B.2, B.3, B.4, B.5 and B.6 pertain to the “Emotional Identification” task and showcase the answers given by our participants for each type of clip - reference mocap, Direct generated on a Kinematic Character, AutoEncoder generated on a Kinematic Character, Direct Generated on a Policy-controlled Character and AutoEncoder Generated on a Policy-controlled character - for each tested emotion - Happy, Afraid, Tired, Angry, Confident and Sad. Figure B.7 shows the results of Friedman Tests done to ascertain whether there were any statistically significant differences between the answers given by participants for the different types of clips for the same emotion on the “Emotional Accuracy” test. Figures B.8, B.9, B.10, B.11 and B.12 are related to the “Primed Emotional Agreement” task and present the results of a Wilcoxon Signed Rank Test done to infer if there was any statistically significant difference between each of of our generated motions and the mocap, each motion generation type for Kinematic or Policy-controlled characters and between Kinematic and Policy-controlled characters using the same motion generation. While the Friedman Test gives us an overview of the whole set of answers, using this test we are able to determine if there is any statistically significant difference between each clip type pair.



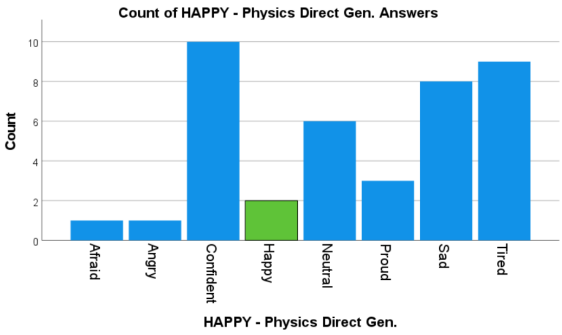
(a) Baseline Mocap performance.



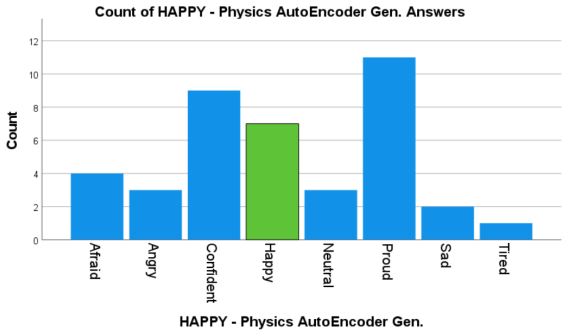
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.

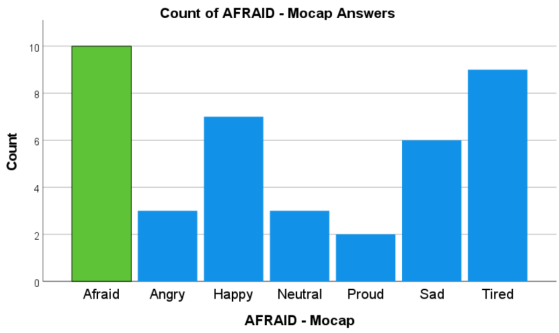


(d) Direct approach on a Policy Controller performance.

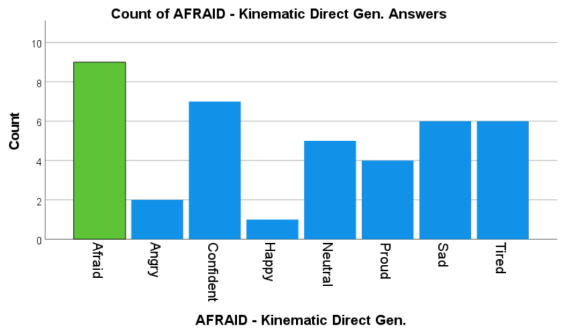


(e) AutoEncoder approach on a Policy Controller performance.

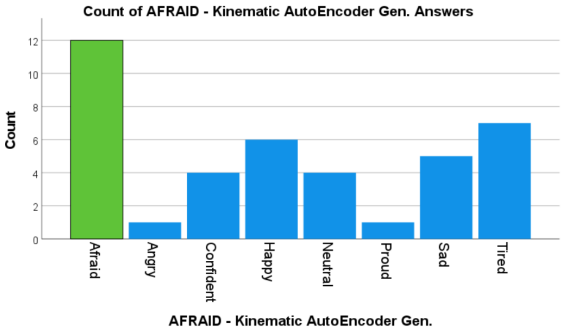
Figure B.1: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion "Happy".



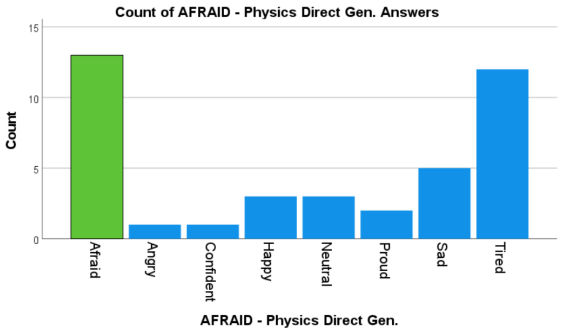
(a) Baseline Mocap performance.



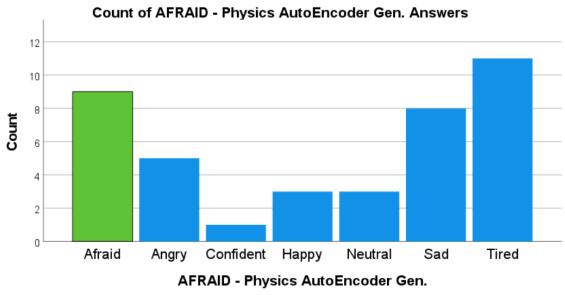
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.

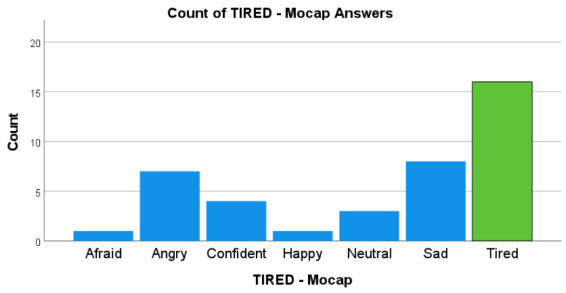


(d) Direct approach on a Policy Controller performance.

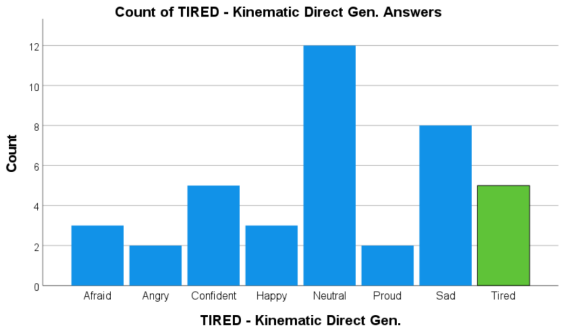


(e) AutoEncoder approach on a Policy Controller performance.

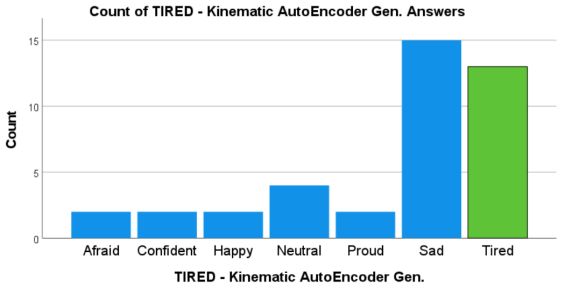
Figure B.2: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion "Afraid".



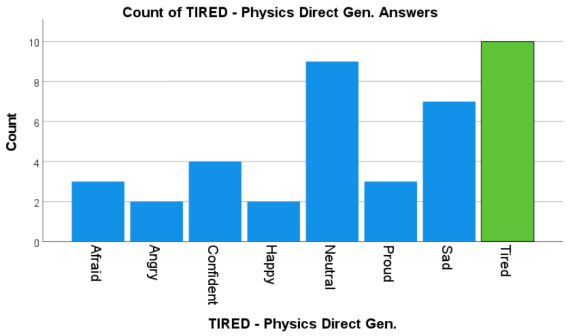
(a) Baseline Mocap performance.



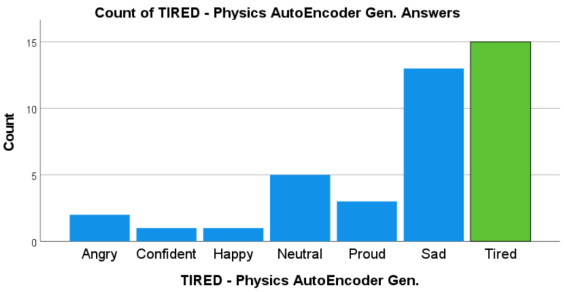
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.

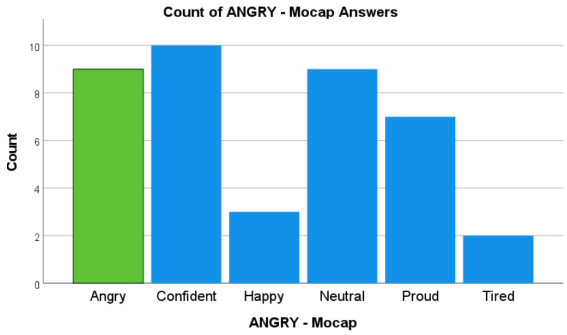


(d) Direct approach on a Policy Controller performance.

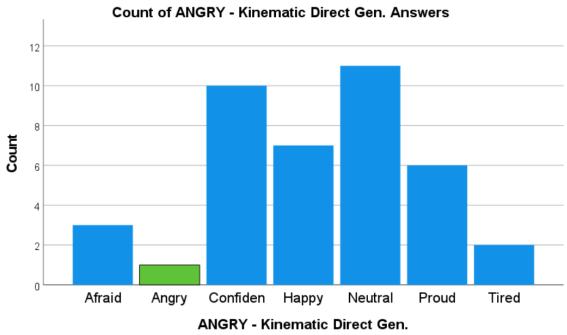


(e) AutoEncoder approach on a Policy Controller performance.

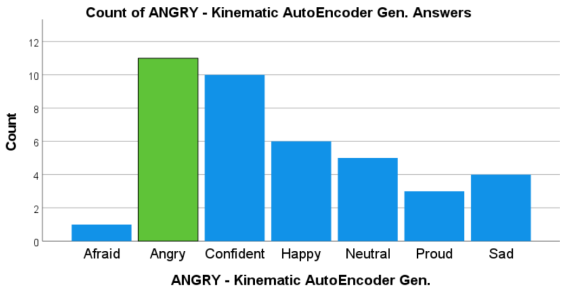
Figure B.3: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion "Tired".



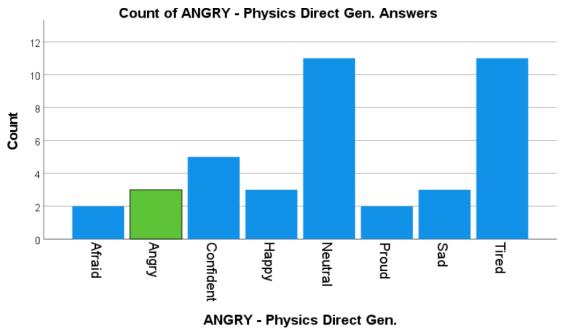
(a) Baseline Mocap performance.



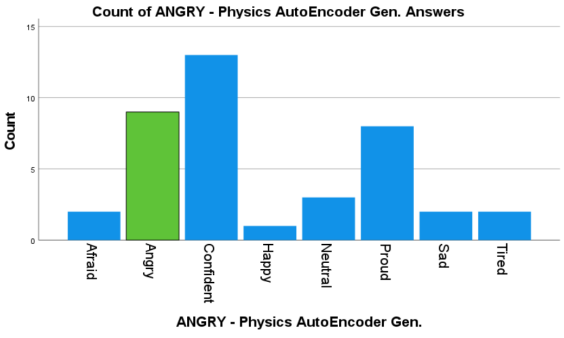
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.

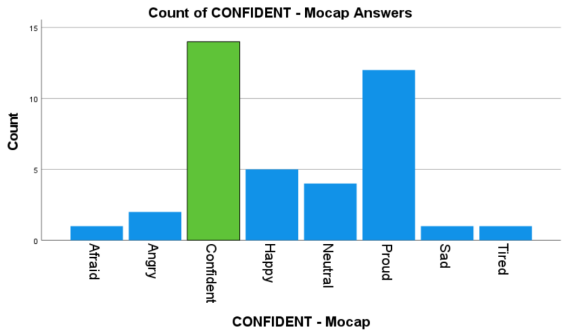


(d) Direct approach on a Policy Controller performance.

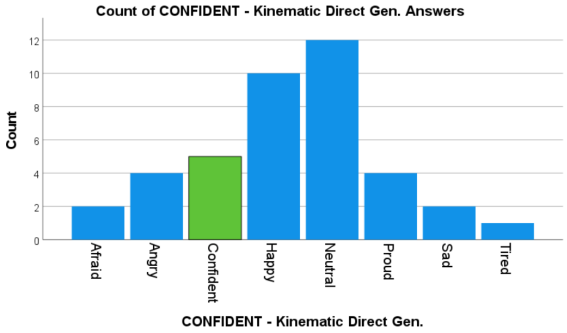


(e) AutoEncoder approach on a Policy Controller performance.

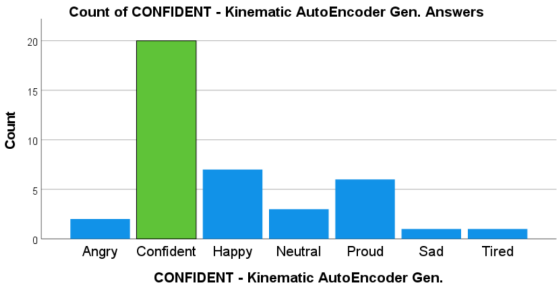
Figure B.4: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion "Angry".



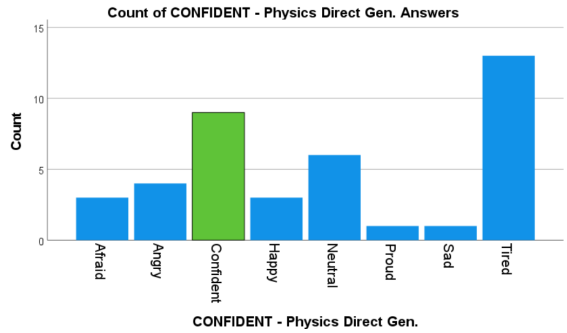
(a) Baseline Mocap performance.



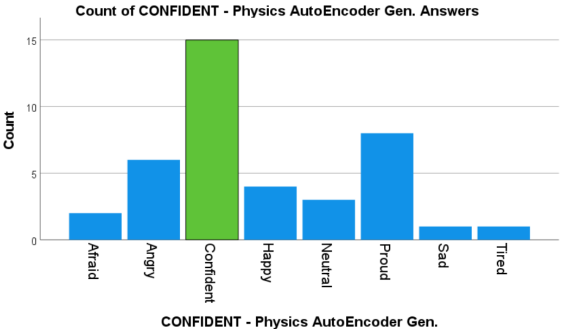
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.

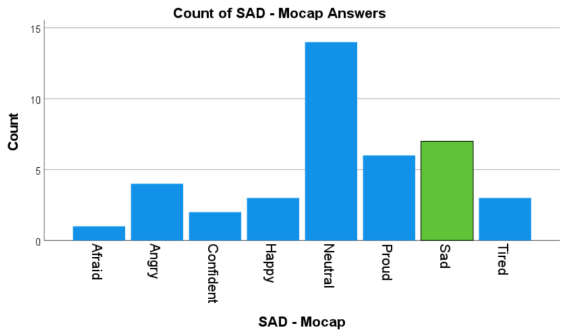


(d) Direct approach on a Policy Controller performance.

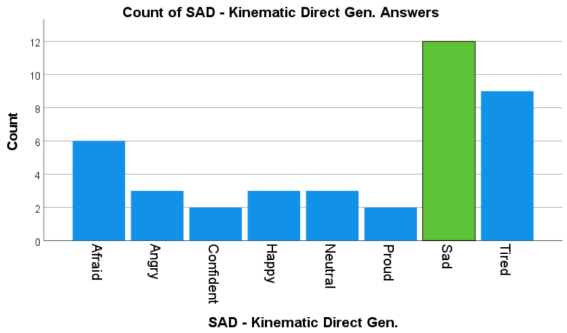


(e) AutoEncoder approach on a Policy Controller performance.

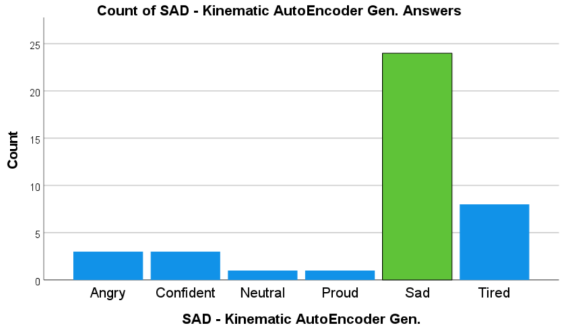
Figure B.5: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion “Confident”.



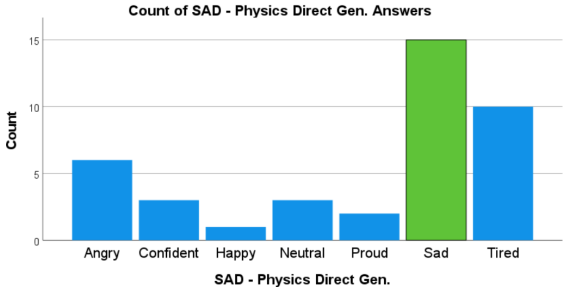
(a) Baseline Mocap performance.



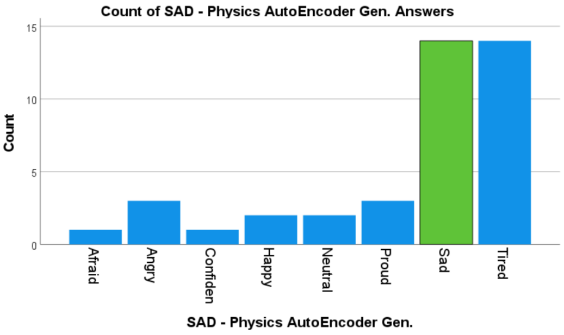
(b) Direct approach on a Kinematic Controller performance.



(c) AutoEncoder approach on a Kinematic Controller performance.



(d) Direct approach on a Policy Controller performance.



(e) AutoEncoder approach on a Policy Controller performance.

Figure B.6: Bar charts showcasing the count of answers given by test participants per video clip type for the emotion "Sad".

Ranks

	Mean Rank
HAPPY - Mocap	4.35
HAPPY - Kinematic Direct Gen.	2.83
HAPPY - Kinematic AutoEncoder Gen.	3.09
HAPPY - Physics Direct Gen.	1.65
HAPPY - Physics AutoEncoder Gen.	3.09

Test Statistics[a]

N	40
Chi-Square	76.571
df	4
Asymp. Sig.	<.001

a. Friedman Test

(a) Friedman Test results for the emotion “Happy”.

Ranks

	Mean Rank
AFRAID - Mocap	3.24
AFRAID - Kinematic Direct Gen.	3.03
AFRAID - Kinematic AutoEncoder Gen.	3.08
AFRAID - Physics Direct Gen.	2.75
AFRAID - Physics AutoEncoder Gen.	2.91

Test Statistics[a]

N	40
Chi-Square	3.400
df	4
Asymp. Sig.	.493

a. Friedman Test

(b) Friedman Test results for the emotion “Afraid”.

Ranks

	Mean Rank
TIRED - Mocap	2.63
TIRED - Kinematic Direct Gen.	2.45
TIRED - Kinematic AutoEncoder Gen.	3.33
TIRED - Physics Direct Gen.	3.28
TIRED - Physics AutoEncoder Gen.	3.33

Test Statistics[a]

N	40
Chi-Square	17.368
df	4
Asymp. Sig.	.002

a. Friedman Test

(c) Friedman Test results for the emotion “Tired”.

Ranks

	Mean Rank
ANGRY - Mocap	3.35
ANGRY - Kinematic Direct Gen.	2.35
ANGRY - Kinematic AutoEncoder Gen.	3.06
ANGRY - Physics Direct Gen.	2.89
ANGRY - Physics AutoEncoder Gen.	3.35

Test Statistics[a]

N	40
Chi-Square	16.066
df	4
Asymp. Sig.	.003

a. Friedman Test

(d) Friedman Test results for the emotion “Angry”.

Ranks

	Mean Rank
CONFIDENT - Mocap	3.30
CONFIDENT - Kinematic Direct Gen.	2.21
CONFIDENT - Kinematic AutoEncoder Gen.	3.66
CONFIDENT - Physics Direct Gen.	2.59
CONFIDENT - Physics AutoEncoder Gen.	3.24

Test Statistics[a]

N	40
Chi-Square	33.223
df	4
Asymp. Sig.	<.001

a. Friedman Test

(e) Friedman Test results for the emotion “Confident”.

Ranks

	Mean Rank
SAD - Mocap	2.10
SAD - Kinematic Direct Gen.	3.14
SAD - Kinematic AutoEncoder Gen.	3.38
SAD - Physics Direct Gen.	3.04
SAD - Physics AutoEncoder Gen.	3.35

Test Statistics[a]

N	40
Chi-Square	27.660
df	4
Asymp. Sig.	<.001

a. Friedman Test

(f) Friedman Test results for the emotion “Sad”.

Figure B.7: Results of a Friedman Test done to compare each of our video clip types for each emotion.

Test Statistics[a]

	HAPPY - Kinematic Direct Gen. - HAPPY - Mocap	HAPPY - Kinematic AutoEncoder Gen. - HAPPY - Mocap	HAPPY - Physics Direct Gen. - HAPPY - Mocap
Z	-4.468[b]	-4.191[b]	-5.386[b]
Asymp. Sig. (2-tailed)	<.001	<.001	<.001

Test Statistics[a]

	HAPPY - Physics AutoEncoder Gen. - HAPPY - Mocap	HAPPY - Kinematic AutoEncoder Gen. - HAPPY - Kinematic Direct Gen.	HAPPY - Physics Direct Gen. - HAPPY - Kinematic Direct Gen.
Z	-4.515[b]	-.783[c]	-3.798[b]
Asymp. Sig. (2-tailed)	<.001	.434	<.001

Test Statistics[a]

	HAPPY - Physics AutoEncoder Gen. - HAPPY - Kinematic AutoEncoder Gen.	HAPPY - Physics AutoEncoder Gen. - HAPPY - Physics Direct Gen.
Z	-.251[b]	-3.938[c]
Asymp. Sig. (2-tailed)	.802	<.001

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks.

c. Based on negative ranks.

Figure B.8: Results of a Wilcoxon Signed Rank Test done to compare each of our video clip types for the emotion "Happy".

Test Statistics[a]

	TIRED - Kinematic Direct Gen. - TIRED - Mocap	TIRED - Kinematic AutoEncoder Gen. - TIRED - Mocap	TIRED - Physics Direct Gen. - TIRED - Mocap
Z	-.720[b]	-2.200[c]	-1.753[c]
Asymp. Sig. (2-tailed)	.472	.028	.080

Test Statistics[a]

	TIRED - Physics AutoEncoder Gen. - TIRED - Mocap	TIRED - Kinematic AutoEncoder Gen. - TIRED - Kinematic Direct Gen.	TIRED - Physics Direct Gen. - TIRED - Kinematic Direct Gen.
Z	-2.951[c]	-3.261[c]	-2.608[c]
Asymp. Sig. (2-tailed)	.003	.001	.009

Test Statistics[a]

	TIRED - Physics AutoEncoder Gen. - TIRED - Kinematic AutoEncoder Gen.	TIRED - Physics AutoEncoder Gen. - TIRED - Physics Direct Gen.
Z	-.257[c]	-.776[c]
Asymp. Sig. (2-tailed)	.797	.438

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks.

c. Based on negative ranks.

Figure B.9: Results of a Wilcoxon Signed Rank Test done to compare each of our video clip types for the emotion "Tired".

Test Statistics[a]

	ANGRY - Kinematic Direct Gen. - ANGRY - Mocap	ANGRY - Kinematic AutoEncoder Gen. - ANGRY - Mocap	ANGRY - Physics Direct Gen. - ANGRY - Mocap
Z	-3.634[b]	-.623[b]	-1.198[b]
Asymp. Sig. (2-tailed)	<.001	.533	.231

Test Statistics[a]

	ANGRY - Physics AutoEncoder Gen. - ANGRY - Mocap	ANGRY - Kinematic AutoEncoder Gen. - ANGRY - Kinematic Direct Gen.	ANGRY - Physics Direct Gen. - ANGRY - Kinematic Direct Gen.
Z	-.499[c]	-2.668[c]	-2.099[c]
Asymp. Sig. (2-tailed)	.617	.008	.036

Test Statistics[a]

	ANGRY - Physics AutoEncoder Gen. - ANGRY - Kinematic AutoEncoder Gen.	ANGRY - Physics AutoEncoder Gen. - ANGRY - Physics Direct Gen.
Z	-1.325[c]	-1.976[c]
Asymp. Sig. (2-tailed)	.185	.048

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks.

c. Based on negative ranks.

Figure B.10: Results of a Wilcoxon Signed Rank Test done to compare each of our video clip types for the emotion “Angry”.

Test Statistics[a]

	CONFIDENT - Kinematic Direct Gen. - CONFIDENT - Mocap	CONFIDENT - Kinematic AutoEncoder Gen. - CONFIDENT - Mocap	CONFIDENT - Physics Direct Gen. - CONFIDENT - Mocap
Z	-3.747[b]	-1.286[c]	-2.553[b]
Asymp. Sig. (2-tailed)	<.001	.198	.011

Test Statistics[a]

	CONFIDENT - Physics AutoEncoder Gen. - CONFIDENT - Mocap	CONFIDENT - Kinematic AutoEncoder Gen. - CONFIDENT - Kinematic Direct Gen.	CONFIDENT - Physics Direct Gen. - CONFIDENT - Kinematic Direct Gen.
Z	-.209[b]	-3.661[c]	-1.627[c]
Asymp. Sig. (2-tailed)	.835	<.001	.104

Test Statistics[a]

	CONFIDENT - Physics AutoEncoder Gen. - CONFIDENT - Kinematic AutoEncoder Gen.	CONFIDENT - Physics AutoEncoder Gen. - CONFIDENT - Physics Direct Gen.
Z	-1.789[b]	-3.041[c]
Asymp. Sig. (2-tailed)	.074	.002

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks.

c. Based on negative ranks.

Figure B.11: Results of a Wilcoxon Signed Rank Test done to compare each of our video clip types for the emotion “Confident”.

Test Statistics[a]

	SAD - Kinematic Direct Gen. - SAD - Mocap	SAD - Kinematic AutoEncoder Gen. - SAD - Mocap	SAD - Physics Direct Gen. - SAD - Mocap
Z	-3.369[b]	-3.696[b]	-3.497[b]
Asymp. Sig. (2-tailed)	<.001	<.001	<.001

Test Statistics[a]

	SAD - Physics AutoEncoder Gen. - SAD - Mocap	SAD - Kinematic AutoEncoder Gen. - SAD - Kinematic Direct Gen.	SAD - Physics Direct Gen. - SAD - Kinematic Direct Gen.
Z	-3.858[b]	-.754[b]	-.364[c]
Asymp. Sig. (2-tailed)	<.001	.451	.716

Test Statistics[a]

	SAD - Physics AutoEncoder Gen. - SAD - Kinematic AutoEncoder Gen.	SAD - Physics AutoEncoder Gen. - SAD - Physics Direct Gen.
Z	.000[d]	-1.292[b]
Asymp. Sig. (2-tailed)	1.000	.197

a. Wilcoxon Signed Ranks Test

b. Based on negative ranks.

c. Based on positive ranks.

d. The sum of negative ranks equals the sum of positive ranks.

Figure B.12: Results of a Wilcoxon Signed Rank Test done to compare each of our video clip types for the emotion "Sad".