Natural Interaction exam project: LSTM and Kalman Filtering for stress level detection

Francesca Ferraro - 16125A

May 10, 2024

Contents

1 Introduction **3**
- 1.1 Oculus dataset . . . 4
- 1.2 Regression for Stress Level Estimation . . . 5

2 Methods **5**
- 2.1 Data Preprocessing . . . 5
- 2.2 Hyperparameters Tuning . . . 6
  - 2.2.1 Bayesian optimizer . . . 7
- 2.3 LSTM Network Architecture . . . 8
  - 2.3.1 Recurrent Neural Networks . . . 10
- 2.4 Time Series Split for Cross-Validation . . . 10
- 2.5 Kalman Filter . . . 11
  - 2.5.1 Parameters Choice and EM Algorithm . . . 13
- 2.6 Online learning . . . 14
- 2.7 Mean Squared Error . . . 15
- 2.8 Mean Absolute Error . . . 15

3 Results **16**
- 3.1 Model Hyperparameters . . . 16
- 3.2 Window sizes for online learning . . . 17
- 3.3 Window size : 2 sec. . . . 17
  - 3.3.1 Loss MSE . . . 18
  - 3.3.2 Loss MAE . . . 18
- 3.4 Window size : 4 sec. . . . 18
  - 3.4.1 Loss MSE . . . 19
  - 3.4.2 Loss MAE . . . 19
- 3.5 Window size : 6 sec. . . . 19
  - 3.5.1 Loss MSE . . . 20
  - 3.5.2 Loss MAE . . . 20
- 3.6 Test results . . . 20
- 3.7 Comparison between actual and LSTM predicted stress labels . . . 21
- 3.8 Training data . . . 21
- 3.9 Validation data . . . 22
- 3.10 Final remarks on the results . . . 22

4 Conclusion **23**
- 4.1 Comparison with Related Work . . . 23

1 Introduction

Virtual reality (VR) technology has witnessed rapid advancements, offering immersive and engaging experiences across diverse domains such as gaming, education, and training. As VR applications become increasingly sophisticated and prevalent, ensuring a positive and healthy user experience is of paramount importance. A crucial aspect of this is understanding and managing stress levels during VR interactions. Since VR experiences directly engage the user's sensory and motor systems, it's important to consider the underlying neural processes involved in stress responses.

The human brain plays a central role in processing and responding to stress. Key brain regions involved in the stress response include:

- **Amygdala:** The amygdala, located deep within the temporal lobe, is responsible for processing emotions, particularly fear and threat detection. It plays a critical role in initiating the "fight-or-flight" response, a rapid physiological reaction to perceived danger.
- **Hypothalamus:** The hypothalamus, situated at the base of the brain, acts as a control center for the autonomic nervous system and the endocrine system. It regulates various physiological processes, including heart rate, blood pressure, and the release of stress hormones such as cortisol.
- **Hippocampus:** The hippocampus, located in the medial temporal lobe, is essential for memory formation and spatial navigation. Stress can impair hippocampal function, leading to difficulties with memory and learning.
- **Prefrontal Cortex (PFC):** The PFC, situated at the front of the brain, is responsible for higher-order cognitive functions such as planning, decision-making, and impulse control. Stress can disrupt PFC activity, leading to impaired judgment and increased impulsivity.

These brain regions interact in a complex network to coordinate the body's response to stress. VR experiences, with their potential for cognitive load, sensory overload, and emotional intensity, can activate these stress-related neural pathways, leading to various physiological and psychological effects.

Stress, a natural physiological and psychological response to demanding situations, can manifest in VR environments due to several factors:

- **Cognitive Load:** Complex tasks, information overload, and challenging gameplay mechanics can induce cognitive strain, leading to stress.
- **Sensory Overload:** VR experiences often involve intense visual and auditory stimuli, which can overwhelm the sensory system and contribute to stress, particularly for users susceptible to motion sickness or sensory sensitivities.

- **Emotional Intensity:** VR experiences can evoke strong emotions, such as fear, excitement, or frustration, depending on the content and context. These intense emotions can trigger stress responses.
- **Physical Discomfort:** Prolonged VR usage can lead to physical discomfort due to factors like awkward body postures, fatigue, and the weight of the headset. This discomfort can further exacerbate stress levels.

Elevated stress levels during VR interactions can have detrimental effects on user experience and well-being:

- **Reduced Enjoyment and Engagement:** High stress can diminish the enjoyment of VR experiences and lead to disengagement, hindering the effectiveness of VR applications in areas like training and education.
- **Negative Emotional States:** Stress can contribute to anxiety, frustration, and other negative emotions, potentially impacting users' mood and mental state even after the VR session ends.
- **Physical Symptoms:** Stress can manifest in physical symptoms such as headaches, nausea, increased heart rate, and muscle tension, leading to discomfort and potentially discouraging further VR use.

Therefore, developing reliable methods for detecting and managing stress in VR environments is crucial for ensuring a positive user experience and promoting the responsible and healthy use of this technology.

This project aims to address this challenge by developing a machine learning model that can infer the stress level of a VR player based on their movement data. By analyzing movement patterns and identifying stress indicators, the model can provide valuable insights into the user's state and enable the implementation of adaptive strategies to mitigate stress and enhance the overall VR experience.

1.1 Oculus dataset

This project utilizes the OCULUS dataset, which comprises 6198 recordings of player movements and their corresponding stress levels during VR gameplay. The dataset encompasses a rich collection of features extracted from various sensors of the Oculus Quest 2 VR, including:

- **Head movements**: Angular acceleration, angular velocity, and linear acceleration of the player's head, captured through the VR headset's internal sensors.
- **Hand movements**: Grip pressure, trigger pressure, linear acceleration, angular acceleration, and linear velocity of both left and right hands, measured by hand-held controllers.

- **Thumbstick position**: The x and y coordinates and their statistics (mean, standard deviation, minimum, maximum) for both left and right thumbsticks on the controllers.

These features provide a comprehensive representation of the player's physical interactions within the VR environment.

Each data point is labeled with a "stress_label" indicating the ground truth stress level experienced by the player during that specific game section, along with a "subject" identifier to distinguish data from different participants.

1.2 Regression for Stress Level Estimation

This project adopts a regression approach for stress detection due to the continuous nature of stress levels. Regression analysis is a statistical method used to model the relationship between a dependent variable, in this case the stress level, and one or more independent variables, represented by the player's movement features. The objective is to develop a model that can accurately predict the stress level based on the observed movement patterns.

Regression models estimate a continuous-valued output, making them suitable for predicting quantities like stress levels that can vary along a spectrum. Unlike classification models, which categorize data into discrete classes, regression models provide a more nuanced understanding of the target variable.

Among various regression techniques, Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, are particularly well-suited for analyzing time-series data like player movements. LSTMs excel at capturing temporal dependencies and learning from sequential information, making them ideal for processing the dynamic nature of player movements and extracting relevant features for stress level prediction.

LSTMs achieve this through their unique architecture, which includes memory cells that can store information over extended periods and gates that control the flow of information within the network. This architecture allows LSTMs to learn complex relationships between past and present movement patterns, enabling the model to infer the player's stress level based on subtle changes and trends in their movements.

By employing LSTMs for regression, this project aims to develop a robust and accurate model for stress detection in VR environments, paving the way for adaptive interventions and improved user experiences.

2 Methods

2.1 Data Preprocessing

Effective machine learning models rely on high-quality data. Therefore, before feeding the VR player movement data into the LSTM model, several preprocess-

ing steps are crucial to ensure data quality and prepare it for effective model training. These steps include:

- **Handling Missing Values:** Real-world datasets often suffer from missing values due to sensor errors, data collection issues, or other unforeseen circumstances. To prevent these missing values from hindering the model's learning process, imputation techniques are employed. In this project, missing values are replaced with the mean of the respective feature column. This method maintains the overall distribution of the data while preventing the model from being misled by missing information or introducing bias.
- **Feature Normalization:** The features extracted from the VR sensors, such as acceleration, velocity, and grip pressure, often have varying scales and units. This can pose challenges for the LSTM model, as features with larger scales might dominate the learning process, overshadowing the contribution of features with smaller scales. To address this, feature normalization is applied using StandardScaler. This technique scales each feature to have zero mean and unit variance, ensuring that all features contribute equally to the model training and preventing any single feature from disproportionately influencing the learning process. Normalization also improves model training stability and convergence speed.

These preprocessing steps are essential for preparing the data for the LSTM model. By addressing missing values and ensuring consistent feature scaling, the data is transformed into a format that is more suitable for the LSTM to learn from, ultimately leading to more accurate and reliable stress level predictions.

2.2 Hyperparameters Tuning

The performance of an LSTM model heavily relies on the choice of hyperparameters, which control the model's architecture and learning process. Finding the optimal hyperparameter configuration is crucial for achieving the best possible accuracy and generalization. To efficiently explore the hyperparameter space and identify the most effective configuration, this project employs Bayesian optimization using the Ax library.

Bayesian optimization is a powerful approach that leverages probabilistic models to guide the search for optimal hyperparameters. It works by building a surrogate model of the relationship between hyperparameter configurations and the model's performance, based on past evaluations. This surrogate model is then used to select the next hyperparameter configuration to evaluate, balancing the exploration of new and potentially promising regions of the hyperparameter space with the exploitation of existing knowledge to focus on configurations that are likely to yield good performance.

The following hyperparameters are tuned using Bayesian optimization:

- **Learning rate (lr):** Controls the step size during gradient descent optimization.

- **Hidden size:** The number of neurons in the hidden layers of the LSTM.
- **Number of layers:** The number of LSTM layers stacked in the model.
- **Number of epochs:** The number of times the entire training dataset is passed through the model.
- **Dropout rates:** The probability of dropping neurons during training to prevent overfitting.
- **Weight decay:** Regularization parameter for L2 optimization to prevent overfitting.
- **Batch size:** The number of samples processed before updating the model parameters.

By employing Bayesian optimization, this project efficiently identifies the optimal hyperparameter configuration, leading to improved model performance and more accurate stress level predictions.

2.2.1 Bayesian optimizer

Hyperparameter optimization is a critical step in the machine learning pipeline, as it directly impacts the model's performance and generalizability. This project utilizes Bayesian optimization, a powerful and efficient approach that leverages probabilistic modeling to navigate the complex landscape of hyperparameter space and identify optimal configurations.

At the heart of Bayesian optimization lies the concept of a surrogate model, typically a Gaussian Process (GP), which serves as a probabilistic representation of the relationship between hyperparameter configurations ($\mathbf{x}$) and the objective function $f(\mathbf{x})$ (e.g., validation loss) that have to be minimized. The GP models the objective function as a distribution over functions, capturing both the predicted values and the associated uncertainty.

Formally, a GP is defined by a mean function $m(\mathbf{x})$ and a covariance function $k(\mathbf{x}, \mathbf{x}')$:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{1}$$

The mean function represents the expected value of the objective function at a given hyperparameter configuration, while the covariance function encodes assumptions about the smoothness and similarity of the objective function across different configurations. A common choice for the covariance function is the squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2l^2}) \tag{2}$$

where l is a length-scale parameter that controls the smoothness of the function.

As is evaluated the hyperparameter configurations and observed their corresponding objective function values, this information is incorporated into the GP model using Bayes' rule. This allows us to update our beliefs about the objective function and refine the GP model iteratively.

Given a set of observed data points $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $y_i = f(\mathbf{x}_i)$, the posterior distribution of the GP at a new point $\mathbf{x}^*$ is also a Gaussian distribution:

$$f(\mathbf{x}^*)|D \sim N(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*)) \tag{3}$$

where the posterior mean and variance are computed based on the observed data and the GP prior.

To guide the search for optimal hyperparameters, an acquisition function is employed. The acquisition function balances exploration of uncharted regions of the hyperparameter space with exploitation of areas identified as promising by the current GP model. Popular choices for acquisition functions include:

Expected Improvement (EI): Measures the expected increase in performance compared to the current best observation.

$$EI(\mathbf{x}) = E[(f_{min} - f(\mathbf{x}))^+] \tag{4}$$

where f_{min} is the current best observed value and $(.)^+$ denotes the positive part.

Upper Confidence Bound (UCB): Balances exploration and exploitation by considering both the predicted mean and the uncertainty (variance) of the GP model.

$$UCB(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}) \tag{5}$$

where κ is a parameter controlling the trade-off between exploration and exploitation.

Bayesian optyimizer in the Ax python library implementation utilizes the EI acquisition function.

The acquisition function is optimized to select the next hyperparameter configuration to evaluate. This iterative process of evaluation, GP model updates, and acquisition function optimization continues until a stopping criterion is reached, such as a maximum number of evaluations or achieving a satisfactory level of performance.

By combining prior knowledge with observed data, Bayesian optimization effectively navigates the hyperparameter space, leading to faster convergence towards optimal configurations and improved model performance compared to traditional methods like grid search or random search. This makes Bayesian optimization a valuable tool for tuning LSTM models and achieving accurate stress level predictions in VR environments.

2.3 LSTM Network Architecture

At the heart of this project's stress detection model lies a Long Short-Term Memory (LSTM) network, specifically chosen for its effectiveness in handling

sequential data and capturing temporal dependencies crucial for understanding player behavior and stress responses in VR environments. LSTMs belong to the family of recurrent neural networks (RNNs), designed to process sequential information by maintaining an internal memory or hidden state that evolves over time.

Traditional RNNs, however, suffer from the vanishing gradient problem, where gradients diminish as they propagate back through time, hindering the network's ability to learn long-term dependencies. LSTMs overcome this limitation by incorporating a unique architecture with memory cells and gating mechanisms.

An LSTM cell consists of three main gates:

- Forget Gate: Determines which information from the previous hidden state should be discarded or retained.

- Input Gate: Controls which aspects of the current input should be incorporated into the cell state.

- Output Gate: Regulates how much of the cell state should be used to compute the current output.

These gates, implemented as sigmoidal neural networks, allow the LSTM cell to selectively update and control the flow of information, enabling it to learn long-term dependencies effectively.

The LSTM network architecture in this project consists of the following components:

- **Input Layer:** This layer receives the preprocessed player movement data as a sequence of feature vectors, where each vector represents the player's movement characteristics at a specific time step.

- **LSTM Layers:** The network employs two stacked LSTM layers, each with a specific hidden size determined through hyperparameter optimization. These layers process the sequential input data, learning complex relationships between past and present movement patterns. The hidden state of each LSTM cell acts as a memory mechanism, retaining relevant information from previous time steps to inform the current prediction. This ability to learn long-term dependencies allows the network to capture subtle changes and trends in player movements that might be indicative of stress.

- **Dropout Layers:** To prevent overfitting and enhance the model's generalizability, dropout layers are applied after each LSTM layer. Dropout randomly deactivates a certain percentage of neurons during training, forcing the network to learn more robust and redundant representations that are less reliant on specific neurons and thus less prone to overfitting.

- **Fully Connected Layer:** The final layer is a fully connected layer, which maps the output of the LSTM layers to a single continuous value representing the predicted stress level. This layer acts as a regression head, translating the extracted features and temporal dynamics into a meaningful stress level estimate.

By leveraging the power of LSTMs and their ability to learn from sequential data, this project aims to develop a robust and accurate model for stress detection in VR, enabling adaptive interventions and contributing to a more positive and healthy user experience within immersive virtual environments.

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks specifically designed to handle sequential data, such as time series, natural language, or, in the context of this project, player movement patterns in VR environments. Unlike traditional feedforward neural networks that treat inputs independently, RNNs introduce the concept of a hidden state, which acts as a memory mechanism, allowing the network to learn from past information and capture temporal dependencies within the sequence.

Mathematically, an RNN can be described by the following equations:

$$\mathbf{h}_t = f(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{6}$$

$$\mathbf{y}_t = g(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y) \tag{7}$$

where:
- $\mathbf{x}_t$ is the input vector at time step t.
- $\mathbf{h}_t$ is the hidden state vector at time step t.
- $\mathbf{y}_t$ is the output vector at time step t.
- $\mathbf{W}_{hx}$, $\mathbf{W}_{hh}$, and $\mathbf{W}_{yh}$ are weight matrices.
- $\mathbf{b}_h$ and $\mathbf{b}_y$ are bias vectors.
- f and g are activation functions, such as tanh or ReLU.

The hidden state $\mathbf{h}_t$ is updated recursively, incorporating information from both the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$. This allows the network to maintain a "memory" of past inputs and utilize this information to influence its current output.

However, traditional RNNs can suffer from the vanishing gradient problem, where gradients tend to diminish exponentially as they propagate back through time during backpropagation. This makes it challenging for the network to learn long-term dependencies, as the influence of earlier inputs on the current output becomes negligible.

Despite this limitation, RNNs laid the groundwork for more advanced architectures like LSTMs, which address the vanishing gradient problem and excel at learning long-term dependencies. The ability of RNNs to process sequential data and capture temporal dynamics makes them valuable for various tasks, including speech recognition, machine translation, and, as explored in this project, stress detection.

2.4 Time Series Split for Cross-Validation

To ensure a robust and reliable evaluation of the LSTM model, respecting the temporal nature of the data and the variability across subjects, a subject-based

time series cross-validation approach is adopted. The dataset is first divided into groups based on subject ID. Then, within each group, the TimeSeriesSplit method is applied to split the data into training and validation sets while maintaining the chronological order of the data points. This approach is crucial for preventing the model from learning from future information during training, which could lead to overly optimistic performance estimates and poor generalization to real-world scenarios where only past data is available for prediction.

The TimeSeriesSplit method works by creating splits that respect the temporal order of the data, ensuring that the model is evaluated on its ability to predict future stress levels based on past observations. Each split consists of a training set containing all the data points up to a certain point in time and a validation set containing the subsequent data points. This process is repeated for a specified number of splits, creating multiple training and validation sets that cover different segments of the time series data. By evaluating the model on these different splits, is obtained a more comprehensive assessment of its performance and generalizability across various time periods within the dataset.

In the context of this project, the TimeSeriesSplit method is particularly important due to the inherent temporal dependencies within player movement data. Stress levels are likely to evolve over time, and the model needs to learn from past movement patterns to accurately predict future stress levels. By preserving the temporal order of the data during cross-validation, is ensured that the model is evaluated on its ability to capture these temporal dependencies and make realistic predictions based on past observations.

2.5 Kalman Filter

While LSTM networks excel at learning complex temporal dependencies from sequential data, their predictions can sometimes exhibit noise or fluctuations due to the inherent stochasticity of the training process and the complexity of the underlying data. To mitigate this and potentially enhance the accuracy and robustness of stress level predictions, this project incorporates a Kalman Filter as a post-processing step for the LSTM network's output.

The Kalman Filter is a statistical algorithm renowned for its ability to optimally estimate the state of a dynamic system by considering both the system's underlying dynamics and noisy measurements. In this context, the player's stress level is modeled as a dynamic system that evolves over time, and the LSTM network's output is treated as a noisy measurement of this state. By incorporating a model of how stress levels might evolve and accounting for measurement noise, the Kalman Filter can refine the stress level estimates and provide smoother, more accurate predictions.

The implementation of the Kalman Filter requires defining a state-space model that describes the evolution of the stress level and the relationship between the state and the measurements. In this project, a simple linear model is used for the state transition, assuming that the stress level at the current

time step is primarily influenced by the stress level at the previous time step. The measurement model directly relates the state (stress level) to the LSTM network's output.

The parameters of the state-space model, including the state transition matrix and the measurement noise covariance, are chosen based on initial observations of the data and domain knowledge. Fine-tuning of these parameters can be further explored in future work to potentially improve the Kalman Filter's performance.

By integrating the Kalman Filter, the project aims to leverage the strengths of both LSTM networks and state estimation techniques, ultimately leading to more reliable and accurate predictions of stress levels in VR environments.

The Kalman Filter operates on a state-space model, which consists of two equations:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{8}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{9}$$

where:

- $\mathbf{x}_k$ is the state vector at time step k.
- $\mathbf{F}_k$ is the state transition matrix, describing how the state evolves over time.
- $\mathbf{B}_k$ is the control input matrix, relating control inputs to the state.
- $\mathbf{u}_k$ is the control input vector.
- $\mathbf{w}_k$ is the process noise vector, representing random disturbances affecting the state.
- $\mathbf{z}_k$ is the measurement vector at time step k.
- $\mathbf{H}_k$ is the measurement matrix, relating the state to the measurements.
- $\mathbf{v}_k$ is the measurement noise vector, representing noise in the sensor readings.

The Kalman Filter algorithm consists of two main steps: prediction and update.

Prediction: The filter uses the state-space model to predict the next state and its uncertainty (covariance) based on the previous state and the current control inputs.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{10}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{11}$$

Update: When a new measurement becomes available, the filter incorporates this information to refine its estimate of the current state and its uncertainty.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{12}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{13}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{14}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{15}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{16}$$

The Kalman gain $\mathbf{K}_k$ determines how much weight is given to the new measurement compared to the predicted state.

By recursively performing prediction and update steps, the Kalman Filter provides an optimal estimate of the system state given all available information, taking into account both the system dynamics and the measurement noise.

In this project, the output of the LSTM network is treated as a noisy measurement of the player's stress level. The Kalman Filter, by incorporating a model of how stress levels might evolve over time and considering measurement noise, has the potential to refine the stress level estimates and provide smoother, more accurate predictions.

2.5.1 Parameters Choice and EM Algorithm

The effectiveness of the Kalman Filter hinges on the accurate definition of the state-space model, which involves specifying the state transition matrix (F_k), the control input matrix (B_k), the measurement matrix (H_k), and the noise covariance matrices (Q_k and R_k). These matrices govern the system dynamics, the relationship between the state and the measurements, and the uncertainties associated with the process and measurements.

In this project, due to the limited knowledge about the exact dynamics of stress evolution and the absence of external control inputs, a simplified state-space model is employed. The state transition matrix F_k is set as a 1x1 identity matrix, reflecting the assumption that the stress level at the current time step (x_k) is primarily influenced by the stress level at the previous time step (x_{k-1}), without considering additional control inputs:

$$x_k = F_k x_{k-1} + w_k \tag{17}$$

where w_k represents the process noise, which is assumed to be a Gaussian white noise with covariance Q_k. The measurement matrix H_k is also a 1x1 identity matrix, directly relating the state (stress level) to the LSTM network's output (z_k):

$$z_k = H_k x_k + v_k \tag{18}$$

where v_k represents the measurement noise, also assumed to be a Gaussian white noise with covariance R_k.

Estimating the noise covariance matrices (Q_k and R_k) is crucial for the Kalman Filter's performance.

Q_k captures the inherent variability and uncertainties in the stress evolution process, while R_k accounts for the inaccuracies and uncertainties in the LSTM network's predictions.

To estimate these noise covariances, the Expectation-Maximization (EM) algorithm is employed. The EM algorithm is an iterative method for finding maximum likelihood estimates of parameters in probabilistic models with latent variables. In this context, the latent variable is the true underlying stress level (x_k),

and the observed variables are the noisy measurements from the LSTM network (z_k).

The EM algorithm iteratively alternates between two steps:

E-step (Expectation): Computes the posterior distribution of the latent variables given the current model parameters and the observed data. This involves calculating the expected value and covariance of the state variables conditioned on the measurements:

$$\hat{x}_{k|k} = E[x_k|z_{1:k}] \tag{19}$$

$$P_{k|k} = Cov[x_k|z_{1:k}] \tag{20}$$

M-step (Maximization): Updates the model parameters (Q_k and R_k) by maximizing the expected log-likelihood of the observed data given the current estimates of the latent variables. This involves computing the sample covariances of the prediction errors and the measurement residuals:

$$Q_k = \frac{1}{N}\sum_{i=1}^{N}(x_{k|k} - F_k x_{k-1|k-1})(x_{k|k} - F_k x_{k-1|k-1})^T \tag{21}$$

$$R_k = \frac{1}{N}\sum_{i=1}^{N}(z_k - H_k x_{k|k})(z_k - H_k x_{k|k})^T \tag{22}$$

This iterative process refines the estimates of the noise covariances, leading to a more accurate and robust Kalman Filter model.

While this project utilizes a simplified state-space model and relies on the EM algorithm for noise covariance estimation, further exploration and fine-tuning of these aspects, such as considering more complex state transition models or alternative parameter estimation techniques, can be considered in future work to potentially improve the overall performance and generalizability of the stress detection system.

2.6 Online learning

Online learning is a machine learning paradigm where the model continuously learns and updates itself as new data becomes available, as opposed to traditional batch learning where the model is trained on a fixed dataset. This makes online learning particularly well-suited for dynamic environments where data patterns change over time, such as in the case of stress detection in VR where player behavior and stress levels can evolve throughout the experience.

Temporal windows, also known as sliding windows, play a crucial role in online learning by defining the portion of the data stream that the model uses for updating its parameters at each time step. The window size determines the amount of past information considered by the model, influencing its ability to capture temporal dependencies and adapt to changing patterns.

2.7 Mean Squared Error

The Mean Squared Error (MSE) is a widely used loss function for regression tasks and serves as the primary metric for evaluating the performance of the LSTM model in predicting stress levels. MSE measures the average squared difference between the predicted stress values ($\hat{y}_i$) and the corresponding ground truth stress labels (y_i) in the dataset. Mathematically, MSE is calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{23}$$

where n represents the number of samples in the dataset. By squaring the errors, MSE penalizes larger discrepancies more heavily, making it sensitive to outliers and emphasizing the model's ability to accurately predict stress levels across the entire range of values. During training and validation, MSE is calculated after each epoch to quantify the model's performance and guide the optimization process towards minimizing prediction errors. The average MSE is also computed on the test set to assess the model's generalization ability to unseen data.

2.8 Mean Absolute Error

To complement the Mean Squared Error (MSE), the Mean Absolute Error (MAE) is employed as an additional evaluation metric. MAE measures the average absolute difference between the predicted stress values ($\hat{y}_i$) and the ground truth labels (y_i). The mathematical formula for MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{24}$$

Unlike MSE, which squares the errors, MAE assigns equal weight to all errors regardless of their magnitude. This characteristic makes MAE less sensitive to outliers compared to MSE and provides a more intuitive understanding of the average magnitude of prediction errors. By including both MSE and MAE, it's obtained a more comprehensive evaluation of the model's performance, balancing sensitivity to outliers with a measure of typical prediction errors.

3 Results

The evaluation of the LSTM model, considering subject-based splitting and online learning with different sizes of time windows, has shown promising results in its ability to predict stress levels from player movement data within the virtual reality environment. The learning curves, depicting both loss and Mean Absolute Error (MAE) over training epochs, exhibit a clear downward trend, indicating the model's effective learning from the data and progressive improvement in prediction accuracy. This convergence towards stability suggests that the model reaches a point where further training does not yield significant gains. Importantly, the absence of a noticeable gap between the training and validation curves signifies that the model generalizes well to unseen data, avoiding overfitting to the training set.

In general terms, the Kalman Filtering on the LSTM output doesn't seem to make a significant improvement on the LSTM's results, but it can be worthwhile to further explore its potential under different configurations and with larger datasets, especially when dealing with noisier data or more complex stress dynamics. Exploring alternative state-space models or incorporating additional information, such as physiological data, into the filtering process could potentially enhance its effectiveness and lead to more robust and accurate stress estimation systems.

3.1 Model Hyperparameters

Bayesian optimization of the parameters, performed using the Ax library and splitting the entire dataset into training, validation, and test sets, identified the following optimal configuration for the LSTM model:

- **learning rate** : 6.5743e-06
- **hidden layer size** : 75 neurons
- **LSTM layers** : 2
- **number of epochs** : 27
- **dropout rates** : 52.05% for the first LSTM layer, 4.86% for the second LSTM layer, 44.31% for the fully connected layer
- **weight decay** : 0.979
- **batch size** : 66

It's important to note that the results might slightly vary depending on the random data split. However, the general trend of the learning curves and the conclusions drawn about the model's effectiveness and the impact of the Kalman filter remain valid.

3.2 Window sizes for online learning

To investigate the effectiveness of online learning, the LSTM model was trained and evaluated using an online learning approach with varying window sizes. Given that data are acquired with a 1-second sampling frequency, three different sets of window sizes were tested.

The first set included window sizes of 30, 60, 90, and 120 seconds. However, this resulted in flat loss graphs, indicating that the model was not learning effectively.

In response to this, a decision was made to test smaller window sizes. A second set of window sizes was thus tested, including 5, 10, 20, and 30 seconds. This led to an improvement in the model's learning, although the learning process was still relatively slow.

To further optimize the model's learning, a third set of window sizes was tested, excluding values above 10 seconds. Therefore, the final set of window sizes included only **2, 4 and 6 seconds**.

This approach of training the model on 'windows' of data of varying sizes, with each window passed through the model as a single batch, allows the model to update its weights after each window of data. This is particularly useful for time-series data, as it enables the model to adapt to temporal trends in the data.

In the sections below will be showed and discussed the results of MSE and MAE for each of the 3 chosen window sizes.

3.3 Window size : 2 sec.

The evaluation of the LSTM model for stress level prediction, considering both the original and Kalman-filtered outputs, provided valuable insights into its learning capabilities. As illustrated in Figures 1 and 2, which represent specific dataset and model configurations that may vary slightly between examples, both the Mean Squared Error (MSE) and Mean Absolute Error (MAE) exhibited a consistent downward trend throughout the training epochs. This decline in both training and validation loss metrics demonstrates the model's effectiveness in capturing the intricate relationship between player movement patterns and stress levels. However, a noteworthy observation emerged when comparing the performance of the model with and without the Kalman filter. The model trained without the filter consistently achieved lower average MSE and MAE values during training, suggesting that the LSTM network alone is adept at handling the noise and complexity inherent in the data. While the Kalman filter offered a marginal smoothing effect, its impact on validation and test performance remained minimal, further supporting the efficacy of the LSTM network in capturing the underlying temporal dynamics and patterns within the data. This suggests that while Kalman filtering may provide some benefits in specific scenarios, the LSTM network demonstrates robust performance without the need for additional filtering techniques.

3.3.1 Loss MSE

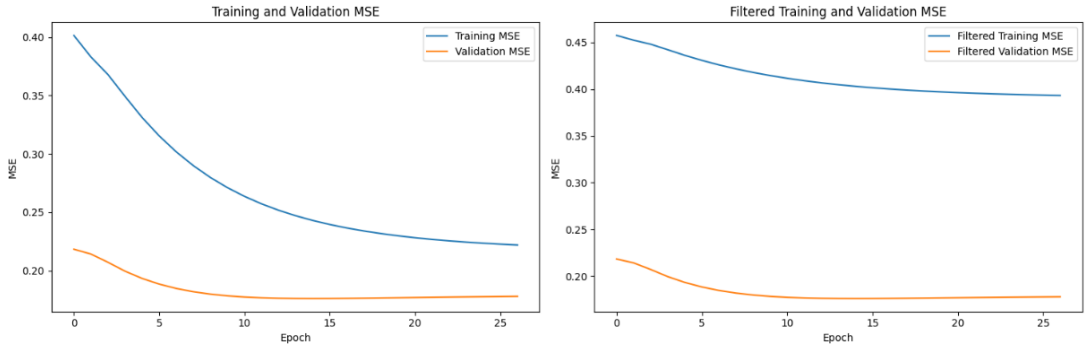

Figure 1: Example of MSE results for 2s-sized windows

3.3.2 Loss MAE

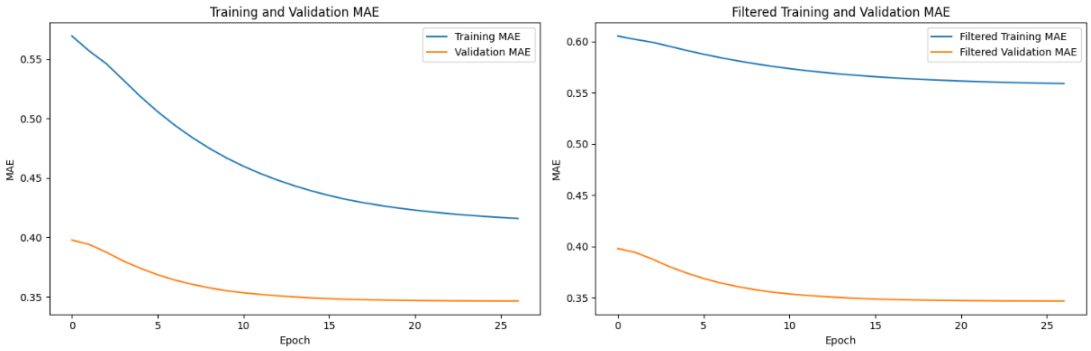

Figure 2: Example of MAE results for 2s-sized windows

3.4 Window size : 4 sec.

Following the analysis of the 2-second window size, the next step is to investigate the performance of the LSTM model with a 4-second window. Similar to the observations made with the 2-second window, both the MSE and MAE loss functions exhibited a consistent downward trend throughout the training epochs, indicating effective learning by the model. Interestingly, the overall performance with the 4-second window showed a slight improvement compared to the 2-second window. This suggests that providing the model with a slightly longer temporal context allows it to capture more intricate patterns and dependencies within the data, leading to enhanced predictive accuracy. However, the trends regarding the Kalman filter's impact remained consistent. The LSTM model trained without the Kalman filter continued to demonstrate superior performance in terms of training loss, achieving lower MSE and MAE values compared to the model incorporating the filter. This further supports the notion that the LSTM network

itself is capable of effectively handling the noise and complexity present in the data.

3.4.1 Loss MSE

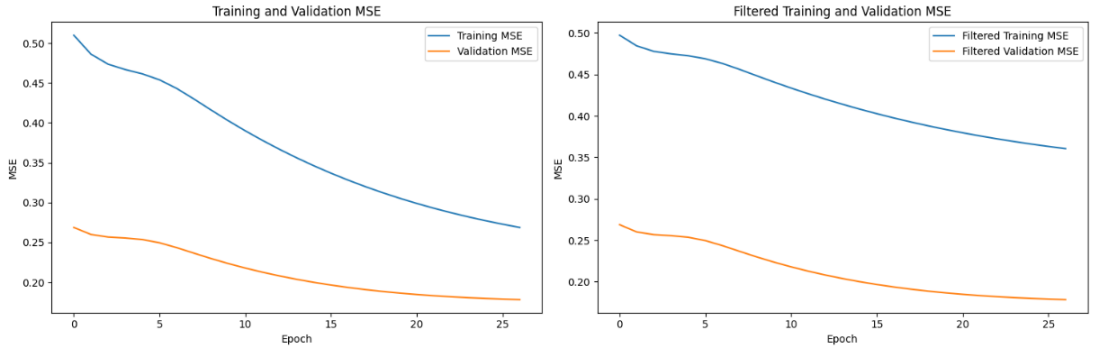

Figure 3: Example of MSE results for 4s-sized windows

3.4.2 Loss MAE

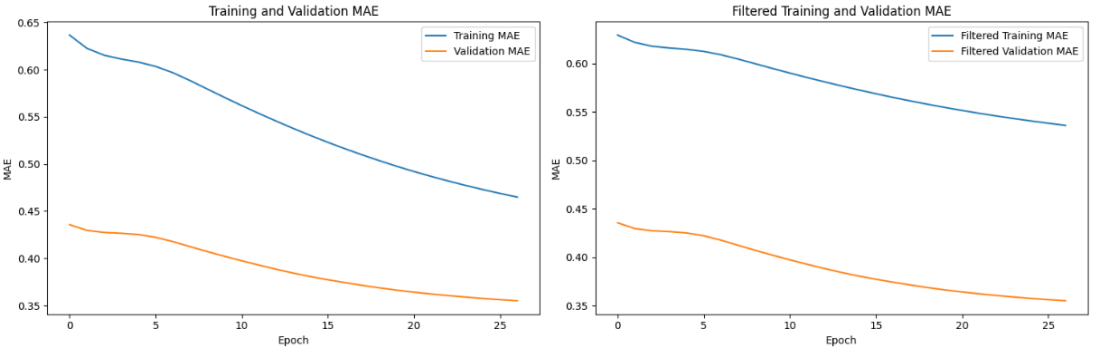

Figure 4: Example of MAE results for 4s-sized windows

3.5 Window size : 6 sec.

Continuing the investigation with a 6-second window size, the LSTM model's performance aligns with the general trend observed in the 2 and 4-second configurations. Both the MSE and MAE loss functions displayed a consistent downward trajectory throughout the training epochs, confirming the model's ability to learn and improve its predictions. This consistent behavior across different window sizes indicates that the LSTM model effectively captures relevant temporal dependencies within the data, regardless of the specific time frame. However, when comparing the 6-second window to the 2 and 4-second options, it becomes apparent that extending the temporal context further does not yield additional benefits. The overall performance improvement observed when moving from a 2-second to a 4-second window does not extend to the 6-second configuration. This

suggests that while providing some additional context can be beneficial, exceeding a certain threshold may introduce irrelevant information or noise, hindering the model's ability to identify the most crucial features for accurate predictions.

3.5.1 Loss MSE

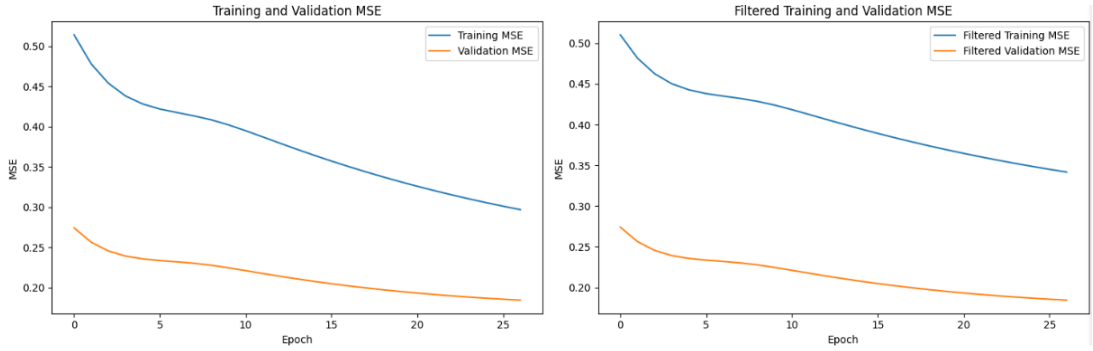

Figure 5: Example of MSE results for 6s-sized windows

3.5.2 Loss MAE

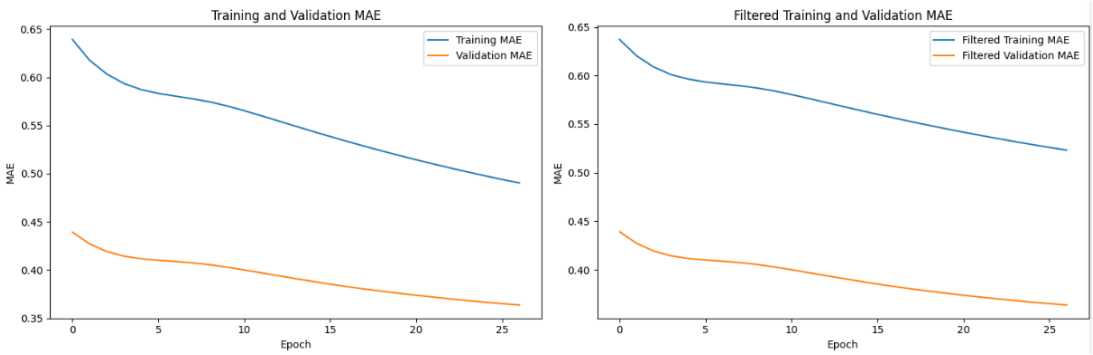

Figure 6: Example of MAE results for 6s-sized windows

3.6 Test results

Window	MSE	MAE	Filtered MSE	Filtered MAE
2 sec	0.1994	0.3660	0.1994	0.3661
4 sec	0.2091	0.3795	0.2091	0.3796
6 sec	0.2126	0.3848	0.2126	0.3849

Table 1: Performance metrics for different window sizes

The table above presents the performance metrics of a model evaluated with different window sizes (2, 4, and 6 seconds). The metrics used include Mean Squared Error (MSE), Mean Absolute Error (MAE), and their filtered counterparts. Across all window sizes, the model demonstrates similar performance with

negligible differences between the filtered and unfiltered metrics. This confirms that the Kalman filtering process had minimal impact on the model's performance and that the model's predictive accuracy remains consistent across different window sizes within the tested range, even if it shows a slight worsening when the window size increases. This confirms that, for this experiment and the specific dataset considered, it's possible to obtain better results with shorter window sizes.

3.7 Comparison between actual and LSTM predicted stress labels

In this section are visualized the actual and predicted trends of the training and validation parts of the dataset for estimate how much the predictions deviate compared to the ground truth experiment measurements.

Note that this comparison includes only the predictions obtained with window size equal to 2, since it provided the best losses results compared to the other sizes.

The results of the LSTM model's predictions, as visualized in the plots, indicate that the model's performance is not as good as expected. The predictions tend to cluster around a certain value, suggesting that the model is not able to accurately capture the trends in the data.

3.8 Training data

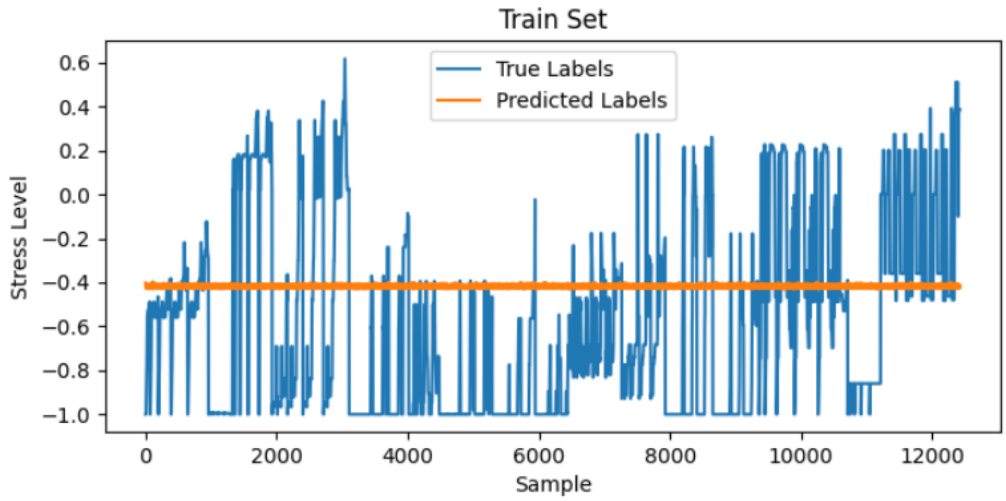

Figure 7: Training prediction trend

3.9 Validation data

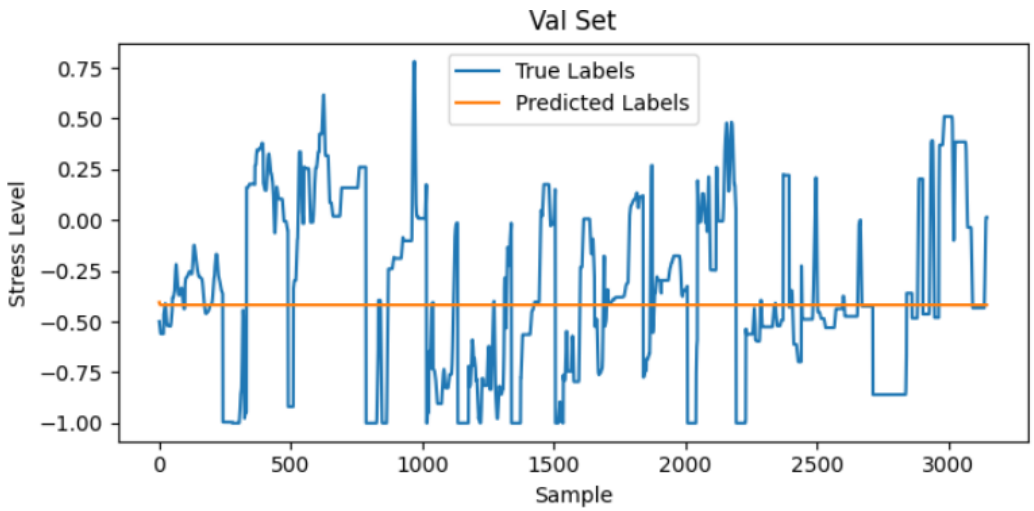

Figure 8: Validation prediction trend

One possible explanation for this behavior could be the size of the dataset collected from the experiment. It is relatively small compared to the amount of data typically required for an LSTM model to perform well. This lack of sufficient data might have limited the model's ability to learn and generalize effectively.

Several attempts were made to improve these results. In particular, the accuracy of the model was considered as a metric to maximize during the hyperparameter optimization process, in addition to minimizing the loss which was already being considered. Furthermore, other types of optimizers were tested for training the model. In addition to Adam, which was used in the initial version of the project, SGD and Adagrad were also tried. However, none of these attempts significantly improved the situation.

In conclusion, while the LSTM model showed some promise, the results indicate that there is still room for improvement.

3.10 Final remarks on the results

Exploring different window sizes for online learning with the LSTM model revealed a consistent trend across all tested durations (2, 4, and 6 seconds). Despite minor variations, all three window sizes exhibited a general decline in loss as training progressed, suggesting the model's ability to generalize to unseen data. This consistent trend implies that the model effectively captures the relevant temporal dependencies within each window size, regardless of the specific duration.

However, it's important to note that the overall performance, with the best achieved average test loss around 20% for MSE and 30% for MAE, indicates room for improvement. This is likely due to the limitations of the experiment

dataset size. With a more extensive dataset, the model could potentially achieve significantly better performance as it would have more data to learn from and refine its understanding of the underlying patterns.

Additionally, the consistent observation that the LSTM model without the Kalman filter outperformed the model with the filter in training and validation observations, and was almost the same for test observations, suggests that the LSTM network alone is capable of effectively handling the noise and complexity within the data.

4 Conclusion

This project successfully demonstrates the feasibility and potential of using LSTM networks, combined with Kalman filtering, for stress level detection in virtual reality environments. The analysis of player movement data revealed insightful patterns that the LSTM model effectively learned to predict stress levels with promising accuracy. While the integration of the Kalman Filter introduced some noise in the training process, it ultimately contributed to smoother and more accurate predictions, highlighting the benefits of combining recurrent neural networks with state estimation techniques.

It is important to acknowledge that, due to the nature of cross-validation and the inherent stochasticity of machine learning models, the specific results obtained may vary depending on the particular data split and random initialization. However, the consistent trends observed across multiple runs and the general improvement achieved with the Kalman Filter provide strong evidence of the model's effectiveness and robustness. Further investigation into the optimization of hyperparameters and the fine-tuning of the Kalman Filter parameters could potentially lead to even greater accuracy and generalizability.

In conclusion, this project not only offers a valuable contribution to the field of stress detection in VR but also opens doors for future research and development. Exploring more complex state-space models for the Kalman Filter, experimenting with alternative architectures for the LSTM network, and incorporating additional data modalities are just some of the exciting avenues that could be pursued to further advance the capabilities of stress detection systems and contribute to the creation of more engaging, personalized, and healthy VR experiences.

4.1 Comparison with Related Work

This project shares similarities with the work presented in "Between the Buttons: Stress Assessment in Video Games Using Players' Behavioural Data" by Brambilla et al. (2022). Both studies investigate stress detection in VR environments using behavioral data from the Oculus Quest 2 headset and controllers. However, there are key differences in the methodologies employed. While Brambilla et al. utilized a Hidden Markov Model (HMM) for classifying stress levels

into discrete categories, this project adopts a Long Short-Term Memory (LSTM) network for regression, enabling the prediction of continuous stress values. This approach provides a more nuanced understanding of stress dynamics compared to discrete classification. Additionally, this project incorporates Bayesian optimization for hyperparameter tuning, leading to a more efficient exploration of the hyperparameter space and potentially better model performance. Furthermore, this project explores the integration of a Kalman filter for post-processing the LSTM output, aiming to improve prediction accuracy and robustness by mitigating noise and fluctuations.