A Method For Active Motion Sickness Reduction Using Predictive Models

---

A Thesis

Presented to the

Faculty of

Wentworth Institute of Technology

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Applied Computer Science

---

by

Jacob L. Ledbetter

Spring 2024

WENTWORTH INSTITUTE OF TECHNOLOGY

The Undersigned Committee Approves the

Thesis of Jacob L. Ledbetter:

A Method For Active Motion Sickness Reduction Using Predictive Models

Yetunde Folajimi, Chair
School of Computing and Data Science

Micah Schuster
School of Computing and Data Science

Lauren Melfi
School of Computing and Data Science

DEDICATION

Dedicated to my grandmother, may she rest in peace.

ABSTRACT OF THE THESIS

A Method For Active Motion Sickness Reduction Using Predictive Models
by
Jacob L. Ledbetter
Master of Science in Applied Computer Science
Wentworth Institute of Technology, 2024

Motion Sickness is a common problem in Virtual Reality (VR). It is often described as discontinuity between seen motion and motion felt by the vestibular system, though multiple other factors play a role. While much research has been done to understand why motion sickness occurs and when it is happening in the user, little has been done to actively prevent motion sickness as or before it occurs. We use a machine learning algorithm to detect when a user is likely experiencing motion sickness. Once motion sickness has been detected, corresponding mitigation features such as vignetting, snap turning, or user warnings are enabled on the user's behalf. We found that this method of active motion sickness mitigation is feasible. We also found that consideration must be taken into the performance impact that this active mitigation system may have on the VR system.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

PAGE

LIST OF FIGURES

PAGE

ACKNOWLEDGMENTS

I would like to thank Dr. Folajimi for helping find articles I could not during my lit review.

I would like to thank Dr. Schuster for guidance, assistance with LaTeX, and being a shoulder to cry on when everything goes wrong.

I would like to thank Dr. Melfi for guidance with some complicated matrix transformations.

Lastly, I would like to thank members of the ChilloutVR Modding Group Discord server for helping with C# issues and pardoning some truly terrible code.

CHAPTER 1
INTRODUCTION

Virtual Reality (VR) is a popular medium for user interaction. Its primary feature is the use of a head mounted display(HMD) that allows the user to see a fully rendered virtual environment. Such a display blocks the user's vision of the real world. In addition to the HMD, the user has controllers allowing them to move through and interact with the environment. The HMD and controllers have their position and rotation tracked in physical space. The tracking data creates a tracking space that maps real world objects to objects in the virtual environment. VR is used in several contexts from telepresence[1], to simulation and training[2], to entertainment[3].

1.1 Problem Statement

A common problem in VR is that of motion sickness. Motion sickness includes symptoms like nausea, fatigue, and upset stomach, among others. This most often occurs in VR when the user is moved in the virtual environment while the user is stationary in reality[4]. Examples include moving using the controllers, standing on moving platforms, or riding vehicles.

Many VR applications are aware of these issues and have implemented some features to prevent motion sickness. One common option is to apply a vignette[5] to the user's vision when they are moving. Another option is to change how the user moves, either by letting the user select an area or moving an avatar in third person to an area, and then teleporting the user[6]. The last common option we explore is to snap the user's rotation a set increment rather than smoothly turn them[7].

Few solutions currently exist to dynamically reduce motion sickness for VR users based on the user's level of motion sickness, and we have found none that do so without any user input.

1.2 Literature Review

The majority of solutions to reduce motion sickness are either user toggleable features, or attempts to reduce latency within the VR system[8] (where latency can cause unexpected environment changes and thus sickness). User toggleable features either require the user to toggle them before or after becoming motion sick. Decreased latency helps in many cases, but is not the only cause of sickness. Many studies have been done to ascertain why people become motion sick in VR. One paper found that

"low FPS, low refresh rates, and realistic pictures lead to high probability that players feel uncomfortable; and fighting and shooting games also tend to make players uncomfortable"[9]. Another found the environment chosen, and its theme or mood, had a significant impact on the user's motion sickness[10]. A third paper lists multiple factors, such as seen movement, HMD field of view(FOV), flickering/stuttering, and whether the user has a fixed point of reference[11]. Some have found that 360°treadmills can provide relief[12] by linking seen movement to felt movement, but they are not economical for most users, and do not solve all potential motion sickness triggers. Some have also found that visual cues can reduce motion sickness[13] as they provide a frame of reference for the motion, but if unsubtle, they may degrade the user experience for those who do not need such cues. Some have tried to dynamically reduce motion sickness by controlling motion sickness mitigations based on how frequently a user clicks a button within some timeframe[14]. However, this requires active input from the user and cannot prevent motion sickness before it occurs.

A variety of models have been made to predict motion sickness in the user. Some use Deep Neural Networks (DNNs)[15], and have found moderate results in detecting nausea. Some use more classical Machine Learning (ML) approaches[16], also finding moderate success in detecting motion sickness in non-extreme situations. Finally, some use Convolutional Neural Networks (CNNs)[17], finding better accuracy than classical machine learning methods. Most models predict using a combination of data from the application as well as user submitted feedback, but others predict using the users themselves from external sensors, specifically user eye movements[18, 19]. Some tools have even been made to help in the creation of predictive models[20], making it easier to create new motion sickness datasets. However, none of these models have been used to drive the reduction of motion sickness, they only predict when/if the user is getting motion sick.

1.3 Objectives

Our goal in this thesis is to create a model that can predict a user becoming motion sick and find the most applicable motion sickness reduction feature. It should then activate the motion sickness reduction, and subsequently deactivate it, once the user is no longer at risk of motion sickness. As stated in section 1.2, there are many models present for predicting motion sickness, thus many options to consider for what kind of model to use.

CHAPTER 2

METHODS

Our approach used a CNN for motion sickness prediction and common motion sickness prevention techniques in an example environment.

2.1 Motion Sickness Detection with Convolutional Neural Networks

A Convolutional Neural Network is a type of Neural Network that involves convolutions in its processing. In a neural network, input values are fed into "neurons" which then produce an output value. A neuron passes the weighted sum of each input through an activation function, which determines its output value. Activation functions are typically non-linear, differentiable, and monotonic. There are many different activation functions e.g., sigmoid, Rectified Linear Unit(ReLU), Hyperbolic Tangent. Groups of neurons are arranged into layers, and each layer tends to have its outputs fully connected to the neurons of the next layer.

Convolutional Neural Networks vary by changing how the neurons are layered and how they receive input. Each layer contains a set of filters. Each filter, also known as a kernel, will be centered on a location in an array of values. The filter takes the weighted sum of its center and adjacent values in the array and puts them through an activation function like the neurons described earlier. What values in the array are adjacent are determined by the kernel size. The array and therefore the kernel can be more than one dimensional. After the filter computes its output, that output is considered the output for the location in the array the filter was centered on. The filter is then centered, and its output is gotten for each position in the array, creating a convolution. This is done for all filters, creating a new array representing the convolved outputs of each filter. This means convolutional layers can capture the spatial relations of values in the array, like images, or values over time. By adding more convolutional layers, more complex patterns can be detected. For example, one layer may find groups of pixels representing straight lines at various rotations in an image, and the next layer may find arrangements of those lines in the shape of a box. As well, CNNs are typically designed to include some sort of pooling, where a smaller array is made from the output of some function applied over the array. This usually takes the form of taking the maximum or the average of a number of elements in the array, also determined by a

kernel size. For example, an image can be pooled by taking the average of all two by two blocks of pixels, making a new image that is a quarter of the original size. Figure 2.1 shows a simple diagram of a CNN[**?**].

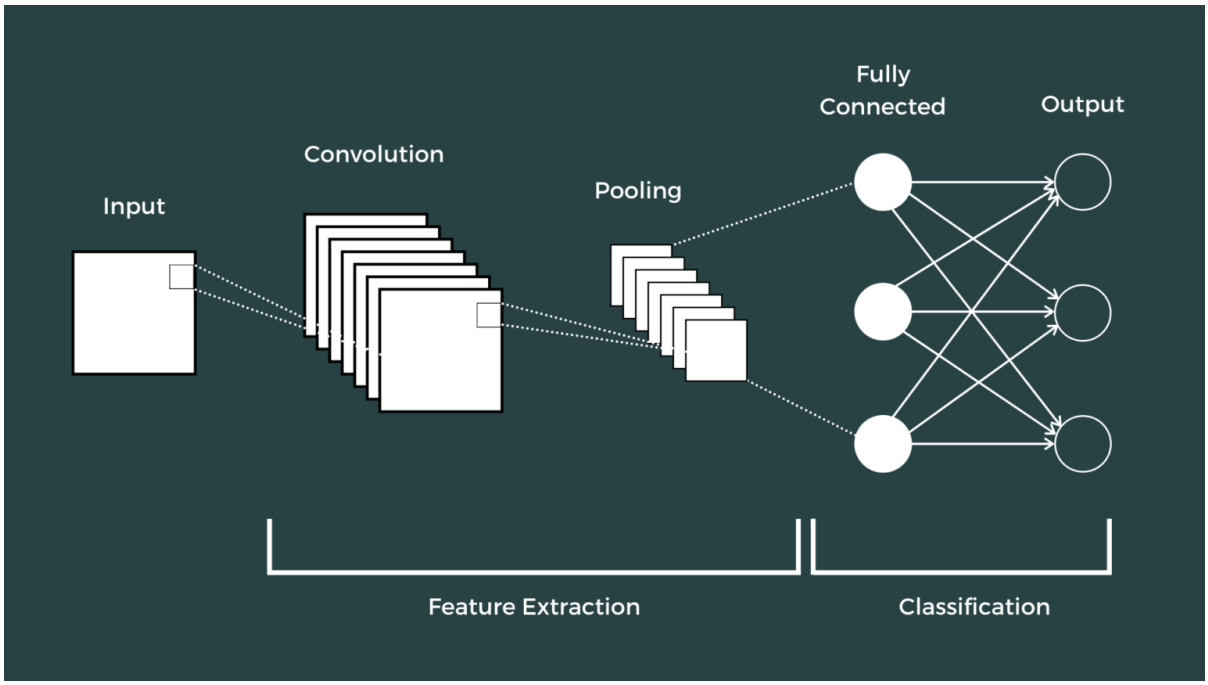

Figure 2.1. Simplified diagram of a Convolutional Neural Network from theclickreader.com

We believe a CNN would be most useful as it can take data from the viewpoint of the user as well as other data, e.g. controller, motion, and pose data, from the user. Like a neural network, a CNN can take in a wide array of information at once, and can be made to categorize into multiple motion sickness categories, causes, or ratings. More importantly, CNNs can convolve over a window of time to provide a more accurate assessment of the user's sickness as it changes over time.

2.1.1 Dataset

To train a CNN, we need a dataset of VR use with a scoring of how motion sick the user was. For this, we used the VRNet Motion Sickness dataset[21]. The dataset contains a set of recordings for multiple VR games. Each recording has a set of frames (images from the HMD screens, labeled as `s<frame number>.jpg`), their motion vectors (movements of blobs of pixels, labeled `m<frame number>.zfp` in the zfp format) and depth (distance from the virtual camera to a virtual object, per pixel, labeled `d<frame number>.jpg`). Each recording also has csv files with data for the

camera, controller inputs, controller pose, in-game objects, and lights for each recording. As well, each recording has a "voice" csv file containing the motion sickness ratings given by participants during the recording.

The dataset comes as a large collection of nested zip files, some are unorganized recording folders while others contain folders for each game. The game folders contained multiple csv files for each recording and maybe some recording folders. This meant that once all the zip files were decompressed, files needed to be sorted into their respective recording folder and each recording folder into a respective game folder. A fair amount of time was also spent making sure file names matched a specific uniform pattern (<game name>/P<participant number>VRLOG-<recording timestamp>/<file name>).

We did not use the lighting data as this tends to remain static in most scenes. Depth data was left unused to reduce complexity. The camera data is the projection and view transformation matrices of the cameras in the scene. These 4x4 matrices represent how to transform a vertex position from within the VR environment to the 2d image shown to the user. The controller data shows what values are received from the joysticks on the controllers. The pose data shows the real world position, rotation, velocity, and angular velocity of the HMD and controllers. The motion vector data could not be decoded from their compressed zfp format as file headers were either corrupt or not included, and the compression parameters could not be found.

The dataset images are inconsistent from game to game. Some games have both eyes' perspective, while others only have one eye's perspective. While all games have their depth images flipped upside-down, some additionally have their frame images flipped. The images are quite large for a CNN with 2016 by 1042 pixels and three color channels. The size is relevant as larger images make the CNN have to perform more calculations and require more filters/more pooling to generalize effectively. Therefore, CNNs are usually designed to work with smaller images. Using Pillow(a Python library), we flipped images and resized them to half-resolution(1008 by 521 pixels). We decided to copy images with only one eye's perspective to make them appear to have two eyes and fit the full resolution of other images.

The motion sickness ratings for each recording were given as the participants in the VRNet dataset recorded them and were not available for each frame of the recording. To account for this, each rating was attached to the nearest recorded frame by its timestamp. For each recorded frame lacking a rating, a rating was generated by linearly interpolating between the two ratings surrounding it and rounding to the nearest whole number. For recorded frames before the first user rating, they were set to the minimum sickness (1) as the participants in the VRNet dataset started recordings

not sick. For recorded frames after the last user rating, the last user rating was used as the value. This interpolation step was preformed for each recording and written to a 'voice_preproc.csv' file per recording.

We removed many columns from all the csv format files as many were duplicates, only zero, or would have little impact on the model. Some columns contained multiple values per row and were split to a column per value. Data for each device(HMD, left-hand controller, right-hand controller) was recorded as separate rows per frame. To have one row of data equal to one produced frame of application runtime for each file, all device values for a single frame were combined into one row where each value was given its own column. Camera transformation matrices were listed for all cameras in the scene, so rows not related to the main camera used by the user were removed. The frame images are referred to as "image" data throughout the rest of this paper, while all other data (controller input, controller pose, camera transformation matrices) are referred to as "numeric" data.

2.1.2 Network Architecture

Two small models were created, one to handle image data and one to handle numeric data. These models were then combined into one larger model. Both models were made using the TensorFlow and Keras Python libraries for machine learning.

The numeric model is a Convolutional Neural Network that takes 20 instances of numeric data. Each instance consists of 116 values in IEEE 32-bit floating point format. Thirty-two values are dedicated to the main camera's transformation matrices. Thirty values are dedicated to the 5-axis (with two dimensions per axis) input data from the HMD, left-hand controller, and right-hand controller. Lastly, 54 values are dedicated to HMD, left-hand controller, and right-hand controller's position, rotation, velocity, and angular velocity within tracking space. This input is passed through a convolution over the window of time using Keras's Conv1D layer. The convolution passes 64 filters over the 20 time steps where each filter is three time steps wide. Each Conv1D layer is followed by a BatchNormalization layer that enforces that the outputs of the Conv1D layer have a mean of 0 and a standard deviation of 1. This increases network performance as most Neural Networks(NN) expect to work with normally distributed values. Following batch normalization is a layer representing the neuron activation function, of which ReLU was chosen. ReLU is quite common for NNs and was chosen as such. The convolution, batch normalization, and ReLU activation layers are repeated twice for a total of three convolutions. To convert the filters from convolution to a classification, the output values from each filter need to be pooled together. This pooling is provided by a GlobalAveragePooling1D layer. Lastly, there are five densely

connected neurons that will output the likelihood the user is a given level of motion sick from one to five, and as such, use the "softmax" function for activation instead of ReLU.

The image model is very similar to the numeric model. Its convolutions occur on two-dimensional images over the time window and as such, use a Conv3D layer. The convolutional layers use 64 filters with each filter spanning three time steps and three pixels on the x-axis and y-axis of the image. The global average pooling layer also becomes a GlobalAveragePooling3D.

Both models are combined into an ensemble model. This was done as we only want one prediction of the user's motion sickness, but we have two different types of data each needing to be handled separately. This was done by connecting their output into a Concatenate layer and a few densely connected layers. The Concatenate layer puts the output of two layers into a flat list of inputs (being 10 as each model predicted a ranking one through five). The densely connected layers had 32, 16, and 8 neurons respectively using ReLU as activation. These were added to let the combined model learn the shortcomings of the separate models and adjust its prediction to something more accurate.

By learning the trends of the smaller models and adjusting the prediction, this model is an ensemble of other models. The combined model ends with a final densely connected layer like the final layer of the numeric and image models. The combined model is represented visually in Figure 2.2, with its various inputs, layers, and output.

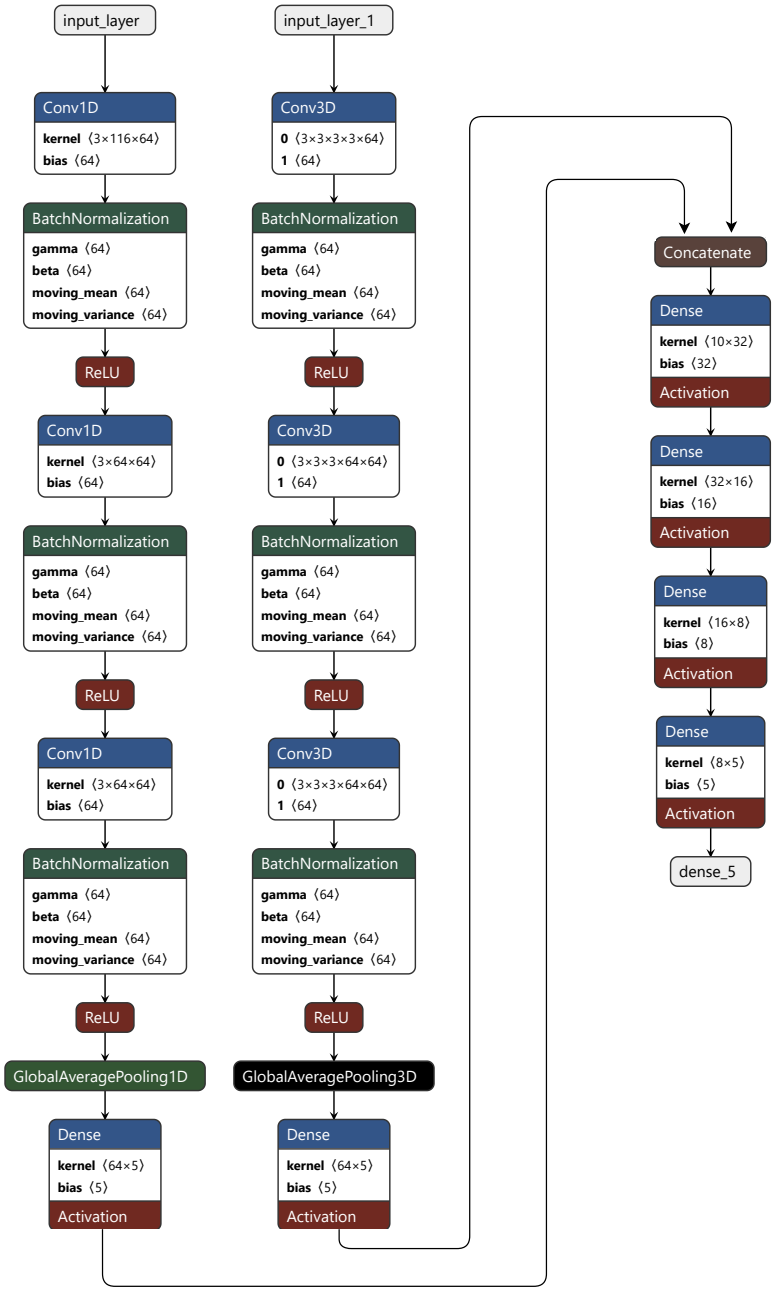

Figure 2.2. Diagram of layers and hyperparameters of the combined motion sickness detection model.

2.1.3 Training

The model was compiled using the Adam algorithm for gradient descent and uses categorical cross-entropy as its loss function. After running over the training data, the model would perform inference over the validation or testing data and get the accuracy and loss during the test. During training, the model was set up to reduce its learning rate (LR) by half when the validation loss plateaued for 20 epochs to a minimum LR of 0.0001. It was also set up to end training early if the validation loss plateaued for 50 epochs and would log the loss(train and validation), accuracy(train and validation), weights, among other things for each training run. Lastly, it would save the weights that had created the lowest training loss to a file at the end of each epoch.

First, the image and numeric models were trained for 100 epochs maximum to get a clue as to how each separated model performed and to see if they worked properly or needed tweaking. After this preliminary test, the combined model was trained for a maximum of 250 epochs. Two hundred fifty epochs were chosen because it was assumed to be around a few hours rather than a few days. Since the model has two inputs, the numeric data and the image data, both need to be zipped together for each row. Every 20 rows of the dataset is a time-period, and is given a rating that is the average of all ratings in the window, and represented as a one-hot vector. Twenty rows per time-period was chosen as this represents one second of application runtime with the application running at 60 frames per second and only every third frame being recorded in the dataset. The dataset, as a set of sequential time-periods, is split into a training and testing/validation set, where the split is 80%/20%. Time periods were batched together in sets of 2 to make training faster. Typically, with time series data like this dataset, each time period would be somewhat dependent on the past time-period. This is still somewhat true, the past time-periods have little impact on the current period. Additionally, by leaving the periods in order, the difference between any two periods while training will be small and so the model will not make larger adjustments to improve its accuracy. Considering this, we shuffled the order of the periods after they were split. Only the training split was shuffled. Lastly, the resolution of the frame images was cut to a fourth as the dataset was loaded to stay within the available system memory of the training machine. This leaves the images with a resolution of 256 by 131 pixels.

2.2 Motion Sickness Prevention In An Application with Vignetting, Snap Turning, and User Warnings

To apply our model, we created an application that would likely cause motion sickness and employed several standard methods for reducing that motion sickness based on the inferences of the model.

2.2.1 The Environment

The application was made using the Unity game engine as well as Unity's XR Interaction Toolkit[22]. It features two roller-coasters the user can ride, one being a simple circle and the other being more complex. The user had both continuous and teleport locomotion methods for moving around the environment. The user enters roller-coasters by pointing at and clicking a button in the environment, after which they will be seated in the roller-coaster's car and will ride along the track. Afterward, they will be dismounted from the ride and able to walk around again as well as select another coaster to ride. The environment contains some hills made using Unity's Terrain Generation tools and realistic texturing received from `https://3dtextures.me/`[23], so the environment resembles those in the dataset instead of a completely flat environment. The environment is shown in Figure 2.3. Within this environment, three motion sickness reduction features were implemented.

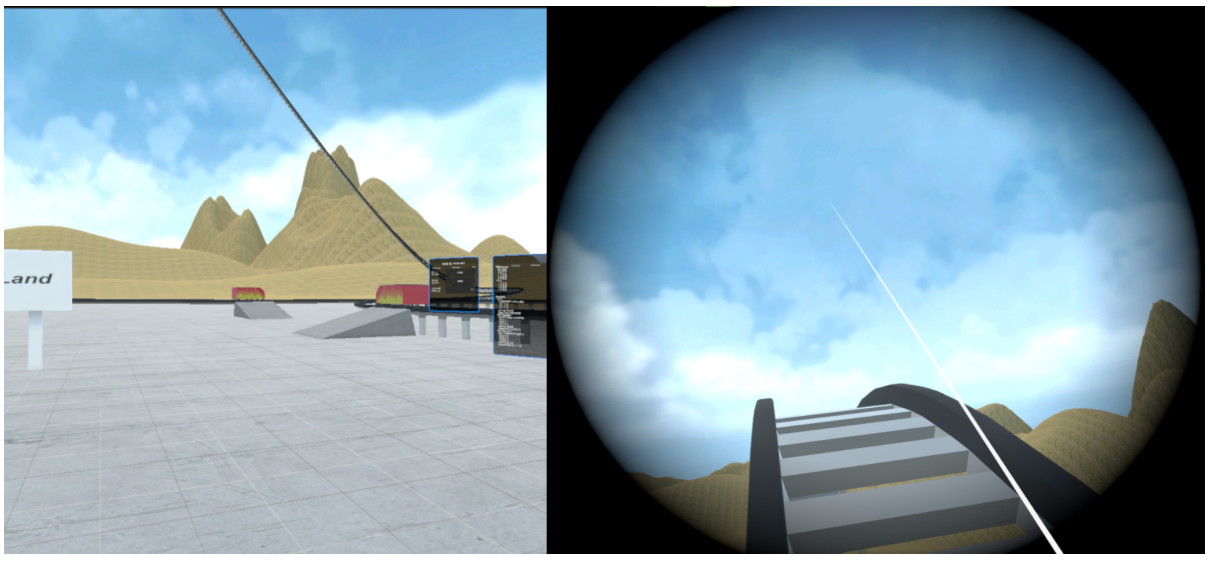

Figure 2.3. Two images of the virtual environment from our application: one from the ground(left) and the other on the complex roller coaster(right)

2.2.2 Vignetting

Vignetting is the term for reducing the user's FOV with a visual filter. The edges of the image become darker while the center of the image is unaffected. This

removes visual context suggesting to the user that they are moving, which would cause motion sickness. Typically, this is done in games based purely on whether the user is locomoting or not. Locomoting is when the user is moving in the environment, usually with the controllers, without moving physically. In this application, the size of the vignette was determined by the rating provided by the model, where the sicker the user was, the larger the vignette and the less of their periphery the user could see. The vignetting was provided by Unity's Tunneling Vignette Shader[5] and examples of our vignetting are shown in Figure 2.4.

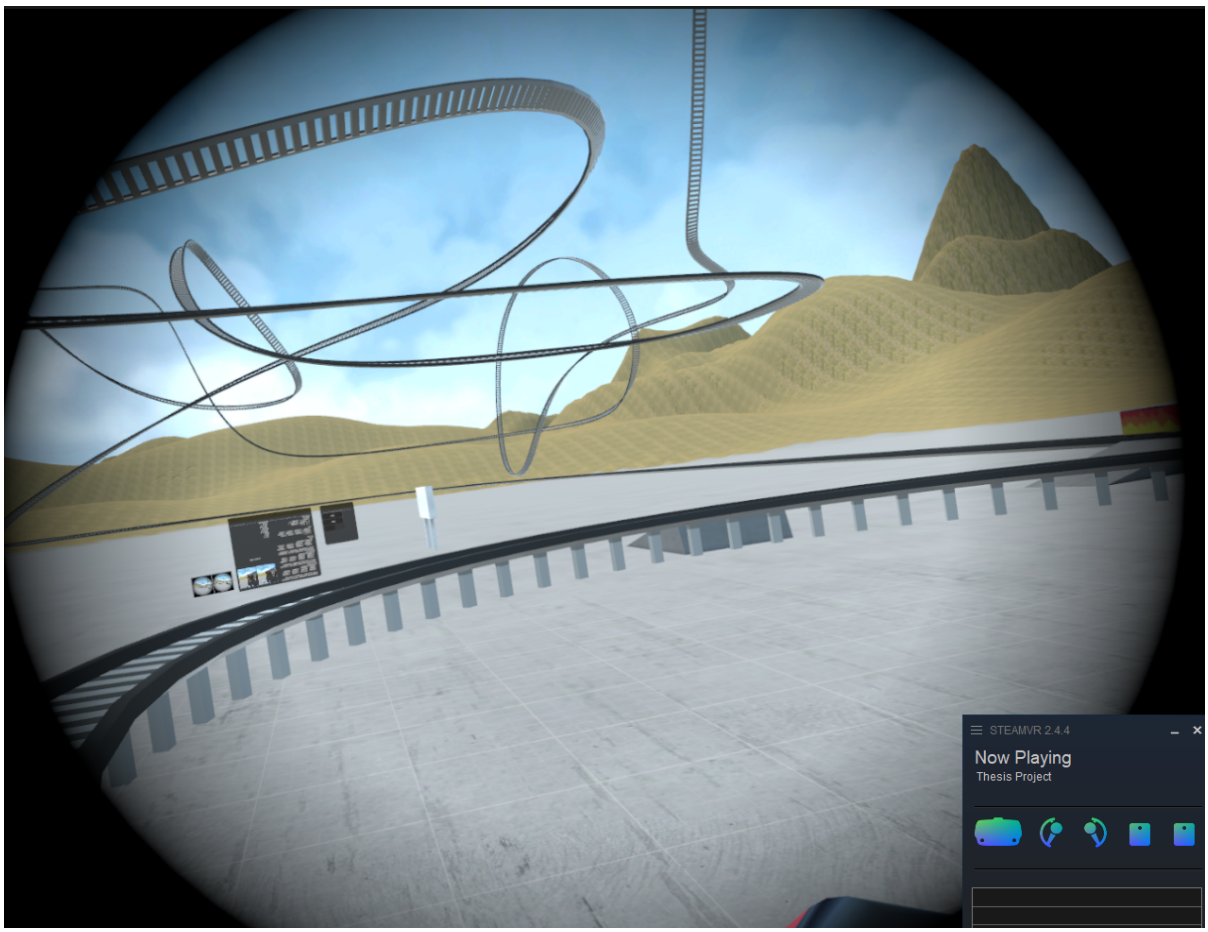

Figure 2.4. The perspective of the user with a black vignette reducing their field of view.

2.2.3 Snap Turning

Snap turning refers to larger user rotations that occur instantaneously with a single input. For instance, the user would press right once on the joystick to "snap" 45° instantaneously. This is contrasted by the normal method of holding the joystick

right to turn at a constant velocity over time. This reduces motion sickness by reducing the motion seen but not felt by the user as they rotate over large increments instantly and simply rotate their head for smaller increments. By default, the application uses continuous turning, unless the user has a motion sickness rating greater than two from the model, where they then use snap rotation at increments of 45°.

2.2.4 User Warnings

Some applications will warn the user to take a break if they have used the application for a certain amount of time. Our application will display a warning if the model has rated the user as having a motion sickness of five (which is the maximum) for thirty seconds. The warning (seen in Figure 2.5) tells the user to take a break and is attached to the user's vision. The warning is removed after five seconds with a lower motion sickness rating from the model. This is meant as a last resort if nothing can reduce the users' detected sickness as to minimize the effects of prolonged sickness.

2.3 Connecting Detection and Prevention

Once the application was built and the model was trained, the two had to be combined.

2.3.1 Pulling the necessary data for inference

Pulling controller inputs and camera matrices was easy as Unity provided functionality to access this data, the device pose was more complicated. The dataset and therefore the model had recorded the position and rotation of each device in a 4x3 transformation matrix. Unity, however, provides the position as a vector and the rotation as a quaternion[24]. Unity provides a function for retrieving a transformation matrix from a translation(position), rotation, and scale(which would remain as $< 1, 1, 1 >$). This function returns a 4x4 matrix, however; the last row is always $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$. This row exists to make sure the matrix is square for other operations, though this is not relevant here as the first three rows had the information we needed. Given our needs, the last row was discarded to create the 4x3 matrices for the poses.

While originally the numeric and image data from the application would be pulled each frame, this approach introduced massive performance losses. This is because of how the frame image was represented. Two images were pulled from the virtual cameras representing the left and right eye's perspective. They were then attached in one image as a Unity Texture2D. This datatype requires a texture format, and while many are available, none were quite suitable. The model is trained on IEEE

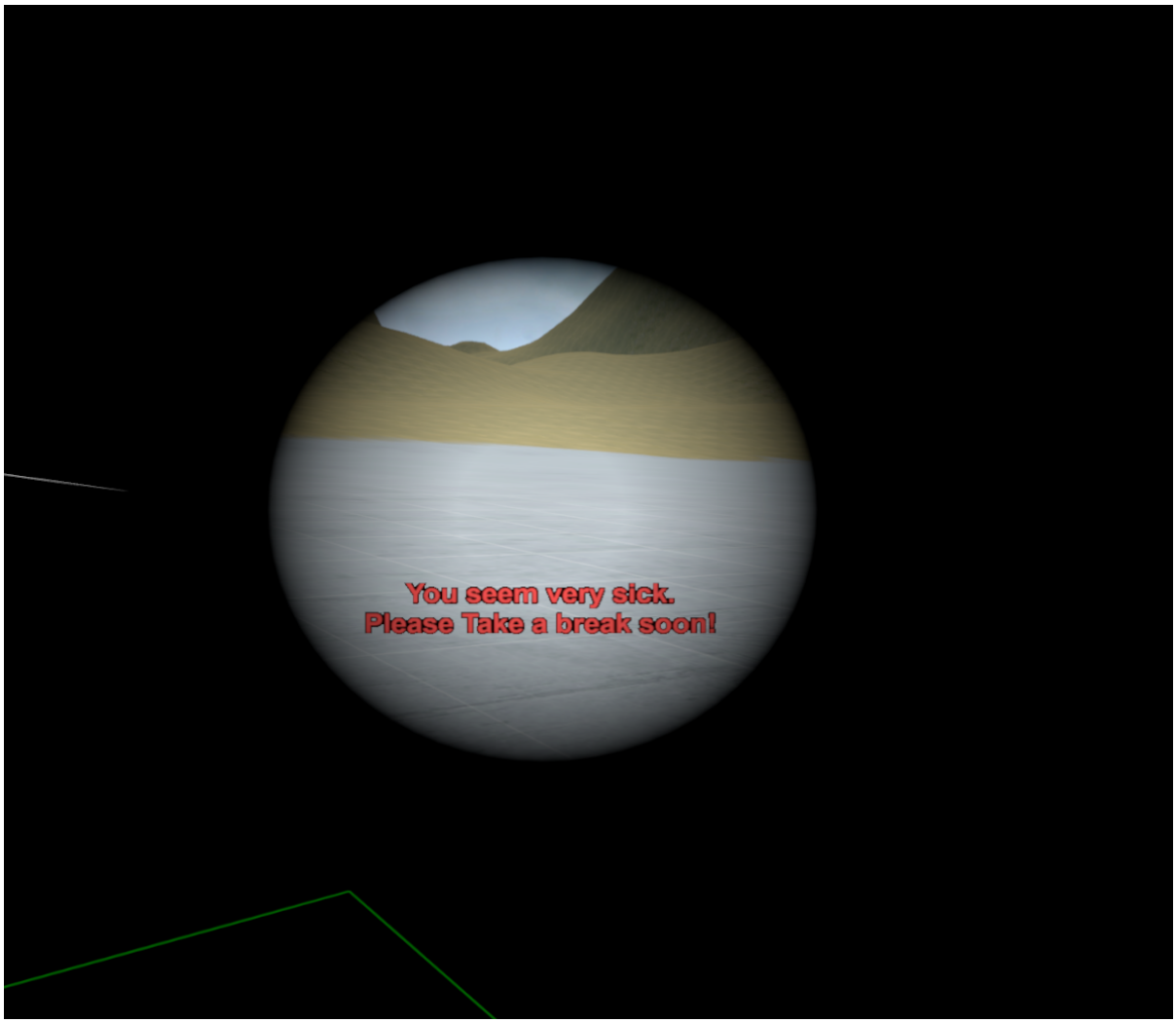

Figure 2.5. The perspective of the user with red text near the bottom, stating "You seem very sick. Please take a break soon!"

floating point numbers, but most formats were either integers, or compressed formats that would have required decompression. Of the floating point formats available, the options were RFloat, RGFloat, or RGBAFloat. The first two formats lack the blue channel the model needs for inference while the last has an additional channel(alpha) that needed to be removed. To remove the alpha value from the image, we iterated over each value in the image copying it to a float array unless it was the alpha value. Since iterating over the image, meant iterating over $256 * 131 * 4$ or 134144 values, this took considerable time, around 60ms. However, frames need to be generated in around 18ms and this code would have otherwise held up the main thread, reducing the framerate to around 15 frames per second, which would make the user more motion sick among other things. This required moving the logic into another thread; using Unity's Jobs

system. Jobs maintain thread safety by requiring that the user only work on memory explicitly allocated for the job. This meant that before starting a job, all data needed for the job would have to be polled, then copied to the job's own memory; then the job could be run. When the job finished, the data would be copied back from the job's memory back to the main processes' memory.

This culminated in putting together the last one-hundred frames of numeric and image data to be processed in a job. First, the current time-period would be sent to the job with the new numeric and image data from the current frame. Jobs also only work with arrays because of specific memory allocation constraints. Therefore, the job would copy the latest 99 frames from the end of the array containing them to the beginning, and would copy the newest data to the end. For numeric data, this simply constituted copying from an array containing that data. For the new image, each value was iterated through to only copy the red, green, and blue values, but not the alpha values. Then the job would end and the new data would be copied back. This process alone would take two or three frames worth of time, meaning the time-period only polled every second or third frame, which matched up with what the model was trained on. This was only to process data to be given to the model.

2.3.2 Loading the Model in Unity

The model was originally planned to be loaded directly from within Unity. This would have been possible as Unity uses the C# programming language, and Microsoft provides a framework for performing ML tasks in their aptly named ML package. The major benefit of loading the model within C# is that it's much more performant. This is because moving data into the model is fast and the ML library can more easily make use of the GPU for predicting than TensorFlow can while within Windows. Inferencing using the GPU is important as, compared to the CPU, it is much better at handling matrix math and manipulations which primarily make up the computation of CNNs. Unlike Keras, Microsoft's ML package uses a different model storage format and runtime named Onnx. Converting a Keras model to onnx is relatively easy. Despite many attempts, the onnx runtime would not function properly. We then decided to try something else due to time constraints.

2.3.3 Using Inter-Process Communication Instead

Without the Onnx runtime, we had to use a different method to load the model to predict motion sickness. Since the model would load in Python, we decided to load it in Python using TensorFlow and Keras as we did for training. This necessitated a new

process to run the model, separate from the C# process that gathered the data for inferencing. This required the use of interprocess communication, specifically sockets.

Sockets are used to send data between processes, typically over a network. Each process creates a socket; one process is a server waiting for clients to connect, clients then connect and send data, the server sends data back. In this case, the Python process with the model would be the server and would have to take care to manage the connection received by the C# process. The Python server would wait for a connection, receive the data it needed for inference, get a rating for the user's motion sickness, and send that rating back to the client, and finally disconnect.

One would think to hold open the connection and let both processes talk until either ended. However, because getting GPU inferencing to work in Windows within TensorFlow is rather finicky, CPU inferencing, which is much slower, is needed instead. This slowness, like with the image iterations, would have caused issues with performance, so the C# process needed to communicate with the Python process through a job on a separate thread. Since jobs have their own segment of memory that cannot be shared with the main process, the connection cannot be passed between the main thread and the job. The job also can't hold onto the connection as its memory is deallocated when the job is complete, unless it's copied back to the main process, which couldn't be done. This necessitated creating a new socket connection to the Python server for every job (which occurs every 2 to 15 frames), after the processing of the new frame's data was completed. Once the inference was received in the C# process, the job would terminate and the prediction would be passed back to the main thread to be handed to the function that handled the motion sickness mitigations.

CHAPTER 3

RESULTS

The model was quite accurate on the testing portion of the dataset. The combined model attained 0.9803 categorical accuracy and categorical cross-entropy loss of 0.1689 after 146 epochs of training. Categorical cross-entropy is a loss function for categorical problems defined as $-log(P_{gt})$. P_{gt} is the model's predicted probability that a given time window belongs to the correct class. Like all loss functions, the closer to zero, the better. Figures 3.1 and 3.2 show the accuracy and loss of both training and validation across each epoch through the training process of the combined model. Figure 3.3 shows the learning rate of the combined model per epoch.

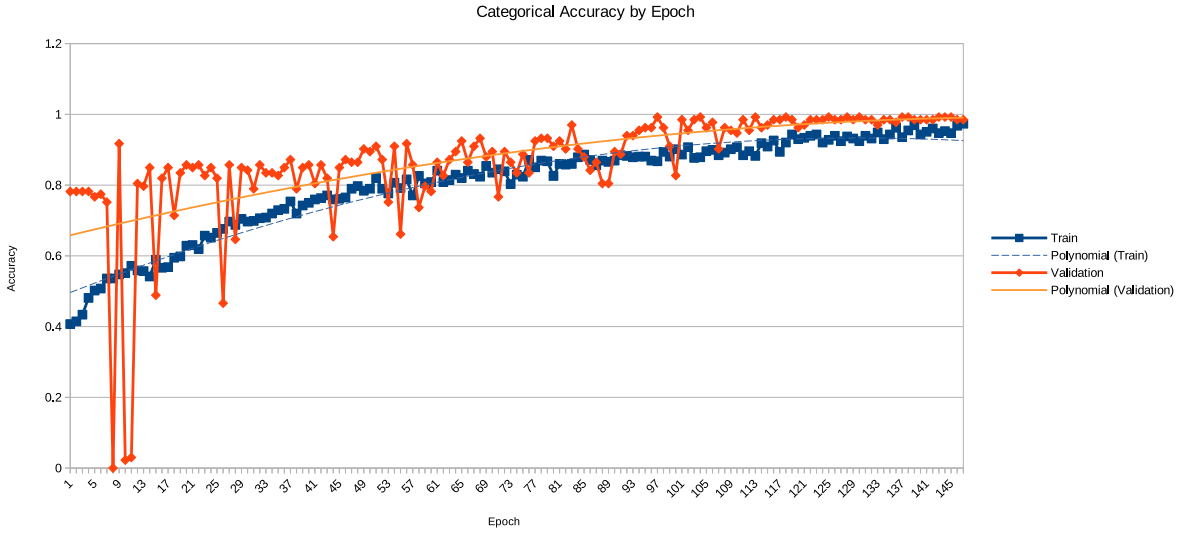

Figure 3.1. Training and validation categorical accuracy of the combined model per training epoch.

The only effective tuning done to the model was to increase the size of the time period as much as possible. The larger the time period is, the more data the model has to work with and the more accurate a prediction it can provide. The model's learning rate, or how much to shift the model's weights to minimize loss, was automatically tuned as stated in section 2.1.3 The model is not overfit or underfit. If the model were underfit, we would expect it to have poor accuracy in both training and testing. If the model were overfit, we would expect the training accuracy to be high while the testing

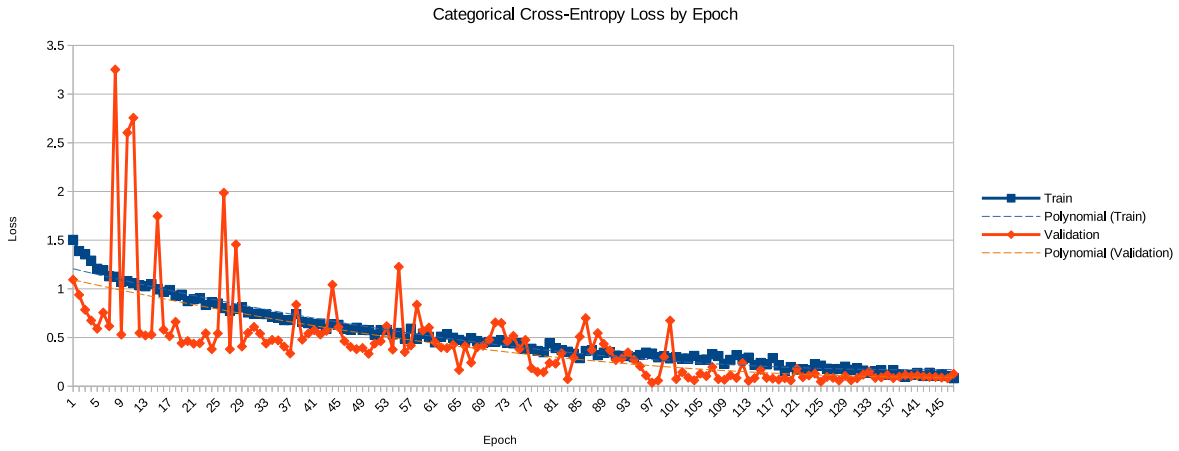

Figure 3.2. Training and valication categorical crossentropy loss of the combined model per training epoch.

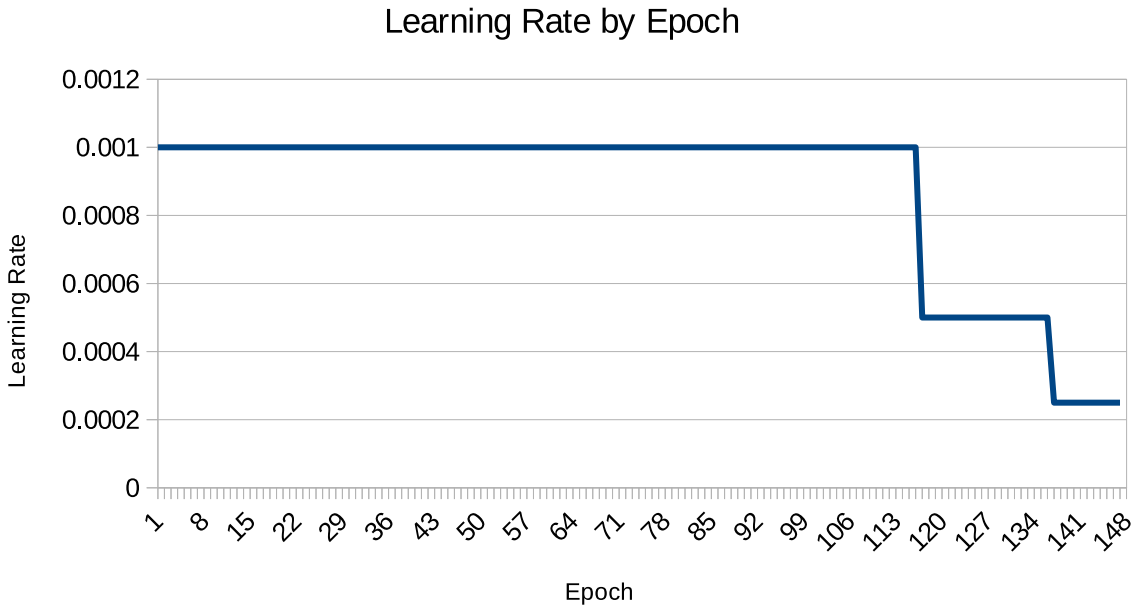

Figure 3.3. Learning rate of the combined model per training epoch.

accuracy was low. This is because as the model trains and improves accuracy on the training data, its lack of generalization would cause it to fail on the data it hasn't seen. The model, however, has a high training and a high testing accuracy. As well, the model's accuracy in training and testing increased as training went on, indicating some degree of learning. The testing accuracy varied wildly at first, but as training went on, the variance reduced significantly, indicating the tested accuracy is trustworthy.

The model's predictions apply decently well to a real-world application. The model rates most stimuli as a 2 (light sickness). This is the model making the assumption that the average user will be a little sick in VR, which the dataset supports with 2 being the most common rating. As the user starts locomoting, the rating rises between 3(somewhat sick) and 4(fairly sick). This behavior effectively re-creates locomotion activated vignetting, which is shown to reduce motion sickness. Turning using the joystick raises the rating to 3 at times. This behavior, if the rating were higher or the threshold for activating snap turning lower, is good for reducing sickness from turning. The simple roller coaster had little change in rating from the average. The simple roller coaster is unlikely to cause much motion sickness due to its speed and gentle, continuous turn. The complex roller coaster raised the rating between 2 and 4. The model seems to identify different parts of the coaster as causing different sicknesses. It seems to find the sections where the roller coaster climbs up slowly to be tame but sections with high speed and sharp turns to be more aggressive. The model's rating varied based on how rapidly the user followed the movement of the coaster with their head, where the less they looked around, the less sick they were predicted to be. The model takes some time between seeing motion sickness inducing stimuli and reacting. This delay varies from non-existent to 30 seconds. The model would not overreact to any particular stimuli, choosing to stay at a 2. Ratings of 1(no sickness) and 5(extreme sickness) were very rare. Table 3.1 shows ratings for a variety of stimuli when the user is standing or seated. This data was collected by viewing the application's reactions to our use of it.

Stimuli	Sitting Response (1-5)	Standing Response (1-5)
Still	2	2
Looking Around	2	2
Physical Walking	N/A	2
Continuous Locomotion	3-4	3-4
Turning	2-3	2-3
Riding Simple Coaster	2-3	2
Riding Complex Coaster	2-3	2-4

Table 3.1. Predictions by the motion sickness model on various stimuli. Responses from the model are from one to five being not motion sick to extremely motion sick.

The performance overhead of this system is significant. With the model and data formatting operations, the application takes 16ms to 40ms to produce a frame compared to a flat 16ms without them. This equates to a frame rate of 30 frames per second. This is not unusable but may cause more motion sickness, and for performance critical applications this may not be feasible to apply. This performance is in part due to bad implementation due to time constraints. Assuming the model was inferencing using the GPU and a better method of loading and passing data to the model were found, and less image processing had to be done, the overhead could be significantly reduced. These could easily be resolved with more time.

CHAPTER 4
CONCLUSION

In conclusion, using a CNN to manage motion sickness mitigation in a VR application is feasible given that the overhead from the model is taken into account. We used the VRNet motion sickness dataset and created an ensemble model that could predict with an accuracy of 98.03% on the held-out samples of the dataset. We created an example VR application with a somewhat detailed environment and motion sickness mitigations. When the model was put in control of those mitigations, it made decisions that were logical, though with some performance concerns.

4.1 Future Work

The first steps are to make prediction and collection of data from the engine more efficient. Beyond this, the model could be extended by adding "Long Short-Term Memory" (LSTM) layers so the model can make predictions on data before the given time-window. This would require not shuffling the trained time periods, which may reduce model accuracy but would yield better long-term predictions. There are other motion sickness mitigation features that could be implemented and controlled by the model. A dataset that could classify the user's motion sickness by cause could provide more specific mitigations by reducing the specific cause of motion sickness. Our model and more performant application should be tested on real users to see its real world efficacy. If all these things were done and the improved application was proven to cause less motion sickness, this system could be deployed in many applications to reduce motion sickness at the best time, before it happens.

BIBLIOGRAPHY

[1] S. D. Hoang, S. K. Dey, Z. Tučková, and T. P. Pham, "Harnessing the power of virtual reality: Enhancing telepresence and inspiring sustainable travel intentions in the tourism industry," *Technology in Society*, vol. 75, p. 102378, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0160791X23001835

[2] J. C. Meister, "How Companies Are Using VR to Develop Employees' Soft Skills — hbr.org," https://hbr.org/2021/01/how-companies-are-using-vr-to-develop-employees-soft-skills, 2021, [Accessed 18-04-2024].

[3] Apr 2016. [Online]. Available: https://store.steampowered.com/app/457550/Bigscreen_Beta/

[4] J. T. Reason, "Motion sickness adaptation: a neural mismatch model," *J R Soc Med*, vol. 71, no. 11, pp. 819–829, Nov. 1978.

[5] U. Techologies, "Tunneling Vignette Controller — XR Interaction Toolkit — 2.1.1 — 2.1," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.1/manual/tunneling-vignette-controller.html, 2023, [Accessed 03-04-2024].

[6] U. Technologies, "Teleportation Provider — XR Interaction Toolkit — 2.1.1 — 2.1," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.1/manual/teleportation-provider.html?q=teleport, 2023, [Accessed 03-04-2024].

[7] ——, "Snap Turn Provider (Action-based) — XR Interaction Toolkit — 2.1.1 — 2.1," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.1/manual/snap-turn-provider-action-based.html?q=snap, 2023, [Accessed 03-04-2024].

[8] R. K. Kundu, A. Rahman, and S. Paul, "A study on sensor system latency in vr motion sickness," *Journal of Sensor and Actuator Networks*, vol. 10, no. 3, p. 53, 2021.

[9] C. Zhang, "Investigation on motion sickness in virtual reality environment from the perspective of user experience," in *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 2020, pp. 393–396.

[10] U. A. Chattha, U. I. Janjua, F. Anwar, T. M. Madni, M. F. Cheema, and S. I. Janjua, "Motion sickness in virtual reality: An empirical evaluation," *IEEE Access*, vol. 8, pp. 130 486–130 499, 2020.

[11] H. T. K. Eunhee Chang and B. Yoo, "Virtual reality sickness: A review of causes and measurements," *International Journal of Human–Computer Interaction*, vol. 36, no. 17, pp. 1658–1682, 2020. [Online]. Available: https://doi.org/10.1080/10447318.2020.1778351

[12] X. Gong, "Research on minimizing VR motion sickness in VRChat," in *International Conference on Artificial Intelligence, Virtual Reality, and Visualization (AIVRV 2021)*, S. Chen, T. Li, D. Wu, and G. Gao, Eds., vol. 12153, International Society for Optics and Photonics. SPIE, 2021, p. 121530L. [Online]. Available: https://doi.org/10.1117/12.2626662

[13] Z. Qiu, M. McGill, K. M. T. Pöhlmann, and S. A. Brewster, "Display rotation for reducing motion sickness caused by using vr in vehicles," in *Adjunct Proceedings of the 14th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, ser. AutomotiveUI '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 25–29. [Online]. Available: https://doi.org/10.1145/3544999.3552489

[14] J. Won and Y. S. Kim, "A new approach for reducing virtual reality sickness in real time: Design and validation study," *JMIR Serious Games*, vol. 10, no. 3, p. e36397, Sep 2022. [Online]. Available: https://games.jmir.org/2022/3/e36397

[15] S. Hell and V. Argyriou, "Machine learning architectures to predict motion sickness using a virtual reality rollercoaster simulation tool," in *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2018, pp. 153–156.

[16] N. Padmanaban, T. Ruban, V. Sitzmann, A. M. Norcia, and G. Wetzstein, "Towards a machine-learning approach for sickness prediction in 360° stereoscopic videos," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 4, pp. 1594–1603, 2018.

[17] T. M. Lee, J.-C. Yoon, and I.-K. Lee, "Motion sickness prediction in stereoscopic videos using 3d convolutional neural networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 5, pp. 1919–1927, 2019.

[18] J. Kim, H. Oh, W. Kim, S. Choi, W. Son, and S. Lee, "A deep motion sickness predictor induced by visual stimuli in virtual reality," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 554–566, 2022.

[19] H. Gao and E. Kasneci, "Eye-tracking-based prediction of user experience in vr locomotion using machine learning," *Computer Graphics Forum*, vol. 41, no. 7, pp. 589–599, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14703

[20] E. Wen, T. I. Kaluarachchi, S. Siriwardhana, V. Tang, M. Billinghurst, R. W. Lindeman, R. Yao, J. Lin, and S. Nanayakkara, "Vrhook: A data collection tool for vr motion sickness research," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3526113.3545656

[21] E. Wen, C. Gupta, P. Sasikumar, M. Billinghurst, J. Wilmott, E. Skow, A. Dey, and S. Nanayakkara, "Vr. net: A real-world dataset for virtual reality motion sickness research," *arXiv preprint arXiv:2306.03381*, 2023.

[22] U. Technologies, "XR Interaction Toolkit — XR Interaction Toolkit — 2.0.4 — 2.0," https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/index.html, 2023, [Accessed 04-04-2024].

[23] J. Paulo, "3D Textures," https://3dtextures.me/, 2024, [Accessed 04-04-2024].

[24] J. Voight, *Quaternion algebras*. Springer Nature, 2021.