Deep learning for human head motion prediction with confidence

Hugo Bell

Master of Science
Cognitive Science
School of Informatics
University of Edinburgh
2021

Abstract

When viewing and exploring a Virtual Reality (VR) scene, human head motion trajectories are fundamentally hard to predict. Trajectories are determined following an underlying stochastic component (the basic unpredictability of human behaviour), while over the course of a single video there are a wealth of possible trajectories that could be taken. For deep learning models aiming to predict these trajectories - to be used by VR rendering systems, thereby allowing high quality streaming - there is significant uncertainty over the accuracy of model predictions that has yet to be accounted for. In this thesis, we aim to address this shortcoming by extending an existing deep head motion prediction model to be capable of quantifying the uncertainty in its predictions. Specifically, we enforce a deep head motion prediction model to be explicitly probabilistic, by adding stochastic latent variables trained under an information bottleneck objective to its architecture, before using density estimation and repeated latent sampling to quantify the uncertainty associated with head motion trajectories. To evaluate uncertainty estimates, we develop an evaluation framework using real head motion trajectory datasets, as well as generating two datasets of synthetic trajectories for evaluation with constrained and known uncertainty in trajectories. We evaluate our model's ability to quantify the two main sources of uncertainty thought to be present in deep learning models, using Out-of-Distribution detection and correlation with predictive error. Experimental results show our model is effective at quantifying aleatoric uncertainty for real head motion trajectories, but shows success when quantifying epistemic uncertainty only when tested on highly simplified synthetic trajectories. Using visualisations of epistemic uncertainty estimates and model latent representations, we argue that failures of epistemic uncertainty estimates are due to a recently cited problem known as *feature collapse* [1] in our model's representations. Overall, our approach produces encouraging results that invite further development - particularly emphasising investigations into learned representations and development of effective representations.

Acknowledgements

- Thanks to my supervisor Prof. Bob Fisher for supporting this project, and for the many hours he spent discussing and bringing fresh insights to its contents - even using his own free time to help me when I struggled with experiments. Bob kept a cool head during what has been a complicated project - which I am very thankful for.

- Thanks to Lucile Sassatelli, who directed me towards the project goals and helped me understand many areas of the project I had no prior experience with. She allowed me great freedom and independence to explore with the project, but kept me heading in the right direction and when necessary kept me grounded. Furthermore, thanks to the Université Cote d'Azur for allowing me to conduct research for them while documenting it as part of my dissertation.

- Many thanks also to Quentin Guimard, who has spent much time struggling with the ideas used in this project like I have, and was always generous with his help whenever problems arose.

- Other thanks goes to Miguel Romero Rondon, Ian Fischer and Janis Postels for advice on extending their respective research.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Hugo Bell*)

Table of Contents

Chapter 1

Introduction

Virtual Reality (VR) is a form of immersive environment wherein a video, or simulated scene, contains content that surrounds a viewer and can be interacted with to determine future content. VR videos and scenes primarily occur with a full sphere (360° latitude, 180° longitude) of content around a viewer available [2]. Scenes are most commonly observed by a viewer via a Head Mounted Display (HMD) - display headsets designed such that viewers can actively turn and move their heads to interact with and observe different parts of the VR environment. In recent years, viewers have predominantly accessed immersive content via internet streaming [3] - which limits quality of experience due to the great data rate required to repeatedly render 360° of content [4] (effects such as streaming delays caused by the lag between head movement and rendering are common [5]).

To reduce the required data rate, solutions suggest rendering in high resolution only the portion of the sphere of content viewable from the viewer's current head position - known as their Field of View (FoV). To ensure high resolution and smooth transitioning between FoVs over multiple timesteps, existing methods [6, 7] require knowledge of viewers' likely future head positions to render 'tiles' that make up future FoVs in advance. This requires predicting the viewer's head trajectory over future video frames (ideally over frames > 5 seconds into the future), for use in a playback buffer that accounts for network instability [8]. Deep learning methods for generating these predictions treat likely future head positions (and therefore future FoVs to render) as a non-linear function of video content over prediction timesteps and previous head positions (i.e. the prior head trajectory). However, these methods [9, 10, 11, 5, 12, 13, 14, 15] have been shown to perform poorly, giving predictions that are worse than always predicting no head motion [4].

Poor performance of existing head motion prediction models is hypothesised to be due to the fact that they do not consider that, for the same input (past head motion trajectory and current video content), the viewer's head may take one of a diverse range of trajectories, different from those observed at training time. Training data for head motion prediction models will therefore necessarily be incomplete and noisy, meaning there will be many superior mapping functions for a given model left unlearned, and many out of distribution samples that cannot be accounted for at test time. In short, there will be uncertainty over the accuracy of the learned model's predictions when deployed on samples outside of training data. Assuming confidence in such predictions at best means overfitting to training data [16, 4], with even this case relying on the trained network function accurately modelling the problem at hand as posed by the training data.

This work aims to address these shortcomings, resulting in a deep learning model for head motion prediction that robustly accounts for uncertainty in viewer behaviours. This can be achieved by extending existing models to generate, in parallel with predictions, estimates for their predictive uncertainty (uncertainty estimates). Uncertainty estimates in effect provide a confidence interval over predictions at each timestep, that will allow streaming systems to render a 'halo' of tiles in high fidelity (proportional in size to estimated uncertainty) around predicted FoVs when models are applied for use in real-world immersive experiences. In this way, the impact of incorrect model predictions on experiences will be reduced [9]. The challenge of this task is in generating uncertainty estimates that are *well calibrated* [17] with the true uncertainty associated with a given video and past head trajectory - where uncertainty refers to both the inherent uncertainty of the data at hand and how it relates to the training data distribution.

In practise, this work extended a deep head motion prediction baseline model (introduced in Chapter 2), the *position-only* baseline [4], that shows comparable (if not superior) performance to all existing head motion prediction models besides the current State of the Art (TRACK [4]), with extensions easily applicable to all existing models (existing models and the position only baseline are described in sections 2.1.2 and 2.1.3). We extended the baseline (see Chapter 3) to be capable of generating uncertainty estimates (capable of *Uncertainty Quantification*, or UQ) using an approach based on density estimation over model hidden layer activations. We developed an evaluation framework (see Sections 4.3 - 4.4) for uncertainty estimates, and compared the results of different approaches taken for UQ using this framework (with experi-

mental results presented in Chapter 4). Furthermore, finding that existing head motion datasets largely did not differ substantially enough in their data distributions for testing of uncertainty estimates, we generated two datasets of synthetic head position trajectories (see Sections 4.1.1 and 4.1.2) that offer different levels of difficulty in the uncertainty estimation task. This allowed explicit control over what was included in the training data, and evaluation of uncertainty estimates using samples generated to be outside the training distribution.

Specifically, we approached developing an uncertainty aware position-only baseline by adding stochastic latent variables and an information bottleneck training objective [18, 19] to the model (see Sections 3.1 and 3.2.1), alongside a density estimating marginal over latent variables. Our evaluation used Out-of-Distribution (OoD) detection with varying distributional shifts, and correlation of uncertainty estimates with test-set error (explained in Section 3.3) to investigate our models' abilities to quantify different forms of uncertainty. Our findings show that density estimation over hidden layer activations is a valid method for uncertainty estimation in deep head motion prediction models, giving good results on the correlation task (Section 4.4) and on the OoD task (Section 4.3) when uncertainty in trajectories is constrained to be simple and known. However, we find that uncertainty in real OoD head motion trajectories is not clearly identified - a problem we reveal may be due to *feature collapse* [1] (see Chapter 5) in model representations after visualising representations and uncertainty estimates.

Chapter 2

Background and Proposed Approach

2.1 Head Motion Prediction Models

2.1.1 Problem Formulation, Assumptions & Saliency

Formally[1], (adopting notation used in [4]) for our interests the task of head motion prediction is: given some VR video or scene viewed at current timestamp t, predict a sequence of future coordinates $\{\mathbf{P}_\tau = [\theta_\tau, \psi_\tau]\}_{\tau=t+1}^{t+H}$ that indicate the predicted center of the viewer's FoV (correlated with their head position) for each video frame occurring within a window H seconds following t. Note that this means we make multiple predictions per second - for example if the video sampling rate is 25 frames per second, prediction timesteps $\tau \in [t : t+H]$ will fall evenly over 0.04 second intervals in this window. We call the prediction window H the *prediction horizon*, with prediction times $\tau \in [t : t+H]$ referred to as time*steps*, and times in the video itself (i.e. times from which H timesteps of predictions will be made) referred to as time*stamps*. As described in chapter 1, existing deep learning models predict future timestep coordinates by learning a non-linear mapping function from previous head coordinates $\mathbf{P}_{t-M}^{t}$ (where M is a historical context window such that $\mathbf{P}_{t-M}^{t}$ represents a viewer's prior head trajectory) as well as previous and future VR video content $\mathbf{V}_{t-M}^{t+H}$, to future head position coordinates. This is based on 2 key assumptions [4]:

(A0): That future head positions are correlated with the history of previous head positions.

(A1): That future head positions are correlated with visual content in VR videos or scenes.

[1] Note that some of the material in this chapter is adapted from my earlier informatics project proposal

Following investigations in [4] and [20], we can conclude models based on A0 and A1 are in combination sufficient to accurately predict head positions in a period up to $H = 5$s in length following some timestamp t for a variety of 360° videos. A0 corresponds to the idea of motion continuity (*inertia* - viewer's heads will move in line with previous motion), while A1 indicates viewers will move their heads (and therefore FoVs) aligned with their gaze coordinates, fixating on regions of interest in VR scenes (i.e. salient regions that capture their attention). For predicting future head positions, both [4] and [20] indicate that inertia is a good indicator for head positions during the first 2-3 seconds after prediction time, while visual content indicates head positions from that point.

Finding salient regions in images has been the subject of much historical computer vision research. Convolutional Neural Networks (CNNs) that output 'saliency maps' (i.e. a set of labels for input image pixels indicating their saliency) are the current standard approach for automatic saliency detection [21, 22, 23]. Deep head motion prediction models tend to model saliency by pre-processing video content to be in the form of a series of said saliency maps using these methods.

2.1.2 Previous Approaches & Architectures

The objectives of existing neural network architectures for head motion prediction often vary slightly from the problem formulation in this project. These include models for predicting gaze coordinates (from which FoVs can be derived) [11] and for predicting 'tiles' (relevant to current methods for VR streaming - see chapter 1) that will be active in future FoVs [5, 13, 14, 15]. Prediction horizons for these models generally vary from 30ms - 2.5s.

The first neural network model for head motion prediction, introduced in [9], considered only head position coordinates occurring in a fixed window before prediction timesteps as input to a static 3-layer feed-forward architecture. Fan et al. in [15] instead explicitly modelled head positions as a time series, also combining coordinates with video saliency information to predict whether tiles would be in future FoVs. A CNN derived from VGG-16 [24] extracted saliency maps from VR videos, which were concatenated with head position coordinates and fed into a single LSTM (Long Short Term Memory) [25] network (a commonly used form of RNN - see [26, Section 6.2-6.3] for more on RNNs and LSTMs). The architecture used is shown in Appendix A. Xu et al. in [11] used a similar architecture for gaze prediction, but instead used head

positions alone as input to the LSTM. LSTM outputs were then concatenated with saliency maps from current video content, and fed as input to a feed-forward neural network. Varying on these approaches, Nguyen et al. in [5] passed saliency maps for each video frame to an LSTM, with head position information encoded as a mask on maps. Similar to the formulation described in Section 2.1.1, saliency maps and head positions from the previous M video frames were used to make predictions for the next H timesteps via the LSTM in these autoregressive models [5, 11, 15].

Alternative approaches to modelling our task have been taken - for example using deep reinforcement learning [10]. However, a unified comparison of models was not possible - due to their differing problem formulations, evaluation metrics and dataset formats - until Rondon et al. introduced their evaluation framework in [2] (covered in more detail in sections 2.1.3 and 4.1). In their subsequent work, Rondon et al. then showed that all prior deep learning models had worse prediction accuracy than a no-motion baseline (predicting viewers' heads would never move in response to VR scenes) [4]. Following investigations regarding why such poor performance had occurred, Rondon et al. leveraged flaws found to design TRACK - a head motion prediction model that outperformed all existing models and the no-motion baseline. Specifically, they found that a failure to model video saliency as a time series (i.e. by processing video content with an individual LSTM) led to degradation in performance. As a result, the TRACK architecture uses individual doubly-stacked LSTMs to process both head positions and VR video content at every timestep. Outputs of these RNNs are concatenated and fed to another doubly-stacked LSTM. Predictions are generated by passing the final LSTM outputs to a feed-forward network layer, whose outputs are added with the head position from the previous timestep (in the manner of a residual network [27] - see Appendix A for an architecture diagram).

2.1.3 Evaluation

Evaluation methods for the models described in Section 2.1.2 vary widely - with direct comparisons between models and metric results facilitated by work conducted by Rondon et al. in [2, 4]. Fundamentally, metrics generally function by comparing predicted (derived either from predicted coordinates or from predicted active tiles) and ground truth FoVs. In common with evaluation in wider deep learning models, Fan et al. in [15] use Accuracy (the percentage of correctly predicted active tiles against the total number of predicted and groundtruth tiles), F1-score (the mean of precision and recall,

where precision is the ratio of correctly predicted active tiles against total number of predicted tiles, and recall is the ratio of ground truth active tiles correctly predicted by the model - as defined in [2]), as well as *Ranking Loss* ("the number of tile pairs that are incorrectly ordered by probability normalized to the number of tiles" [15]) as metrics. Nguyen et al. in [5] similarly use accuracy alone as a metric.

Rondon et al. in [2] compared these models and metrics, firstly by standardising model datasets such that they held shared formats and sampling rates. Crucially, they then introduced two baseline models to compare all prior models against: (1) the *No-motion* and (2) *Position-only* baselines. These were models that (1) assumed the viewer's head stayed still for the duration of all prediction horizons, and (2) predicted future head position coordinates using past position coordinates alone as input to an LSTM respectively. Subsequently, they showed these baselines outperformed all prior models, when evaluated on all described metrics and trained on the relevant datasets [4]. Similarly, when evaluating their model (TRACK), Rondon et al. in [4] compared its performance to prior models on the described metrics in the same manner as used with baselines - allowing direct comparison of models and showing TRACK outperforms all rivals. Motivated by the fact that there is a conversion factor (that may encounter noise in reality) between head coordinate predictions and predicted FoVs, they additionally evaluated the quality of predicted head position coordinates directly using *Average Orthodromic Distance* as a metric. This is a measure of the shortest distance between 2 positions on the surface of a unit sphere [28].

2.2 Uncertainty Quantification

Total uncertainty in deep learning model predictions (predictive uncertainty) is derived from two sources: *epistemic* and *aleatoric* uncertainty. Aleatoric uncertainty (sometimes referred to as *data* uncertainty) refers to inherent, irreducible randomness in the data. In our case, aleatoric uncertainty describes how, given some current video content and head position history input, future head positions are determined with an irreducible stochastic component (related to the unpredictability of human behaviour). Epistemic uncertainty (sometimes known as *knowledge* uncertainty) refers to uncertainty in predictions due to a lack of knowledge about the best possible model (i.e. the best possible model parameter settings), and can be reduced with additional information - such as additional training data or a more suitable model architecture [29, 30].

Modelling uncertainty in deep neural networks requires formulating networks as

probabilistic models, such that outputs (predictions) y are samples from a distribution $p(y|\mathbf{x}, \boldsymbol{\omega})$ - where $\mathbf{x}$ and $\boldsymbol{\omega}$ are model inputs and parameters respectively. Often this formulation is made explicit, by introducing random variables or activations into the network (forming a *stochastic neural network*) [31, 32]. Optimisation of implicit or explicit probabilistic models aims to maximise similarity between $p(y|\mathbf{x}, \boldsymbol{\omega})$ and the true distribution mapping inputs to outputs on some task.

2.2.1 Density Estimation for Uncertainty Quantification

Historically, Bayesian Neural Networks (BNNs) have been the main form of stochastic model used to estimate uncertainty. BNNs draw network weights from a prior distribution and generate outputs from the sampled model, with weights repeatedly sampled during training and inference (for use in uncertainty estimation). They aim to learn the true *Bayesian posterior* distribution $p(\boldsymbol{\omega}|\mathbf{x})$ for the task at hand, such that at test time the variance in the learned posterior, for a given input, represents epistemic uncertainty [33]. However, in practise BNNs are difficult to train, perform poorly on tasks with high dimensional inputs [34, 35, 36], and often fail to model the full problem distribution in multi-modal problems [37]. A valid alternative method consists of building an ensemble of deterministic network models that output both mean predictions and predictive variances (with variance representing uncertainty), with uncertainty estimates generated by averaging over ensemble outputs [35]. However, for real-world applications this is not practical as it includes specialised adversarial training processes and training of multiple models in parallel [38].

Variational Autoencoders (VAEs) [39, 40] are an alternative family of stochastic networks that hold similar probabilistic properties, without suffering from these problems. In an encoder-decoder architecture, they introduce a set of latent random variables $\mathbf{z}$ drawn from a latent conditional distribution $p(\mathbf{z}|f(\mathbf{x}))$ between encoder and decoder (where $f(\mathbf{x})$ represents the model encoder), trained to model (point-wise) a transformed probability density function of the input data distribution, in a manner useful for modelling the true output density function. In their original formulation, VAEs aimed to reconstruct their inputs (i.e. ground truth outputs were perfect reconstructions of inputs) - such that, given a test input, the likelihood of that input could be estimated. VAEs therefore can be usefully classed as density estimating models - they approximate the density of an unknown training distribution given limited samples from that distribution. VAEs are significant in this respect because they can also

be trained using conventional optimisation methods, treating density estimation as a *Variational Inference* problem [30].

For generating uncertainty estimates, the idea of density estimation from VAEs has seen much work in recent years. Given a density estimating model of the task input data distribution, the epistemic uncertainty associated with a predicted output can be estimated by observing the likelihood of the test input under the density model [41]. However, VAEs rely on using simple gaussian priors over latent variables - which is limiting for complex data distributions - and approximations in the variational inference process limit the quality of learned latent distributions $p(\mathbf{z}|\mathbf{x})$ if they are high dimensional [42]. As a result, work on density estimation for UQ (Uncertainty Quantification, or for the similar task of OoD detection) has increasingly focused on developing density estimation methods for deterministic networks [43, 44, 45] or variational models that maintain quality of high dimensional latent variables [46, 18, 19, 36, 46] (note that we consider only techniques compatible with regression problems, as is the case with head motion prediction).

The concept of the *Information Bottleneck* (IB) [47] has been used with variational neural network models to reduce the dimensionality of latent distributions while maintaining effectiveness. Specifically, the IB method achieves this by constraining latent representations to have minimal mutual information with network inputs, while having maximal mutual information with network outputs [48]. Alemi et al. in [18] and [19] have been the primary source of developments in this respect. They developed a variational encoder-decoder model that used the IB principle in [18] (calling it the *Variational Information Bottleneck*, or VIB) to ensure the quality of latent variables in approximate distributions inserted between encoder and decoder, while ensuring the model could be trained via standard optimisation methods as in VAEs. They then utilised these effective latent variables for UQ via density estimation in [19], introducing a learned marginal distribution $m_\phi(\mathbf{z})$ modelling the density function of latent encodings over the training distribution (allowing computation of test input likelihoods under the marginal).

An important development upon Alemi et al.'s work saw Sinha et al. in [36] demonstrating that aleatoric uncertainty estimates can be generated from the VIB model by repeatedly sampling from the latent encoding distribution for a test input, generating the corresponding network outputs and observing their empirical variance. Furthermore, Postels et al. in [43] theoretically and empirically proved that the VIB density estimating marginal could be used to express epistemic uncertainty. Postels et al. also

developed a method of density estimation for UQ that was applicable to arbitrary deterministic neural networks, extracting training set hidden layer activations and fitting both a Conditional Normalizing Flow [49] and Gaussian Mixture Model (GMM) to these latent encodings. Density estimation, with uncertainty estimates generated as described previously using likelihoods, was achieved using the outputted GMM and normalizing flow as density models. Although alternative methods for density estimation using GMMs (often in the form of Mixture Density Networks) have been developed in the UQ literature, practical issues have been found with these models such as non-convergence for high-dimensional problems [50] and mode collapse [51].

2.3 Proposed Approach

As described in sections 2.1.1 and 2.1.2, the current standard in deep learning models for head motion prediction accept a sequence of video frames alongside a history of head positions as input. This comprises a high-dimensional input sequence, which would normally require relatively high dimensional representations in a deep learning model for processing - for example, the position-only baseline model [2] utilises LSTMs with 512 dimensional hidden layers. In Section 2.2.1, we have seen such high dimensions will cause problems with UQ-capable architectures using conventional variational inference. Furthermore, given models will be deployed in real-time for VR streaming, it is important that uncertainty estimates can be generated efficiently by our head motion prediction model.

For efficiency, and to allow handling of a high dimensional input sequence, we therefore consider Alemi et al.'s VIB model for generating uncertainty estimates, as implemented in their 2018 paper [19] using density estimation. This can generate epistemic uncertainty estimates with a single forward pass and handles high input dimensionality via training with an IB objective. Aleatoric uncertainty estimates can be generated with a minimal number of forward passes through the VIB decoder using Sinha et al.'s approach.

To validate the results generated from implementing Alemi et al.'s VIB model, we also implemented the main alternative method to the VIB approach, as introduced by Postels et al. in [43], wherein they fit a Conditional Normalizing Flow to training set activations. This allows exact evaluation of the log likelihood of test inputs [42, 52]. However, Postels et al.'s approach was found to be unsuitable in its current form for our purposes, as their method assumes a uniform distribution over the output space

(which is not the case for the head motion prediction task).

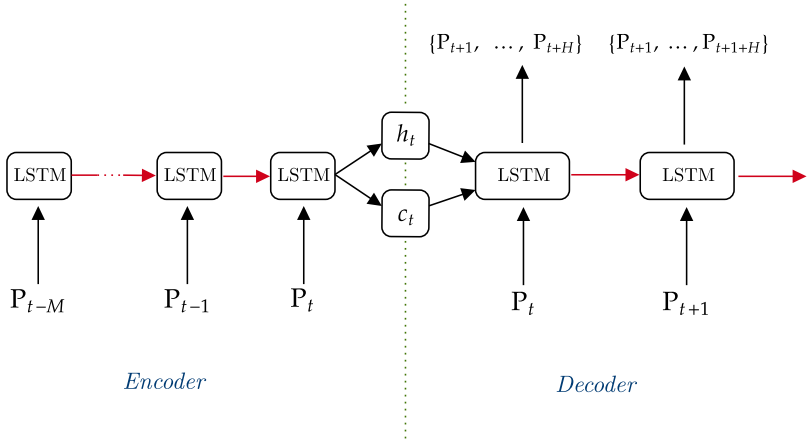

Figure 2.1: The position-only baseline head motion prediction model from Rondon et al. in [4]. $\mathbf{P}_t$ indicates head position coordinates $[\theta_t, \psi_t]$ at timestep t.

Ideally, we would implement the VIB approach in order to extend the State of the Art in deep head motion prediction models, TRACK [4]. However, as described in Section 2.1.2, TRACK utilises 3 separate deep encoder-decoder LSTM networks in its architecture. Since the VIB UQ approach relies on inserting latent variables between an encoder and decoder, in a deep architecture with parallel computation (i.e. in the processing of video and head position history using parallel encoder-decoder LSTMs) such as TRACK, there is significant uncertainty over where best to place latent variables.

As a result, to reduce the number of unknowns necessary to explore in this work, we extend the *Position-only baseline* head motion prediction model. This model consists of only a single encoder-decoder LSTM architecture (shown in figure 2.1), such that there is a single eligible position to insert latent variables.

Using the baseline model allows us to focus on which elements of the VIB approach may be problematic (or most important to tune) in general, and when used in conjunction with the head motion prediction task (particularly recalling that the majority of head motion prediction models from recent literature are at least partially autoregressive, which may be problematic). Theory and implementation details of these approaches will be described in Chapter 3.

Chapter 3

Methods

In this chapter, we introduce the theory and practicalities associated with our VIB approach (Section 3.1) as motivated and defined in Section 2.3. We then describe our methodology for adapting the VIB approach to the position-only baseline in Section 3.2.1.

3.1 Variational Information Bottleneck

The VIB seeks to replicate the capability of VAEs to generatively model a data distribution using latent variables, in a supervised learning formulation. Specifically, VIB maximises the log likelihood of outputs $\log p(\mathbf{y}|\mathbf{x})$, to learn a set of stochastic latent variables $\mathbf{z}$ that model the factors of inputs $\mathbf{x}$ that are generative for outputs $\mathbf{y}$. Note that stochastic $\mathbf{z}$ also mean outputs are treated as random variables, with an encoder for latent variable inference and a decoder for output value generation. The IB objective is used in conjunction with this architecture, to reduce the dimensionality of latent encodings (and motivated by the success of information theoretic constraints in the β-VAE architecture [53]) - shown in equation 3.1.

$$\max\ I(\mathbf{z};\mathbf{y}) \text{ subject to } I(\mathbf{z};\mathbf{x}) \leq I_c \tag{3.1}$$

The IB objective states that the mutual information between inputs and latent encodings, shown as $I(\mathbf{z};\mathbf{x})$, should not exceed some constant I_c - while latent encodings should still be maximally informative, and thereby effective, for computation of outputs. Effectiveness is enforced in the objective by maximising mutual information between encodings and outputs $I(\mathbf{z};\mathbf{y})$. These constraints allow encodings to be significantly lower dimensional than would otherwise be the case, by penalising the pres-

ence of excess information from inputs. The IB objective can be written as an objective function using a langrange multiplier β:

$$\max\ I(\mathbf{z};\mathbf{y})\ -\ \beta I(\mathbf{z};\mathbf{x}) \tag{3.2}$$

where $\beta \geq 0$ controls the amount information from inputs is penalised (i.e. the "size of the information bottleneck" [19]).

In practise, calculating $I(\mathbf{z};\mathbf{y})$ and $I(\mathbf{z};\mathbf{x})$ analytically requires computation of difficult and intractable integrals. As a result, the original VIB architecture is made up of a stochastic encoder $e_\theta(\mathbf{z}|\mathbf{x})$ from which latent variables (i.e. encodings) can be sampled, a stochastic decoder $q_\psi(\mathbf{y}|\mathbf{z})$ from which outputs can be generated, and a variational marginal distribution over latent variables $m_\phi(\mathbf{z})$ [18]. These factors can be combined for optimisation under a single objective:

$$\max_{\theta,\phi,\psi} \mathbb{E}_{p(\mathbf{x},\mathbf{y})e_\theta(\mathbf{z}|\mathbf{x})}\left[\log q_\psi(\mathbf{y}|\mathbf{z}) - \beta \log \frac{e_\theta(\mathbf{z}|\mathbf{x})}{m_\phi(\mathbf{z})}\right] \tag{3.3}$$

To allow for uncertainty estimation, note that the variational marginal $m_\phi(\mathbf{z})$ is parameterized by ϕ. This is an innovation from Alemi et al. [19] that means the marginal can be trained (in parallel with the encoder-decoder model parameters during training) to model the distribution of latent encodings over the training data (meaning $m_\phi(\mathbf{z})$ models the density of the training data distribution in the lower dimensional encoding space). $m_\phi(\mathbf{z})$ can be used for UQ by computing the log-likelihood of samples from the encoder distribution under the marginal (evaluating their density within the training distribution). For a deeper understanding of the theory behind the VIB model, and how it quantifies uncertainty, see Appendix B.1.

To measure aleatoric uncertainty in the VIB, Sinha et al.[36] (albeit in a different model formulation) use the intuition that if we sample multiple M latent encodings from $e_\theta(\mathbf{z}|\mathbf{x})$ and generate corresponding predictions from the decoder, the set of outputs $\{y_i\}_{i=1}^M$ approximates those from an ensemble [18, 35]. As a result, the variance of these predictions represents aleatoric uncertainty.

3.2 Position-Only Baseline

As described in Section 2.1.3, the position-only baseline is an LSTM encoder-decoder architecture, that for each timestamp t takes a sequence of head position coordinates (of the form $\mathbf{P}_\tau = [\theta_\tau, \psi_\tau]$) from the $t-M$ prior timestamps as input, and makes predictions for $t+H$ future timesteps (using ground truth or prediction head position coordinates

as inputs to the decoder during training or inference respectively, under a teacher forcing training procedure [54]). The decoder LSTM is initialised using the final hidden state and cell-memory state vectors $\mathbf{h}_t$ and $\mathbf{c}_t$ from the encoder, where t is the final encoder timestep (see [25] for detail on the meaning of $\mathbf{h}$ and $\mathbf{c}$ vectors). We extend the implementation of the position-only baseline as used by Rondon et al. in [2, 4][1].

3.2.1 Position-Only-VIB

We adapt the position-only baseline to be capable of UQ using the VIB approach (naming the implemented model the "Position-Only-VIB" model) by introducing latent variables between the encoder and decoder LSTMs. Given there are two vectors passed between the LSTMs, we model both $\mathbf{h}_t$ and $\mathbf{c}_t$ as stochastic latent variables. We do this by doubling the size of encoder LSTM activations from 512 to 1024, before adding individual trainable affine layers to process the $\mathbf{h}_t$ and $\mathbf{c}_t$ vectors each (also 1024 dimensional). We then initialise two 512 dimensional diagonal Gaussian distributions using the outputs of affine layers, with the first 512 affine activations being used for distribution means, and last 512 forming standard deviations. We can then sample the initial decoder vectors from their respective distributions. This architecture is shown in figure 3.1.

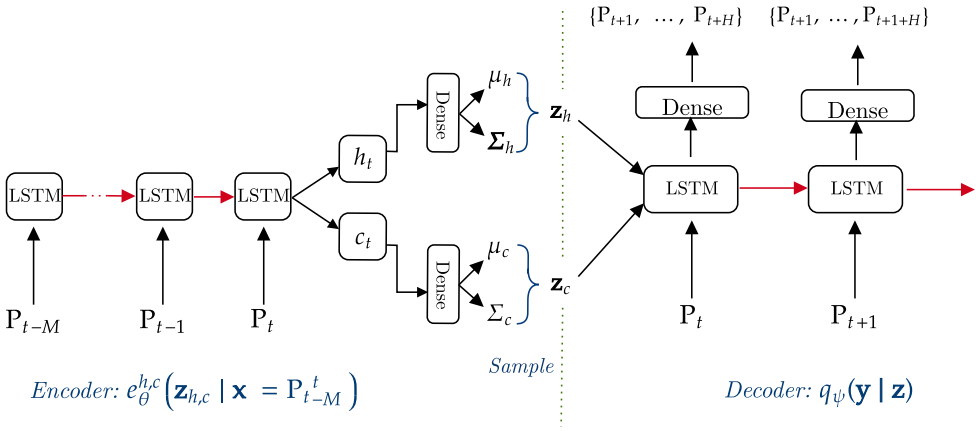

Figure 3.1: The position-only model with stochastic latent encoding distributions for the LSTM hidden and cell-memory states respectively $p(\mathbf{z}_{h,c}|\mathbf{x})$, which are sampled from to initialise the decoder LSTM.

Alemi et al.'s implementation of the VIB model in their 2018 paper uses a 3-

[1]Code implementing the position-only baseline was taken from `https://gitlab.com/miguelfromeror/head-motion-prediction/-/blob/master/position_only_baseline.py`.

dimensional, fully covariant Gaussian distribution over their encoder and a mixture of 200 3-dimensional fully covariant Gaussians for the marginal. Our model has two 512 dimensional Gaussians that generate initial decoder LSTM states (smaller dimensional distributions were tested and resulted in significantly worse error performance). Since these two distributions could be radically different (LSTM **h** and **c** states can encode very different information [25]), and the VIB loss function requires encoder samples to be compatible with the marginal (for density estimation under the marginal), we use two distributions formed from a mixture of 512-dimensional Gaussian components - giving two marginals m_ϕ^h and m_ϕ^c that respectively model each of the encoder distributions. Note that this means our model generates two parallel epistemic uncertainty estimates, the **h**-state and **c**-state rates. The number of components in marginal mixtures was set initially to 200 (mirroring the method of Alemi et al.), and was varied over testing to observe its effect. To generate aleatoric uncertainty estimates we follow Sinha et al.'s approach closely. We sample M sets of $\mathbf{z_h}$ and $\mathbf{z_c}$ tensors from encoder distributions for a single input example and generate the corresponding H predictions from the decoder, giving us a set of predictions $\{\mathbf{y}_i = [\theta_i, \phi_i]\}_{i=1}^M$ for each timestep. We then take the variances across θ and ϕ predictions individually, before generating a single variance prediction for each timestep (the aleatoric uncertainty estimate) by taking the mean of these two variances. The number of samples M in experiments was varied from 4 to 50, and will be covered in chapter 4.

Training the model is achieved through maximum likelihood estimation of the objective shown in equation 3.3 (albeit we use two β values for each of the encoder distributions: β_h and β_c), with the training algorithm pseudocode shown in algorithm 1 (with the *OrthDist* computation defined in Section 4.4). Note that computations $\boldsymbol{z}_{h,c} \sim \boldsymbol{\mu}_{h,c} + \boldsymbol{\Sigma}_{h,c}^2 \cdot \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$ document the *reparameterisation trick*. This was the method introduced by Kingma & Welling [39] for training parameters of distributions through samples, for use with VAEs.

The code implementation used here is written using Tensorflow and Keras, and is adapted from a GitHub repository by Alex Alemi[2] that demonstrates the use of the original (i.e. without uncertainty quantification [18]) VIB method for the MNIST dataset. Additionally, we adapted code for the marginal $m_\phi(\mathbf{z})$ from a sample supplied by Ian Fischer [55], a co-author of the VIB papers. The original and adapted versions of VIB code are available in Appendix C.1.

[2] Alemi GitHub repository: `https://github.com/alexalemi/vib_demo`

Algorithm 1 Training procedure for the position-only-VIB model. Note that $q_\psi(\mathbf{z}_{h,c})$ represents a decoder LSTM initialised with $\mathbf{z}_h$ and $\mathbf{z}_c$, with affine layers over outputs. $e_\theta^{h,c}$ represent the affine layers used over the encoder LSTM to generate $\mathbf{h}$ and $\mathbf{c}$ distributions respectively. $\texttt{log\_prob}(a|b)$ represents calculating the log likelihood (evaluating the density) of sample a's occurrence under distribution b.

Input: Head position history $\mathbf{P}_{t-M}^{t}$, Target output $\mathbf{P}_{t+1}^{t+H}$

Output: VIB Loss $\mathbb{L}_{VIB}$

$Encoder : e_\theta^{h,c}(\mathbf{x})$

$Decoder : q_\psi(\mathbf{z_h}, \mathbf{z_c})$

$Marginals : m_\phi^{h,c}(\mathbf{z}_{h,c}) = \sum_{i=0}^{N} \alpha_i \cdot \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \quad \sum_{k=0}^{N} \alpha_k = 1$

$\mathbf{x} \leftarrow \mathbf{P}_{t-M}^{t}$

$\boldsymbol{\mu_h}, \boldsymbol{\Sigma_h} \leftarrow e_\theta^h(\texttt{LSTM}(\mathbf{x}))$

$\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c} \leftarrow e_\theta^c(\texttt{LSTM}(\mathbf{x}))$

$\boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$

$\mathbf{z_h} \sim \boldsymbol{\mu_h} + \boldsymbol{\Sigma}_h^2 \cdot \boldsymbol{\varepsilon}$, $\mathbf{z_c} \sim \boldsymbol{\mu_c} + \boldsymbol{\Sigma}_c^2 \cdot \boldsymbol{\varepsilon}$

Prediction $\hat{\mathbf{P}}_{t+1}^{t+H} \leftarrow q_\psi(\mathbf{z_h}, \mathbf{z_c})$

$$\mathbb{L}_{VIB} \leftarrow OrthDist(\mathbf{P}_{t+1}^{t+H}, \hat{\mathbf{P}}_{t+1}^{t+H}) + \beta_h\big(\texttt{log\_prob}(\mathbf{z_h}|\mathcal{N}(\boldsymbol{\mu_h}, \boldsymbol{\Sigma_h})) - \texttt{log\_prob}(\mathbf{z_h}|m_\phi^h)\big) + \beta_c\big(\texttt{log\_prob}(\mathbf{z_c}|\mathcal{N}(\boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})) - \texttt{log\_prob}(\mathbf{z_c}|m_\phi^c)\big)$$

Calculate gradients with respect to $\mathbb{L}_{VIB}$ and adjust parameters to minimise

3.3 Evaluation

In this section we introduce the means through which we evaluate model performance. As defined in Section 2.2, there are generally considered to be two sources of uncertainty in deep learning model predictions: epistemic and aleatoric uncertainty. In sections 3.1 and 3.1 we have described how our position-only-VIB model will estimate each of these uncertainties. To evaluate this model, we must evaluate epistemic and aleatoric uncertainty estimates by their *calibration* - how well model-generated epistemic and aleatoric uncertainty estimates align with the true epistemic and aleatoric uncertainty associated with test inputs respectively. Sections 3.3.1 and 3.3.2 introduce the ideas behind evaluation for epistemic and aleatoric uncertainty estimates respectively.

3.3.1 Epistemic uncertainty calibration

Epistemic uncertainty describes model uncertainties that depend on reducible factors such as choice of training data (from which model parameters are derived) or model architecture. Since we cannot feasibly test the uncertainty associated with our chosen model architecture, the most effective way to test and evaluate epistemic uncertainty estimates is to consider how well they express (*calibrate* with) the similarity of a test example to those within the training data distribution (*in-distribution*, or iD, samples). Intuitively, this is because iD test samples will exhibit behaviours seen previously, and therefore predictions should be accurate with high confidence (i.e. uncertainty estimates should be small). By contrast, test samples that are increasingly distant from the training distribution (increasingly OoD) may exhibit behaviours increasingly less similar to those seen during training - meaning uncertainty associated with predictions for those samples should be higher [1].

A "good" uncertainty-aware model will generate epistemic uncertainty estimates that calibrate well with this intuition. Testing for calibration is conducted by repeatedly using test sets compiled from a mix of iD and OoD input samples and generating uncertainty estimates in response to samples. We then threshold estimates to see how well input iD and OoD samples can be classified correctly based on the magnitude of uncertainty estimated by the model. Repeated test sets use OoD samples drawn from distributions increasingly distant from the training distribution with repetitions. Calibration is then assessed by observing if classification performance increases proportionally with the size of OoD samples' distributional shift (shifts away from the training distribution should give similar improvements in classification performance, as uncertainty estimates for OoD samples should increase in magnitude while iD estimates remain constant, meaning the classification task is increasingly simple).

In practise, uncertainty estimates generated under the VIB approach are not calibrated themselves (in that per-instance rate values are meaningful only in how they vary in response to iD or OoD samples). As a result, to set up the binary classification task as described it is common practise to use the Area Under the Receiving Operating Characteristic (AUROC) as a threshold independent metric [19, 43, 1]. The AUROC score treats a continuous input signal as a predictor for a set of ground truth binary labels, and integrates over possible thresholds in the signal to test how well it discriminates between binary labelled cases. For each threshold True Positive (TPR) and False Positive (FPR) Rates are generated, with the AUROC score expressed by the

area under the curve for TPR against FPR. AUROC scores range from 0-1, with 0.5 indicating the input signal is a random classifier, 1 indicating a perfect classifier (100% classification accuracy), and 0 indicating a perfectly incorrect classifier (in our case, the signal labels all iD cases as OoD, and vice versa).

The process of testing uncertainty estimate AUROC scores on OoD samples that are close to being iD, while incrementally shifting samples away from the training distribution, is known as evaluation via *distributional shift* [43]. Testing uncertainty estimates on their ability to discriminate between iD input samples and OoD samples far from the training distribution is known as *Out-of-Distribution detection*. We will describe our approach to these evaluation methods in Section 4.3.

3.3.2 Aleatoric uncertainty calibration

Aleatoric uncertainty describes model uncertainty that is caused by inherent uncertainty or noise in the data, and is therefore irreducible. Assuming a well-fitted model, this uncertainty causes fluctuations in the model's performance within the training data distribution. For this reason, a good method for evaluating aleatoric uncertainty estimates is by calculating the correlation between estimates and predictive error for iD test samples [43], over a set of sampled error and uncertainty prediction values. The intuition of this is that, over a test set of iD samples, any increase or decrease in predictive error should be matched by a similar increase or decrease in the generated aleatoric uncertainty estimate - in effect "explaining" the fluctuation in performance.

For evaluating VIB aleatoric uncertainty estimates we simply generate uncertainty estimates from our model and calculate their correlation with predictive error over a set of iD test samples. To measure predictive error we use average orthodromic distance between ground truth and predicted coordinates as a metric - calculated for a pair of predicted $\hat{\mathbf{y}} = [\hat{\theta}, \hat{\phi}]$ and target $\mathbf{y} = [\theta, \phi]$ coordinates described in terms of Euler angles following Rondon et al.'s implementation in [2, 4]:

$$\begin{aligned} OrthDist &= \frac{\sqrt{(\cos\hat{\phi}\cdot\sin\delta\theta)^2 + (\cos\phi\sin\hat{\phi} - \sin\phi\cos\hat{\phi}\cos\delta\theta)^2}}{\sin\phi\sin\hat{\phi} + \cos\phi\cos\hat{\phi}\cos\delta\theta} \\ \delta\theta &= \arctan\left(\frac{\sin(\theta-\hat{\theta})}{\cos(\theta-\hat{\theta})}\right) \end{aligned} \tag{3.4}$$

We calculate orthodromic distances between corresponding target and predicted positions for sets of H predictions made on each timestamp. Our experiments and results for evaluating aleatoric uncertainty are presented in Section 4.4.

Chapter 4

Experiments & Results

4.1 Data

The dataset used for conventional training and testing of all models was that compiled by David et al. in [56] (following notation in [2, 4] we will refer to this as the David-MMSys18 dataset)[1]. It contains 19 high definition dynamic 360° videos collected from YouTube, all at least 4K in resolution (3840 × 1920 pixels) sampled at 90 frames per second. Each video was viewed by 57 subjects (25 women, 32 men) wearing HTC VIVE headsets, allowing viewing of scenes with 110° horizontal by 110° vertical FoV [56]. Headsets have built-in head position tracking sensors, from which 3D (x, y, z) coordinates corresponding to the centre of viewer FoVs for each video frame were recorded (giving ground truth head position data for all subjects viewing videos). Gaze position data for each head position (such that eye orientations can be calculated relative to head positions) were calculated using an SMI (*SensoMotoric Instrument*) eye-tracker with a precision of 0.2°. Videos are all 20 seconds in length, are filmed using both fixed and moving camera views, and contain content including panoramas, drone flights and water park rides (see [56, Table 1] for more details). During viewing, subjects were instructed to freely explore scenes as naturally as possible. We utilise the framework introduced by Rondon et al. in [2] to process the dataset, standardising videos to include 0.2 second intervals between neighbouring frames (as done by Rondon et al. - see [2, Section 3.1] for details of the subsampling process). Pre-processing in this way was implemented so that position-only baseline's performance was directly comparable with previous work in [4, 2]. Example frames from videos in the dataset are shown in figure 4.1, displayed in equirectangular projections (the format they were

[1] Available for download from `ftp://ftp.ivc.polytech.univ-nantes.fr/`

used in our implemented models).

Figure 4.1: Example video frames from the David-MMSys18 dataset (clockwise, frames are from videos titled: 1_PortoRiverside, 3_PlanEnergyBioLab, 4_Ocean, 14_Warship, 16_Turtle, 18_Bar)

Figure 4.2 shows the Cumulative Distribution Function (CDF) for maximum angular distances from initial to final head positions in position window lengths (equivalent to the prediction horizons defined in Section 2.1.1) $H \in 0.2, 0.5, 1, 2, 5, 15$ seconds, for viewers in the David-MMSys18 dataset. To visualise these viewing behaviours, figure 4.3 displays a trace of head (and gaze) position coordinates in 3D space for a subject exploring a video in the David-MMSys18 dataset. For conventional training and testing of models, the David-MMSys18 dataset is randomly split into 14 videos for training, each viewed by 29 subjects, with 5 videos for testing, each viewed by 28 subjects.

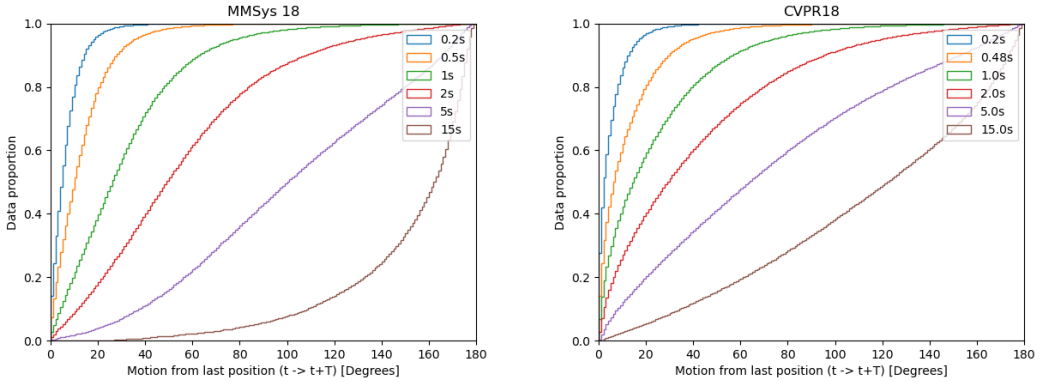

Figure 4.2: Cumulative Distribution Function expressing the motion distribution of viewers in the David-MMSys18 (left) and Xu-CVPR18 (right) datasets. Generated using code from [2].

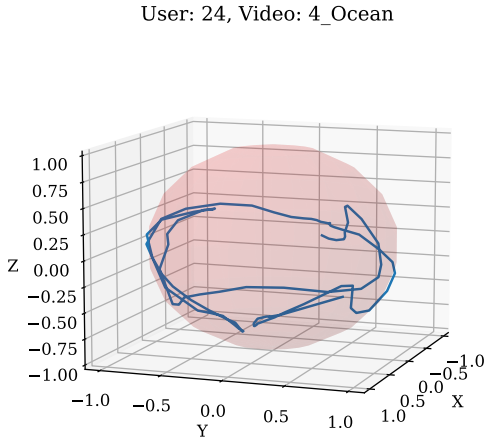

Figure 4.3: Diagram showing an example trace (represented in the unit sphere) that follows the (x, y, z) coordinates of viewer 24's head when watching video '4_Ocean', extracted from the David-MMSys18 dataset. Generated using code from [2].

For evaluation on OoD detection, we used an additional dataset to David-MMSys18 that acted as a source of 'true OoD' trajectories when testing epistemic uncertainty estimates. For this, we used the dataset compiled by Xu et al. in [11] (following notation in [2] and [4] we will refer to this as the Xu-CVPR18 dataset)[2]. It contains 208 high definition dynamic 360° videos collected from YouTube, all at least 4K in resolution sampled at 25 frames per second. Head position data was collected in the same manner as in David-MMSys18, and eye-tracking data was recorded similarly using a *7invensun a-Glass* eye tracker. Videos varied in length from 20 - 60 seconds, and contain a similar variety of contents to David-MMSys18, again with a mix of fixed and moving camera views. Each video was viewed by at least 31 subjects (from a pool of 45 potential viewers) who were instructed to explore scenes freely. We pre-processed data using Rondon et al.'s framework as for David-MMSys18. The CDF for maximum angular distances from initial to final head positions (as seen for David-MMSys18) for Xu-CVPR18 is shown in figure 4.2.

4.1.1 Hard Synthetic Data

To supplement experiments conducted using real head motion trajectories from the David-MMSys18 and Xu-CVPR18 datasets, we generated further datasets of *synthetic* head motion trajectories wherein the uncertainties associated with each trajectory were

[2]Instructions for downloading the dataset can be obtained from `https://gitlab.com/miguelfromeror/head-motion-prediction/tree/master`

manually constrained. Since the model we are implementing considers only positions as input, synthetic trajectories can be arbitrarily generated without consideration of video content. Particularly, generated trajectories do not need to be consistent in featuring the same positions, as the position-only-VIB model is coordinate independent. For the 'Hard Synthetic Trajectories Dataset' (HSTD), we constructed a single constrained synthetic training set, and multiple test sets wherein constraints on OoD trajectories were relaxed to varying extents, allowing distributional shift.

All trajectories generated have the same form as those used from David-MMSys18 and Xu-CVPR18 - each featuring 100 (x, y, z) head positions on the surface of a unit sphere, the equivalent of one head position for each frame in a 20 second video sampled at 5 frames per second. We construct each trajectory by generating a population of 2000 random candidate positions on the surface of a unit sphere, and sequentially selecting a limited number of positions (this varies over training and test sets, between 2 and 10 positions following the initial position) that will be members of the final trajectory. Candidate positions are sampled and added to the trajectory if they meet a series of 3 constraints. Once a full set of positions has been collected, a trajectory is generated by interpolating between selected positions using spherical linear interpolation [57]. The 3 constraints on positions are as follows (the motivations behind constraints are described in Section 4.3.1):

1. **Distance constraints:** We limit selected positions in the trajectory to be close to the equator of the unit sphere. Positions selected across training and test sets are constrained to be less than 22.5° ($\frac{\pi}{8}$ radians) elevated from the equator of the unit sphere (in terms of Euler angles, $-22.5^\circ < \phi < 22.5^\circ$).

2. **Angular constraints:** Any one position selected to be within the trajectory (excluding initial positions) must have an angle of rotation from the previous portion of the trajectory that is within set bounds (see the left of figure 4.4, where any position p_3 - for which c is a candidate - must have an angle θ with respect to the trajectory $p_1 \rightarrow p_2$ that is within set bounds).

3. **"Speed" constraints:** The perceived "speed" of a synthetic trajectory is constrained via the number of positions selected to occur in it (i.e. a trajectory with 2 selected positions, excluding the initial position, is "slower" than one with 10 separate selected positions).

Constraints 1 and 3 are simple to implement. In practise, we implement constraint 2 by calculating the rotational angle between the plane formed by the 2 prior selected

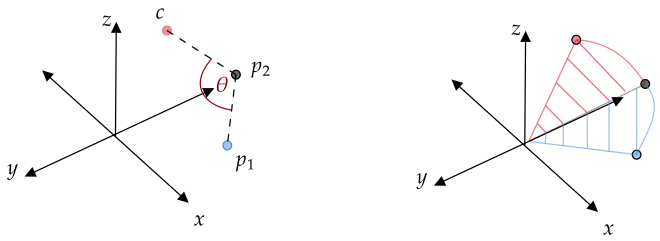

Figure 4.4: Diagram showing 2 synthetic head positions p_1, p_2 that are already members of a generated synthetic trajectory, with a 3rd candidate position c. (Left) θ is the angle between p_1, p_2 and c that we constrain in the trajectory. (Right) In practise we express θ as the rotational angle between the planes constructed from p_1, p_2 (shown in blue), and p_2, c (shown in red)

positions, and the plane formed by the last selected position and a candidate position (see right of figure 4.4). We calculate the line equations of the normal to each plane by taking the cross product of vectors describing the points it is formed from, and then calculate the angle between planes following equation 4.1.

$$\theta = \cos^{-1} \frac{|A_1 \cdot A_2 + B_1 \cdot B_2 + C_1 \cdot C_2|}{\sqrt{{A_1}^2 + {B_1}^2 + {C_1}^2} \cdot \sqrt{{A_2}^2 + {B_2}^2 + {C_2}^2}} \tag{4.1}$$

Where the plane normals are described by equations $Ax + By + Cz$. For the first selected position following the initial position, no previous trajectory segment is available. In this case we enforce the angle constraint by defining an axis from the initial position to some candidate position, calculating the rotation around the axis in terms of euler angles, and ensuring pitch and yaw angles fall within the angle constraints (discarding candidates that do not meet these specifications).

For the training set, we generated 1000 trajectories composed from a sequence of 4 positions (the speed constraint) with angles between planes of consecutive segments (the angular constraints) $50° < \theta < 60°$ - an example of these trajectories is displayed in figure 4.5. For testing, we generated 200 trajectories for each test set, preserving the distance constraint from training conditions throughout. We generated test sets varying on the speed constraint - with trajectories made up from one of 2,4,5 or 10 positions in each set - as well as separately varying on the angular constraint - with trajectories containing angles θ in a range from one of $0° - 30°, 30° - 50°, 50° - 60°, 50° - 80°$ or $70° - 180°$ (in total forming 8 test sets including an in distribution set, see Appendix D.1 for visualisations of example trajectories). In OoD experiments using HSTD, test sets consisted of one SSTD test set mixed with 200 generated samples following training set constraints.

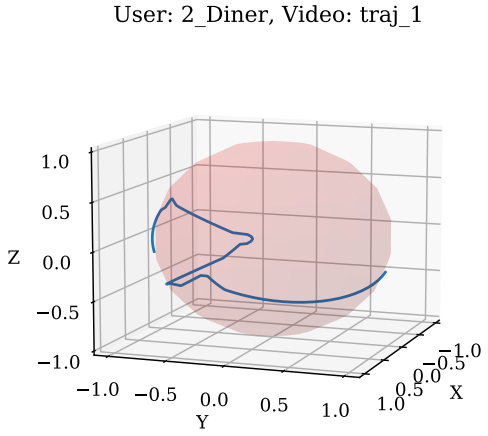

Figure 4.5: Diagram showing an example trace (represented in the unit sphere) that follows the (x, y, z) coordinates of a synthetic trajectory extracted from the Hard Synthetic Trajectories Dataset. It uses angular constraints $50° < \theta < 60°$ and is generated by interpolating between 4 selected positions via spherical linear interpolation (the speed constraint).

4.1.2 Simple Synthetic Data

The synthetic trajectories dataset described in the previous section constitutes a set of constrained trajectories for which it is "hard" for an uncertainty-aware model to accurately class trajectories' exact associated uncertainty. This is because, although uncertainty is constrained via the enforcement of constraints on angles between selected positions and number of positions (speed), the exact uncertainty associated with a trajectory is hidden, defined as some function of angle, distance and speed constraints. As a result, modelling this uncertainty is still challenging - hence the dataset title.

For this reason, we introduce the "Simple Synthetic Trajectories Dataset" (SSTD) to supplement the HSTD. This is a dataset of synthetic head motion trajectories, where the uncertainty associated with any one trajectory is entirely known and defined by a single source. This is achieved by setting trajectories to be based around only three positions (expressed in Euler angles) on the unit sphere: $A = (\frac{7\pi}{8}, \frac{\pi}{6}), B = (\pi, \frac{\pi}{4}), C = (\frac{9\pi}{8}, \frac{\pi}{6})$. Trajectories in the dataset are constructed by generating a random combination of these coordinates (for example, possible trajectories could be ABCA, BCAB, and so forth), such that there are 6 possible trajectories that appear in the dataset. This is an entirely deterministic process, as we simply linearly interpolate between known points in the θ and ϕ axes. However, trajectories are not generated directly from positions A,B and C - they are instead generated by interpolating between points within *some*

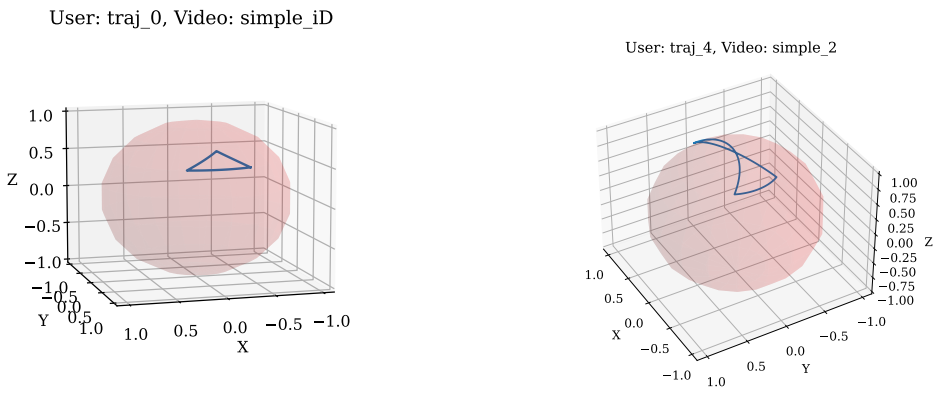

Figure 4.6: Diagram showing example training ($\sigma_{pos} = 0.01$, left) and test ($\sigma_{pos} = 2$, right) traces (represented in the unit sphere) that follow the (x, y, z) coordinates of synthetic trajectories extracted from the Simple Synthetic Trajectories Dataset.

fixed standard deviation of A, B and C. This is the only source of uncertainty in the trajectory (meaning standard deviation magnitude determines trajectory uncertainty).

In the training set for the SSTD, we set the standard deviation over positions to be 0.01. As in the HSTD, we generate 1000 trajectories for training, with each trajectory selected from our 6 candidate trajectories. All trajectories have the same form as those from the HSTD, David-MMSys18 and Xu-CVPR18 datasets. An example training trajectory is shown in figure 4.6. As for the HSTD, we generate multiple training sets with varying levels of uncertainty - each containing 200 trajectories. We vary the standard deviation over positions A, B and C between values $\sigma_{pos} \in [0.05, 0.1, 0.5, 1, 2]$ (an example of a trajectory with a standard deviation of 2 is also shown in figure 4.6). OoD experiments mix test and training samples as for the HSTD.

4.2 Experimental Setup

For all experiments, we implement models that predict future head coordinates for a prediction horizon of $H = 5$ seconds. Since videos are sampled at 5 frames per second, this means predictions encompass the 25 future timesteps following a timestamp t. Similarly, findings in [20] suggest viewers of a new VR scene generally spend the first period of viewing exploring the scene - meaning this period is highly unpredictable. For this reason we do not generate predictions for the first 6 seconds of any input trajectory (chosen to replicate experiments undertaken by Rondon et al. in [4]). Given each video in the David-MMSys18 dataset is 20 seconds long, equating to 100 times-

tamps after subsampling, our model makes 45 sets of $H = 25$ predictions for each input trajectory.

All models were trained using the ADAM optimizer with a learning rate of 5×10^{-4} and otherwise default parameters [58]. Models are trained to convergence, with an early stopping scheme employed to prevent model overfitting (i.e. training is stopped if validation error consistently increases), and a batch size of 128.

4.3 OoD Detection

Section 3.3.1 described how epistemic uncertainty is best evaluated by incrementally shifting the test data distribution away from the training distribution, and setting up repeated binary classification tasks using AUROC score as a metric. However, there is no clear way to measure the distributional shift between two head trajectories in response to VR scenes. We therefore hypothesised that discrete incremental distributional shifts could be implemented by first introducing test trajectories from viewers who were not in the David-MMSys18 training set (we call this the 'Users' OoD case), but who were watching the same training set videos (i.e. viewers might exhibit behaviours outside of the training distribution, but would follow the same regions of interest as in training trajectories). A second shift would be to introduce trajectories from viewers in the David-MMSys18 training set, but instead exploring VR videos outside of the training set (videos from the David-MMSys18 test set - such that viewers would exhibit behaviours seen in the training distribution, but videos may include OoD content). We call this the 'Videos' OoD test case. A third, further, shift would be to use OoD trajectories where *neither* viewers or videos are featured in our model's training set trajectories (OoD trajectories are extracted from the David-MMSys18 dataset such that subjects and videos do not intersect with the training set). We call this the 'Videos & Users' OoD case. Finally, the most extreme shift would be to use test trajectories extracted from an entirely different dataset - for this we use trajectories from the Xu-CVPR18 dataset. We refer to this as 'True OoD'.

The conventional David-MMSys18 test set is made up of 145 trajectories (5 videos, each viewed by 29 subjects). We set each of our OoD test sets to be 20% larger than this (arbitrarily chosen), containing 174 trajectories. For the 'Users' case, we take 145 trajectories from the David-MMSys18 training set, and add 29 trajectories from prior unseen subjects watching iD videos (videos were randomly selected). We do the same for the 'videos' case, randomly selecting and adding 29 trajectories from

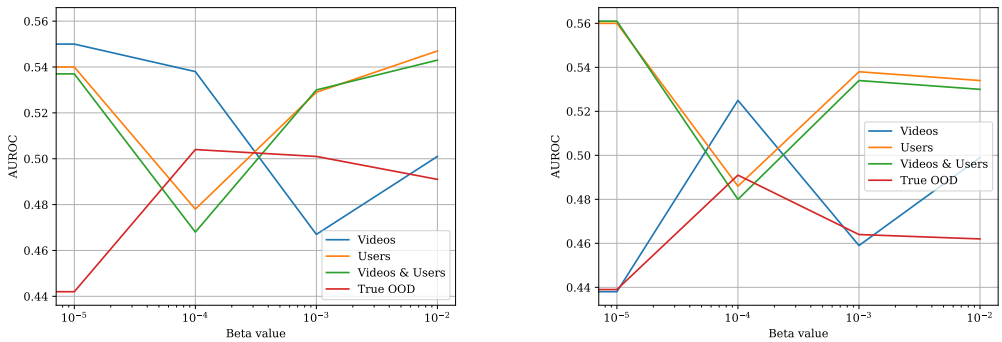

Figure 4.7: AUROC results evaluating (left) **c**-state and (right) **h**-state per-instance rates, generated by the position-only-VIB model when tested on different OoD test cases, when β values are varied from 1×10^{-5} to 1×10^{-2}.

iD subjects watching unseen videos. This fashion continues, selecting 29 additional trajectories that match the constraints of each OoD case and augmenting a training set of previously seen, iD trajectories. As these 4 datasets progressively deviate from the training data, we hypothesise that AUROC scores should progressively increase.

These OoD test cases form our initial evaluation framework for uncertainty estimates for an uncertainty-aware head motion prediction model. Above this, the position-only-VIB model has two factors influencing uncertainty estimation performance: number of mixture components N in the marginal distributions $m_{\phi}^{h,c}(\mathbf{z})$ and the β values controlling the amount information from inputs in latent encodings is penalised. To constrain the search space over possible setups of these two hyperparameters, we only consider cases where β values and mixture component numbers are the same for both the **h** and **c** distributions (i.e. $\beta_h = \beta_c$ and N is constant for $m_{\phi}^{h,c}(\mathbf{z}_{h,c}) = \sum_{i=0}^{N} \alpha_i \cdot \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$). We vary $\beta \in [1 \times 10^{-5} : 1 \times 10^{-2}]$, and vary number of components $N \in [60 : 200]$ (values lower than 60 were tested, but always produced NaN loss results during model training). Note that for all experiments we present both the **h** and **c**-state rates. Figure 4.7 shows AUROC results from experiments on our OoD evaluation framework for varying β with fixed $N = 200$. We present results only for $N = 200$ as there was not found to be any significant change across varying N for all values of β. This figure shows that the per-instance rate is poor at indicating whether trajectories are iD or OoD, with varying distributional shifts. AUROC scores remain close to 0.5, indicating no threshold on rate predictions exists that is close to dissecting iD and OoD ground truth examples correctly (rates could be random with respect to binary labels). Furthermore, AUROC scores are not calibrated with the relative sizes of distributional

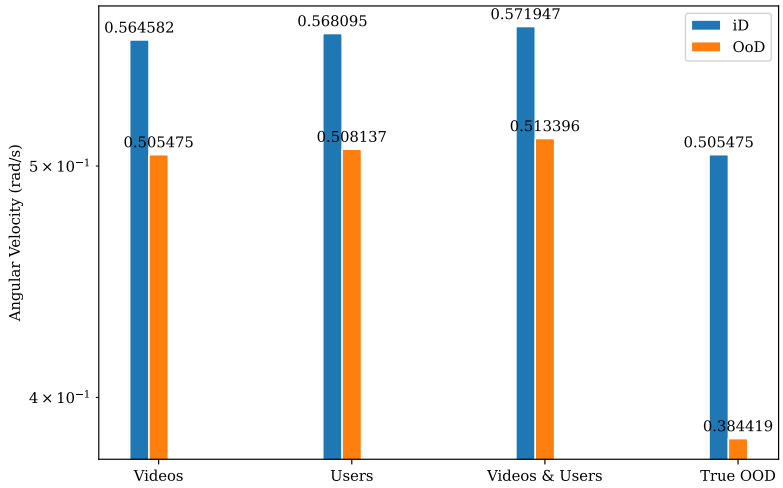

Figure 4.8: Average angular velocity of iD and OoD trajectories for different OoD test cases in our evaluative framework.

shift we hypothesised each OoD case would have. To verify the cause of this failure, we calculate the average angular velocity of iD and OoD trajectories from each OoD test case. We hypothesise that, because the position-only-VIB model considers only a sequence of positions, a difference in velocities may be indicative of distributional shift between iD and OoD trajectories, and that our OoD cases are not significantly different in this respect. Figure 4.8 shows the results of this calculation - iD and OoD trajectories are all extremely close in angular velocity (generally less than 0.2 rad/s from one another) besides those in the True OoD test case. This suggests that the True OoD case may in fact offer a significant distributional shift from the training data, but all other OoD cases are not different enough from training data to be detected by the position-only-VIB model.

4.3.1 Synthetic Data

From the results in figure 4.8, we hypothesise that generally our initial experiments were not an effective test of epistemic uncertainty estimates, because distributional shifts between iD and OoD samples were not large enough. This motivates the evaluation of uncertainty estimates on OoD test cases where distributional shifts between iD and OoD samples can be controlled (and forced to be large) - allowing us to test if uncertainty estimates are effective, what distributional shift they are effective from, and how calibrated they are when effective. Furthermore, figure 4.8 suggested 'True OoD' trajectories *did represent* a significant distributional shift from iD trajectories, that the position-only-VIB did not account for in its uncertainty estimates. Locating

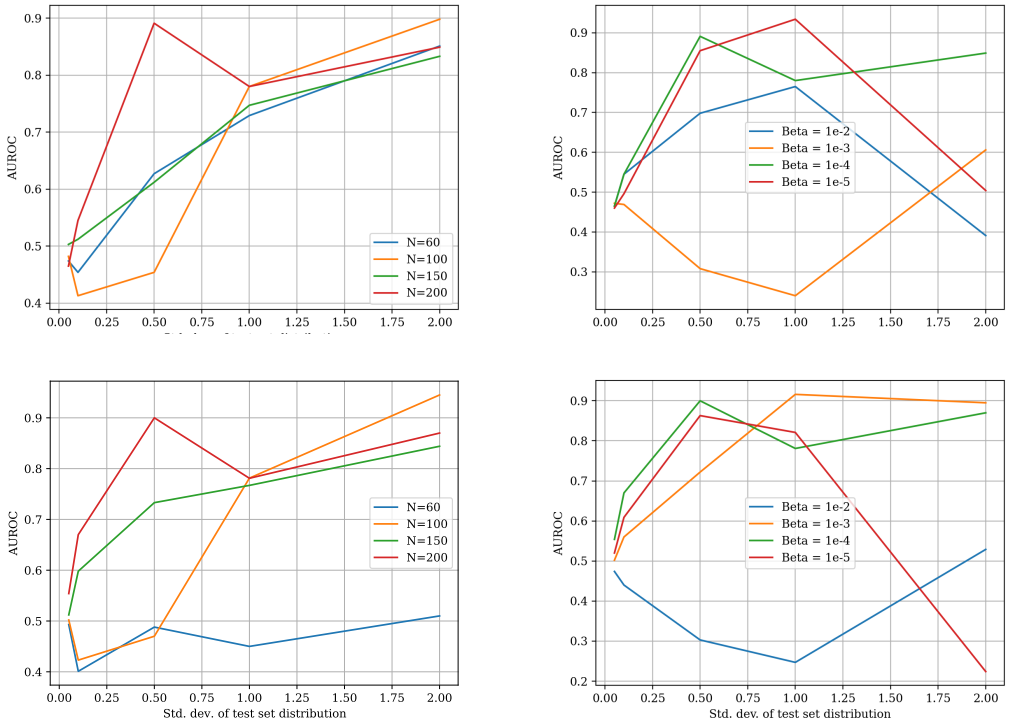

Figure 4.9: Results from OoD experiments on the SSTD dataset, with OoD samples generated using $\sigma_{pos} \in [0.05, 0.1, 0.5, 1, 2]$. These consist of AUROC scores (left) when varying number of components N in the marginal mixture $m_{\phi}^{h,c}$, and (right) when varying the IB coefficient β. Upper figures are AUROC results using the $\mathbf{h}$-state per-instance rate, while lower figures use the $\mathbf{c}$-state per-instance rate.

where uncertainty estimates are effective (expressed through high AUROC scores) using known uncertainties demonstrates whether 'True OoD' trajectories are OoD, and the model is failing, or whether they are not truly OoD at all.

For this task, we train and test the position-only-VIB using trajectories from the HSTD and SSTD described in Sections 4.1.1 and 4.1.2 respectively. Within the SSTD training and test sets (recalling it includes one training set and 5 test sets, with varying standard deviations used across test sets) distributions, and therefore uncertainty, are exactly known. This means we can exactly test the per-instance rate's viability as an epistemic uncertainty estimate for varying amounts of uncertainty, by observing its calibration with differing distances from the training distribution. The values for SSTD test set standard deviations $[0.05, 0.1, 0.5, 1, 2]$ (see Section 4.1.2 were chosen because they represent a range of trajectories varying from those near-to the training distribution ($\sigma_{pos} = 0.05$), to those far from the training distribution ($\sigma_{pos} \in [1, 2]$) - showing the diversity in test cases that is hypothesised to be important by Postels et al.

Figure 4.10: Results from OoD experiments using test trajectories with varying (left) speed and (right) angular constraints (shown as ranges expressed in degrees) from the HSTD. We test the per-instance rate's ability to differentiate between iD and OoD samples - with higher "speed" and angular ranges distant from $50 - 60^{\circ}$ indicating OoD samples are further from the data distribution.

in [43]. We supplement experiments on the SSTD with those on HSTD trajectories, as these represent a body of synthetic trajectories that are more realistic, but can still be used to control distributional shift in OoD test cases (although the exact magnitude shift, and therefore the exact quantity of uncertainty associated with a given shift, between training and test distributions is not known).

The three constraints set on HSTD trajectories are designed to ensure this partial faithfulness to real trajectories. The distance constraint is enforced to replicate the fact that viewers generally take up head positions close to the equator within their possible sphere of positions [20]. The angular constraint is enforced to create in-distribution trajectories that are largely continuous, replicating the fact that real viewers tend to follow continuous trajectories with their heads. The fact that trajectories are manually created means that we can remove the stochastic component of human behaviour, that will occasionally violate this continuity (making modelling epistemic uncertainty a harder task) within in-distribution trajectories from the David-MMSys18 dataset. Finally, speed constraints are enforced to test if our model can account for distributional shifts represented by change in average angular velocity (as between the David-MMSys18 and Xu-CVPR18 datasets) in its uncertainty estimates.

AUROC results on HSTD test trajectories mirror those from our original OoD ex-

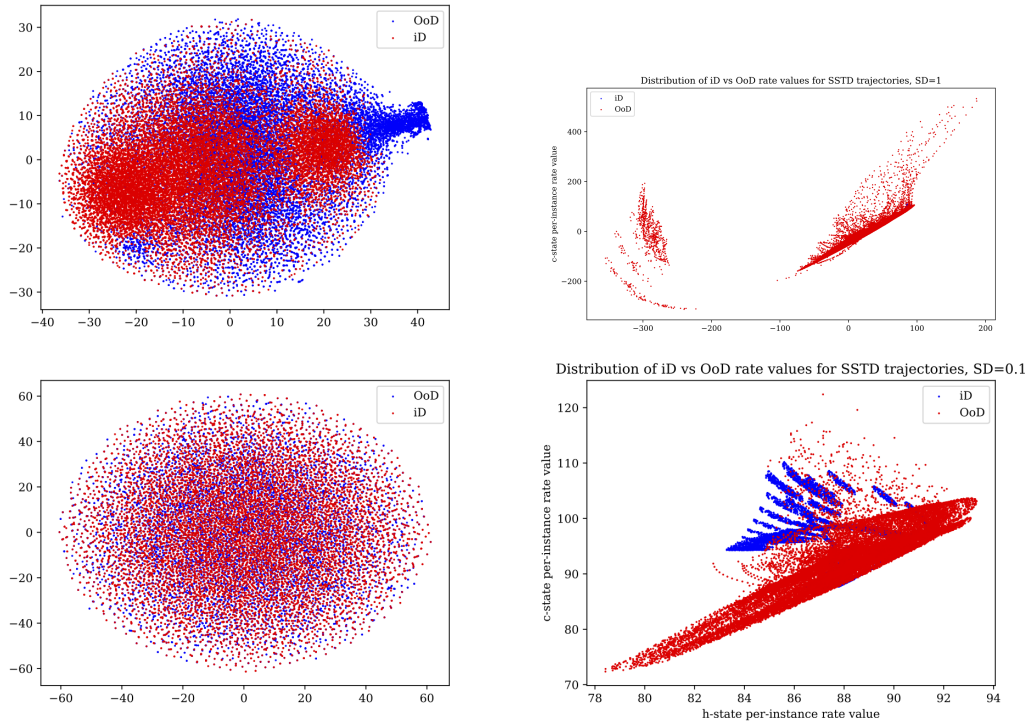

Figure 4.11: Visualisations of (left) samples $\mathbf{z_c}$ from the $\mathbf{c}$-state encoding distribution (e^{c}_{θ}) for iD and OoD input trajectories from the SSTD dataset, with (right) the per-instance rate values generated using these samples. Upper figures display results for $\sigma_{pos} = 1$, while lower figures display results for $\sigma_{pos} = 0.1$.

periments, in that, for various speed and angular constraints in all test cases, AUROC on the per-instance rate remains near to 0.5 (see figure 4.10). This is true for all β and N values, indicating that the position-only-VIB model cannot effective estimate epistemic uncertainty for real or pseudo-real head motion trajectories. By contrast, AUROC results using SSTD test trajectories are far more successful, and are displayed in plots for different values of β and N in figure 4.9. We see AUROC scores increase in line with increasing σ_{pos}, which indicates the position-only-VIB is capable of detecting the distance of a given test input from its training data distribution. For extremely OoD samples (for example, $\sigma_{pos} = 2$), we even see AUROC scores of more than 0.9. Moreoever, it is notable that, for $\sigma_{pos} \leq 0.1$, in the majority of cases we see the per-instance rate in effect acts as a random classifier (AUROC scores tend to 0.5). From plots of traces, trajectories with $\sigma_{pos} \leq 0.1$ likely fall within a range of noise that we might expect from a slightly OoD human head motion trajectory - that our model should be able to predict with confidence (whereupon AUROC scores *should* tend to 0.5). Over-

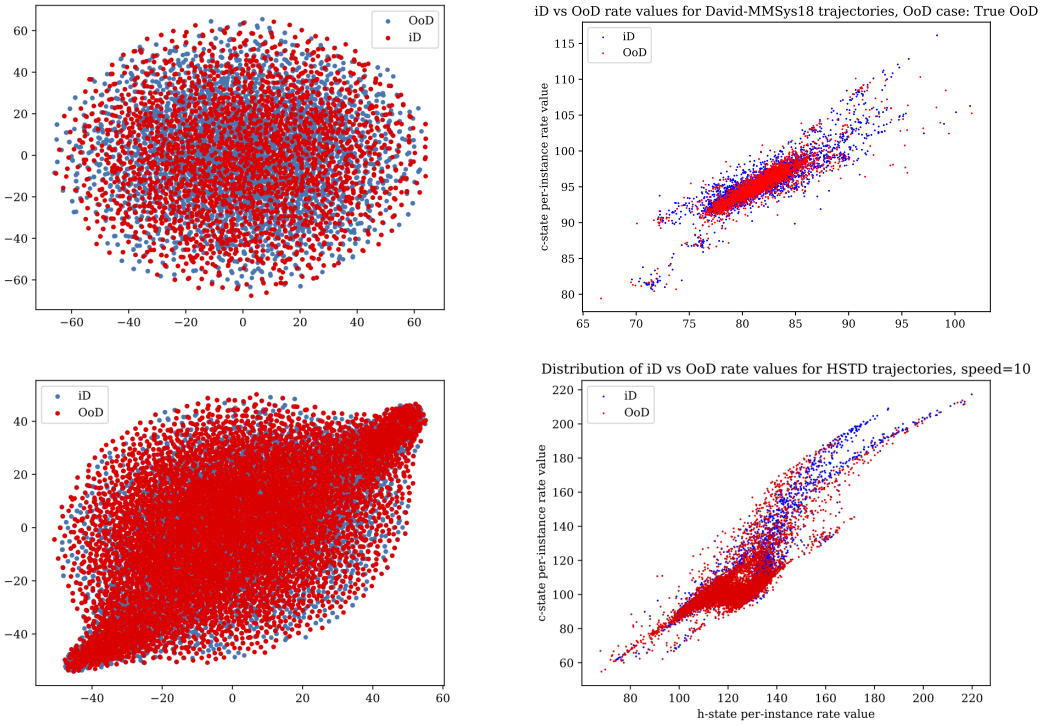

Figure 4.12: Visualisations of (left) samples $\mathbf{z_c}$ from the $\mathbf{c}$-state encoding distribution (e^c_θ) for iD and OoD input trajectories, with (right) the per-instance rate values generated using these samples. Upper figures display results for trajectories from the 'True OoD' David-MMSys18 OoD test case, while lower figures display results for trajectories from the HSTD dataset, for the OoD case with 10 selected positions per trajectory.

all, SSTD experimental results indicate the per-instance rate is a moderately effective uncertainty estimate at a wide range of distances from the training distribution - when trajectories contain a single source of uncertainty.

To interpret AUROC results, we visualise in 2D the distributions of iD and OoD activations (extracting both the sampled $\mathbf{z_h}$ and $\mathbf{z_c}$ vectors generated for iD and OoD test trajectories) over SSTD test cases, as well as HSTD and David-MMSys18 test cases for comparison, using the *t-distributed Stochastic Neighbor Embedding* (t-SNE) method [59], alongside the distribution of rate values generated using these activations. Details of the t-SNE method are described in Appendix D.3.1. Figure 4.11 shows examples of these visualisations for an SSTD OoD test case in which outputted rates gained high AUROC scores ($\sigma_{pos} = 1$), and a case ($\sigma_{pos} = 0.1$) where rate signals generally gained low AUROC scores (tending towards 0.5), while figure 4.12 shows the same plots for example HSTD and original David-MMSys18 OoD test cases. Further

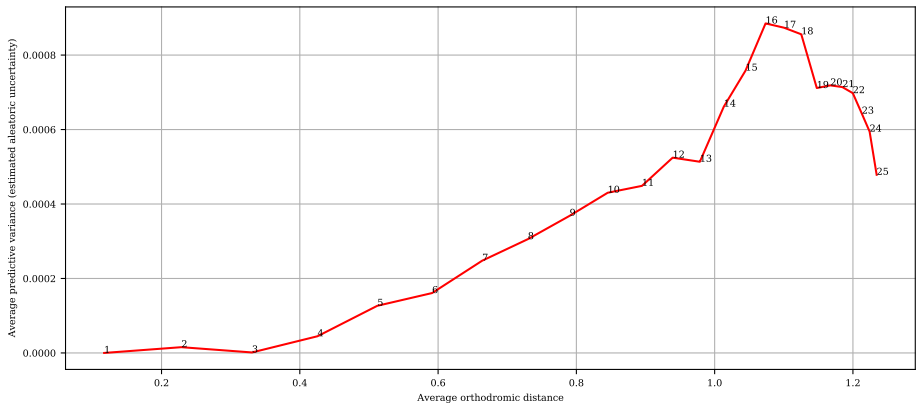

Figure 4.13: A plot of average estimated aleatoric uncertainty against average orthodromic distance generated from mean position-only-VIB model predictions. Numbers labelled on the curve indicate the future timestep $\tau \in [1 : H = 25]$ uncertainty estimates and predictions were made for to generate data points.

examples can be found in Appendix D.3, noting that visualisations of activations are not comparable with one another as t-SNE reduces dimensionality of activations using methods individual and local to each set of activations.

From these visualisations, we note that iD and OoD distributions (in terms of activations and rate value distributions for these test examples) are extremely similar for all HSTD and David-MMSys18 test cases, and are generally similar (although there are some differences between distributions) for SSTD test cases using small σ_{pos}. For SSTD test cases using larger σ_{pos}, we begin to see iD and OoD distributions diverge, with distributions extremely different for $\sigma_{pos} \geq 1$.

4.4 Aleatoric Uncertainty

As described in Section 3.3, evaluation of aleatoric uncertainty estimates is conducted by calculating the correlation between average orthodromic distance and aleatoric uncertainty estimates over the iD test set. We do this by treating the David-MMSys18 test set as a collection of iD trajectories (figure 4.8 shows David-MMSys18 train and test set trajectories share similar angular velocities, suggesting this is a valid assumption), and simply correlating aleatoric uncertainty predictions and error over the test set. To quantify correlation, we use Pearson's and Spearman's rank correlation coefficients as metrics - expressing a measure of the *linear correlation* between two sets of data and extent to which datasets' relationship forms a *monotonic function* respectively

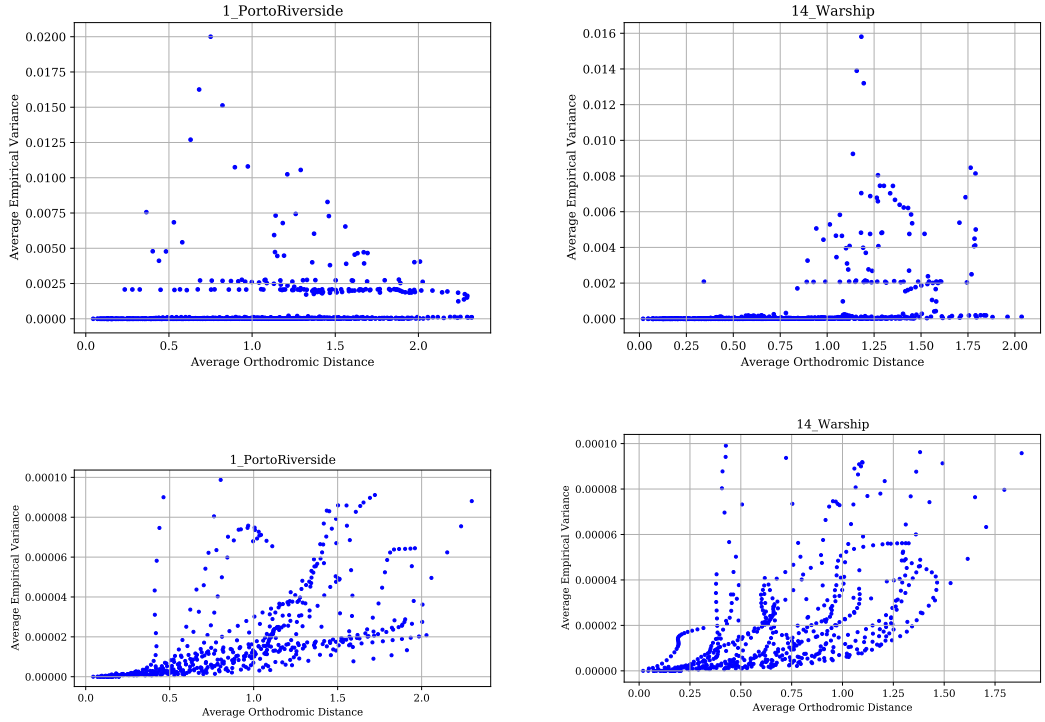

Figure 4.14: Scatter plot for predicted aleatoric uncertainty (defined as the empirical variance of predictions over M latent samples) against orthodromic distance mean prediction error results, averaged over trajectories from the videos (left) '1_PortoRiverside' and (right) '14_Warship'. Lower plots contain the same data as those in the top row, but exclude data points where predictive variance exceeds 1×10^{-4}.

[60]. The position-only-VIB model generates aleatoric uncertainty estimates in parallel with mean regression predictions for each timestep $\tau \in [t+1 : t+H]$. As a result, for evaluation we can directly correlate uncertainty estimates with model error. For each subject viewing each video, we have 45 sets of predictions to correlate, while for each video we have 28 subjects. To generate a base correlation score for uncertainty estimates, we first simply average over predicted uncertainty and mean predictive orthodromic distance error results for each timestep over all timestamps (giving us 25 pairs of uncertainty estimates and error values, each the average of 45 timestamps, for each subject on each video), over all subjects for each video (providing 25 pairs of results averaged over 28 viewers), and finally over all videos. We then correlate the two averaged sets of results. Note that the all results here are generated by repeatedly sampling $M = 4$ times from the encoder distribution and generating the corresponding outputs. However, we see minimal difference in uncertainty results when using up to 50 samples (with experiments abandoned above $M = 50$ due to the sheer time required

Video	**Pearson Correlation**	**Spearman Correlation**
1_PortoRiverside	0.135	0.661
3_PlanEnergyBioLab	0.294	0.722
5_Waterpark	0.394	0.709
14_Warship	0.373	0.745
16_Turtle	0.254	0.814

Table 4.1: Pearson and Spearman correlation results calculated between all aleatoric uncertainty estimates (empirical variances) and mean predictive error results, averaged over trajectories for individual videos in the David-MMSys18 test dataset.

for inference). The position-only-VIB model used to generate these results was the best performing model from OoD Experiments - with a mixture of $N = 200$ Gaussians for the marginal and β values set to 1×10^{-4}.

Base correlation experiments produce Pearson and Spearman correlation scores of 0.912 and 0.794 respectively. A plot of orthodromic distance against uncertainty (averaged over videos) visualising this trend is shown in figure 4.13. We can see from correlation scores and figures that correlation between predicted aleatoric uncertainty and error is strong over the test set, implying we have generated effective aleatoric uncertainty estimates. To verify this at an increased level of granularity, we also calculate correlation between uncertainty and error for each video - with results shown in table 4.1, and scatter plots visualising data for two sample videos shown in figure 4.14. These results suggest that correlation is much lower per-video than when averaged over the whole test set, and that there are quite large fluctuations between videos.

Observing the pattern of aleatoric uncertainty estimates in figure 4.14, we note that a large number of uncertainty estimates appear close to zero (on inspection values average at $\sim 1 \times 10^{-5}$) - with a smaller number of estimates that are extremely large in comparison. We hypothesise that lower correlation results are associated with these estimates (evidenced by the more linear relationship between uncertainty and orthodromic distance in thresholded lower half plots in figure 4.14), and that it is some parameter of videos that is causing these large uncertainty estimates. To test this hypothesis, we conduct an experiment wherein we threshold aleatoric uncertainty predictions to be above or below 16 values in the range 1×10^{-5} to 0.02. We then use the fact that uncertainty estimates and orthodromic distance should be strongly correlated (and thereby large uncertainty estimates should only occur when a large orthodromic distance is expected), by treating orthodromic distance values as a predictor for the bi-

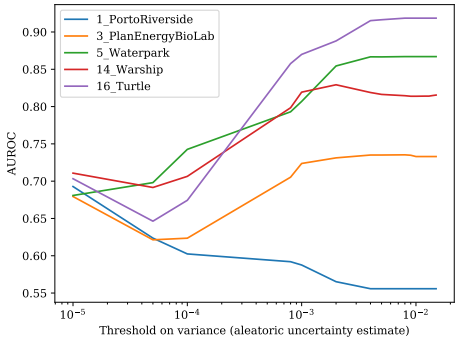

Figure 4.15: AUROC results for using orthodromic distance as a predictor on the binary classification task of whether estimated aleatoric uncertainty exceeds a series of thresholds. Results are generated using orthodromic distances and uncertainty estimates per-video, for all videos in the David-MMSys18 test set.

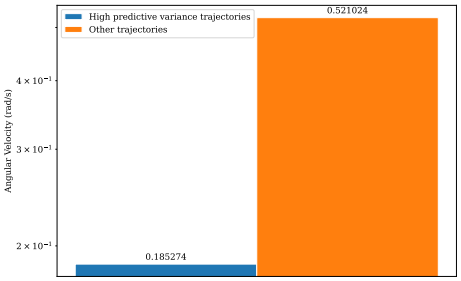

Figure 4.16: Average angular velocity of trajectories that generated predicted aleatoric uncertainty estimates over a threshold of 0.018 when used as input to the position-only-VIB model, compared to average angular velocity of all other trajectories.

nary classification task of whether aleatoric uncertainty values exceed each threshold using AUROC as a metric.

The results of this experiment, shown in figure 4.15, confirm our hypothesis. Not only do AUROC scores at high thresholds align roughly with correlation scores from table 4.1 (i.e. videos with low correlation scores align with those that have low AUROC scores at high thresholds), but they also align roughly with number of uncertainty estimates above high thresholds (see Appendix D.4 for the counts of uncertainty estimates exceeding some example thresholds for each video).

Finally, to allow qualitative conclusions to be made about what the hypothesised parameter of videos causing these failures might be, we inspect uncertainty estimates made *per-viewer* (i.e. for individual trajectories on videos). We extract those test trajectories where an uncertainty estimate of high magnitude (over a threshold of 0.018 - chosen as, heuristically, there are unlikely to be many accurate uncertainty estimates made over this threshold) has been predicted. We then compute the average angular velocity exhibited over these trajectories, and compare it to the rest of test and training examples - with results shown in figure 4.16. This figure shows extracted "unnaturally high predicted uncertainty" trajectories display significantly lower average angular velocity than the David-MMSys18 dataset at large. Qualitatively, this indicates low ve-

locity trajectories may be problematic for the position-only baseline to predict future head positions for.

Chapter 5

Discussion & Conclusions

5.1 Discussion

5.1.1 Epistemic uncertainty & per-instance rate

Results from Section 4.3 indicate that the per-instance rate, as an uncertainty estimate, is not well calibrated with the true epistemic uncertainty associated with head motion trajectories drawn from foreign data distributions (with respect to the position-only-VIB training distribution). Furthermore, visualisations show the per-instance rate does not behave as a simple uncertainty estimate (as hoped at the beginning of experiments), where increased uncertainty in trajectory results in increased rate value. Instead, it appears that low uncertainty and in-distribution trajectories (i.e. trajectories similar to those from the training distribution) result in rate values that do not vary greatly, and lie closely (with a small amount of noise) on well defined manifolds. Notably, these manifolds are more clearly defined for larger values of β (see Appendix D.3.4 for a demonstration of this finding), although our results indicate larger β do not result in better uncertainty estimates. Our results, from experiments on the SSTD dataset, indicate that increasingly uncertain trajectories then generate rate values that appear initially more noisy on manifolds, before diverging from manifolds. Divergence from manifolds often presents as rate values spreading to much lower magnitudes (for example, spreading from 84 for iD trajectories to -300 for extremely OoD trajectories).

This divergence of rate values from the 'iD manifold' was never observed for actual human head motion trajectories, and can be observed minimally for extremely OoD HSTD trajectories (for trajectories with a "speed" of 10, shown in Appendix D.3). For VR streaming systems using the rate signal to account for uncertainty, without diver-

gence from the manifold it is impossible for them to differentiate between high and low uncertainty trajectories. As a result, we say the per-instance rate is not *well-calibrated* with epistemic uncertainty, because values never diverge from the iD manifold where they should. This qualitative analysis is based upon generated visualisations of rate distributions, and, significantly, agrees with our AUROC results from OoD experiments using the David-MMSys18 and HSTD datasets, where we saw consistent AUROC scores of close to 0.5. This non-divergence of OoD rate values from the iD manifold for real and pseudo-real (HSTD) trajectories, combined with the divergence we observed for SSTD trajectories (where the number of modes in the data to be learned by the model was very small), suggests that the position-only-VIB model is not learning to correctly emphasise the aspects of trajectories that are key sources of epistemic uncertainty in representations. For example, distributional shifts in the angular velocity of OoD trajectories were not detected in 'True OoD' experiments on the David-MMSys18 dataset, and, furthermore, t-SNE visualisations of activations regularly show OoD trajectories are represented using similar feature tensors as iD trajectories in all test cases. This may be related to the problem of *feature collapse* (introduced by van Amersfoot et al. in [1]) - wherein out-of-distribution input data is mapped to in-distribution feature representations, meaning their likelihoods will be similar under marginals $m_{\phi}^{h,c}$.

Finally, our AUROC results show that neither of samples from the **c**-state and **h**-state stochastic encoding distributions are consistently more effective for generating uncertainty than one another. With respect to the role played by the hyperparameters β and number of mixture coefficients N, our results show that larger N (we gained best results using $N = 200$) and moderate values of β (our best AUROC results were observed for $\beta = 1 \times 10^{-4}$) work best to generate effective uncertainty estimates. These findings suggest that a moderate amount of information from model inputs is necessary to estimate uncertainty associated with outputs (i.e. we cannot penalise too heavily information from inputs being present in latent encodings), and that the distribution over activations being fitted by variational marginals is highly multi-modal. Note that, for values of $N \geq 200$ inference of the per-instance rate becomes significantly slower, meaning there may be a trade-off between uncertainty estimate quality and efficiency.

5.1.2 Aleatoric Uncertainty

Our results from Section 4.4 indicate that the empirical variance of position-only-VIB predictions, generated from multiple samples from encoding distributions, is an ef-

fective estimate of model aleatoric uncertainty for head motion prediction. These estimates correlate strongly with the orthodromic distance between ground truth and predicted future head positions. However, our results show that aleatoric uncertainty estimates under this method are unable to account for low velocity head movement trajectories - with artificially large estimates for these trajectories being generated by the position-only-VIB model.

Observing figure 4.13, we can also see that the strong correlation between orthodromic distance and aleatoric uncertainty falls steeply after prediction step $\tau = 16$. This indicates aleatoric uncertainty estimates are not effective over longer prediction horizons. In combination with aleatoric uncertainty estimates failing for low velocity trajectories, we believe that this is not a failure of the UQ method itself - but rather a failure of the position-only baseline model. Findings in [2], [4] and [20] indicate that head positions for prediction horizons longer than 2 - 3 seconds (corresponding to 10-15 timesteps when sampled at 5 frames per second) start to become largely determined by visual content in VR videos, rather than previous head positions. This information directly explains the fall in correlation we see in our experiments, as the position-only-VIB model ignores visual content, meaning it is likely not effective for prediction horizons longer than 15-16 timesteps. Similarly, we hypothesise that low velocity periods in trajectories could be as a result of subjects inspecting visual content in VR videos, which again cannot be accounted for under the position-only-VIB model. This hypothesis is validated by the fact that videos with the greatest number of artificially large aleatoric uncertainty estimates (‘1_PortoRiverside’ and ‘3_PlanEnergyBioLab’) are known as *Exploration* videos (as defined by Almquist et al. in [61]) wherein video content incentivizes users to explore and inspect different aspects of VR visuals. Head positions in Exploration videos are therefore more likely to be determined by video content.

5.2 Conclusions

This thesis investigated extending a deep learning model for head motion prediction in VR environments to be capable of quantifying uncertainty in its predictions. The model was comprised of an encoder-decoder LSTM architecture that accepted a sequence of past head position coordinates as input. The extension to this model introduced stochastic latent variables constrained under an information bottleneck objective - such that at test time the likelihood of input head motion trajectories could be evalu-

ated to estimate epistemic uncertainty. The variance of predictions made from sampled latent variables acted as an aleatoric uncertainty estimate. We developed an evaluation framework for epistemic uncertainty estimates based on distributional shift and out-of-distribution detection using existing head motion trajectory datasets, and introduced two datasets of synthetic trajectories with constrained uncertainty and varying amounts of faithfulness to real head motion trajectories. We evaluated aleatoric uncertainty estimates using correlation with model mean predictive error on real head motion trajectories.

Results on the baseline evaluation framework showed our model's epistemic uncertainty estimates could not clearly distinguish between in and out-of distribution (with varying distributional shift) head motion trajectories, a finding that was repeated for our dataset of realistic synthetic trajectories. However, for our dataset of simplified synthetic trajectories, generated with a single constrained source of uncertainty, our results show epistemic uncertainty estimates clearly distinguish between in-distribution and out-of-distribution trajectories, given a sufficient difference between distributions. With visualisations of the distributions of epistemic uncertainty estimates, and the latent encodings they were generated from, we hypothesise that failures of epistemic uncertainty estimates are due to feature collapse - where in and out-of distribution samples are mapped to similar feature representations.

Evaluation of generated aleatoric uncertainty estimates showed they have strong correlation with model predictive error up to prediction horizons of roughly 3 seconds. However, we show aleatoric uncertainty estimates struggle for low velocity head motion trajectories - hypothesising that this failure, and failures in estimates past prediction horizons of 3 seconds, are due to the influence of VR video content on head positions. This factor can't be accounted for under our chosen prediction model.

Given our model's failures, the main concern of future work should be first to investigate the problem of feature collapse in the position-only-VIB model, identifying if this is the cause of issues found here, before exploring possible methods for ensuring latent encodings are sufficiently different for in-distribution and out-of-distribution input head motion trajectories. Ideas from [1] may be useful for treating this problem. Similarly, if the problem of feature collapse is treated, future work should adapt the successful method for use with TRACK, the current state of the art in deep head motion prediction models. This will require some additional exploration of the optimal positions to place latent variables, taking care that aleatoric uncertainty estimates correlate well with head motion strongly influenced by VR video content.

Bibliography

[1] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9690–9700. PMLR, 13–18 Jul 2020.

[2] Miguel Fabián Romero Rondón, Lucile Sassatelli, Ramón Aparicio-Pardo, and Frédéric Precioso. A unified evaluation framework for head motion prediction methods in 360° videos. In *Proceedings of the 11th ACM Multimedia Systems Conference*, MMSys '20, page 279–284, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Laura Feinstein. 'beginning of a new era': how culture went virtual in the face of crisis. *The Guardian*. [Online] Available from: `https://www.theguardian.com/culture/2020/apr/08/art-virtual-reality-coronavirus-vr` (Accessed 30/03/2021).

[4] Miguel Fabián Romero Rondón, Lucile Sassatelli, Ramón Aparicio Pardo, and Frédéric Precioso. Track: a multi-modal deep architecture for head motion prediction in 360° videos. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2586–2590, 2020.

[5] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 1190–1198, New York, NY, USA, 2018. Association for Computing Machinery.

[6] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. Tiling in interactive panoramic video: Approaches and evaluation. *IEEE Transactions on Multimedia*, 18(9):1819–1831, 2016.

[7] H. Hristova, G. Simon, X. Corbillon, A. Devlic, and V. Swaminathan. Heterogeneous spatial quality for omnidirectional video. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(4):1411–1424, 2021.

[8] Savino Dambra, Giuseppe Samela, Lucile Sassatelli, Romaric Pighetti, Ramon Aparicio-Pardo, and Anne-Marie Pinna-Déry. Film Editing: New Levers to Improve VR Streaming. In *Proceedings of the 9th ACM Multimedia Systems Con-*

ference, MMSys '18, page 27–39, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1161–1170, 2016.

[10] Mai Xu, Yuhang Song, Jianyi Wang, Minglang Qiao, Liangyu Huo, and Zulin Wang. Predicting head movement in panoramic video: A deep reinforcement learning approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(11):2693–2708, Nov 2019.

[11] Y. Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, J. Yu, and Shenghua Gao. Gaze prediction in dynamic 360° immersive videos. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5333–5342, 2018.

[12] A. Deniz Aladagli, Erhan Ekmekcioglu, Dmitri Jarnikov, and Ahmet Kondoz. Predicting head trajectories in 360° virtual reality videos. In *2017 International Conference on 3D Immersion, IC3D 2017 - Proceedings*, pages 1–6, United States, January 2018. Institute of Electrical and Electronics Engineers. 7th International Conference on 3D Immersion (IC3D 2017), IC3D 2017 ; Conference date: 11-12-2017 Through 12-12-2017.

[13] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Wang. Cub360: Exploiting cross-users behaviors for viewport prediction in 360 video adaptive streaming. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2018.

[14] Yunqiao Li, Yiling Xu, Shaowei Xie, Liangji Ma, and Jun Sun. Two-layer fov prediction model for viewport dependent streaming of 360-degree videos. In Xingang Liu, Dai Cheng, and Lai Jinfeng, editors, *Communications and Networking*, pages 501–509, Cham, 2019. Springer International Publishing.

[15] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360° video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV'17, page 67–72, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Y Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[17] A. P. Dawid. The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610, 1982.

[18] Alex Alemi, Ian Fischer, Josh Dillon, and Kevin Murphy. Deep variational information bottleneck. In *5th International Conference on Learning Representations*, Palais des Congrès Neptune, Toulon, France, April 24 - 26 2017.

[19] Alex Alemi, Ian Fischer, and Josh Dillon, editors. *Uncertainty in the Variational Information Bottleneck*, 2018.

[20] Vincent Sitzmann, Ana Serrano, Amy Pavel, Maneesh Agrawala, Diego Gutierrez, Belen Masia, and Gordon Wetzstein. Saliency in VR: How do people explore virtual environments? *IEEE Transactions on Visualization and Computer Graphics*, 24(4):1633–1642, April 2018.

[21] Guanbin Li and Yizhou Yu. Visual saliency based on multiscale deep features. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5455–5463, 2015.

[22] Junting Pan, Elisa Sayrol, Xavier Giró-i Nieto, Kevin McGuinness, and Noel OConnor. Shallow and deep convolutional networks for saliency prediction. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 598–606, 06 2016.

[23] X. Huang, C. Shen, X. Boix, and Q. Zhao. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 262–270, 2015.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[26] Graham Neubig. Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. *arXiv e-prints*, page arXiv:1703.01619, March 2017.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[28] *MathWorld*-A Wolfram Web Resource. "Great Circle.". `https://mathworld.wolfram.com/GreatCircle.html`. Accessed: 21 June 2021.

[29] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Mach Learn*, 110:457–506, March 2021.

[30] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, Vladimir Makarenkov, and Saeid Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. In *35th AAAI conference on Artificial Intelligence*, Virtual Conference, Feb. 6-9 2021.

[31] Andreas Damianou. *Deep Gaussian Processes and Variational Propagation of Uncertainty*. PhD thesis, University of Sheffield, 2015.

[32] C. Turchetti. *Stochastic Models of Neural Networks*. IOS Press, NLD, 2004.

[33] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users. *arXiv e-prints*, page arXiv:2007.06823, July 2020.

[34] Florian Wenzel, Kevin Roth, Bastiaan Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10248–10259. PMLR, 13–18 Jul 2020.

[35] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6405–6416, Red Hook, NY, USA, 2017. Curran Associates Inc.

[36] Samarth Sinha, Homanga Bharadhwaj, Anirudh Goyal, Hugo Larochelle, Animesh Garg, and Florian Shkurti. Diversity inducing Information Bottleneck in Model Ensembles. In *35th AAAI conference on Artificial Intelligence*, Virtual Conference, Feb. 6-9 2021.

[37] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep Ensembles: A Loss Landscape Perspective. *arXiv e-prints*, page arXiv:1912.02757, December 2019.

[38] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations*, 2020.

[39] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[40] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[41] Shiyu Liang, Yixuan Li, and R. Srikant. Principled detection of out-of-distribution examples in neural networks. *CoRR*, abs/1706.02690, 2017.

[42] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *CoRR*, abs/1605.08803, 2016.

[43] Janis Postels, Hermann Blum, Cesar Cadena, Roland Siegwart, Luc Van Gool, and Federico Tombari. Quantifying aleatoric and epistemic uncertainty using density estimation in latent space. *arXiv preprints*, abs/2012.03082, 2020.

[44] Sergey Prokudin, Peter V. Gehler, and Sebastian Nowozin. Deep directional statistics: Pose estimation with uncertainty quantification. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IX*, volume 11213 of *Lecture Notes in Computer Science*, pages 542–559. Springer, 2018.

[45] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7167–7177, Red Hook, NY, USA, 2018. Curran Associates Inc.

[46] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[47] Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. *Proceedings of the 37th Allerton Conference on Communication, Control and Computation*, 49, 07 2001.

[48] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *2015 IEEE Information Theory Workshop, ITW 2015*, March 2015.

[49] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. *arXiv preprint*, abs/1907.02392, 2019.

[50] Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2931–2940, 2019.

[51] Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[52] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[53] I. Higgins, L. Matthey, A. Pal, Christopher P. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[55] I. Fischer. Private communication, June-July 2021.

[56] Erwan J. David, Jesús Gutiérrez, Antoine Coutrot, Matthieu Perreira Da Silva, and Patrick Le Callet. A dataset of head and eye movements for 360° videos. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 432–437, New York, NY, USA, 2018. Association for Computing Machinery.

[57] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3):245–254, July 1985.

[58] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[59] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[60] Wayne W. Daniel. *Applied nonparametric statistics*. The Duxbury advanced series in statistics and decision sciences. PWS-KENT, Boston, Mass, second edition. edition, 1990.

[61] Mathias Almquist, Viktor Almquist, Vengatanathan Krishnamoorthi, Niklas Carlsson, and Derek Eager. The prefetch aggressiveness tradeoff in 360° video streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 258–269, New York, NY, USA, 2018. Association for Computing Machinery.

[62] Laurens van der Maaten. t-Distributed Stochastic Neighbor Embedding. https://lvdmaaten.github.io/tsne/.

Appendix A

Previous Head Motion Prediction Models

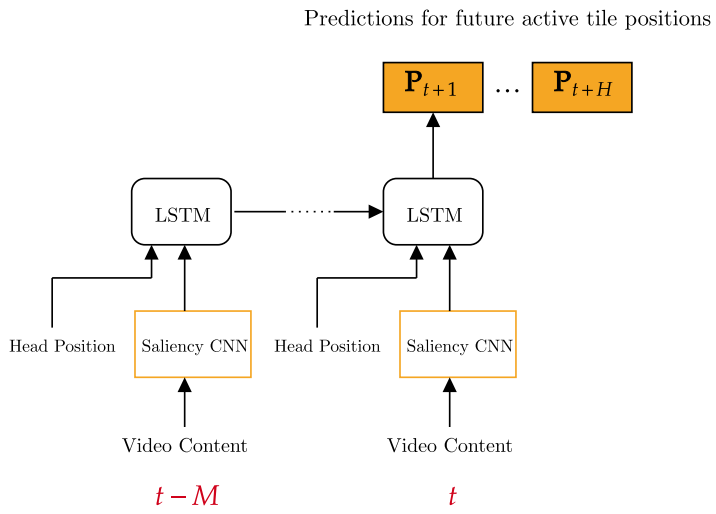

Figure A.1: Deep head motion prediction model architecture used by Fan et al. in [15]. Head positions static saliency maps are concatenated for input to an LSTM for processing as a time series. $\mathbf{P}_t$ indicates active tile coordinates at timestep t.

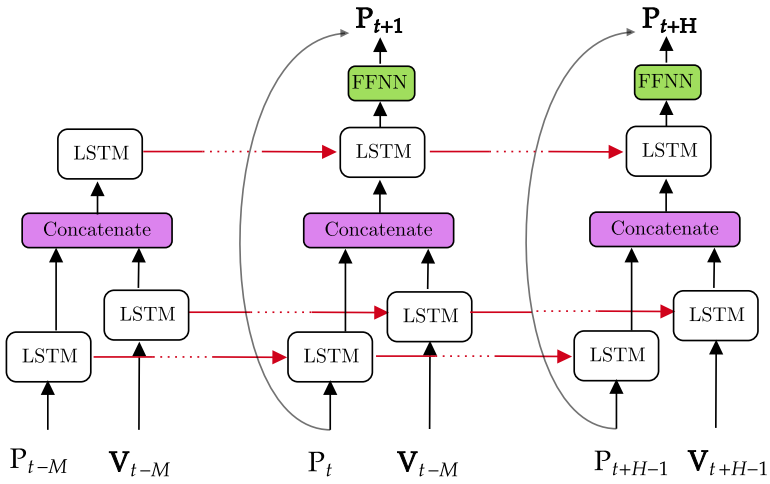

Figure A.2: TRACK head motion prediction model from Rondon et al. in [4]. $\mathbf{P}_t$ and $\mathbf{V}_t$ indicate head position coordinates $[\theta_t, \phi_t]$ and saliency maps (from visual content) respectively at timestep t.

Appendix B

Theoretical motivations behind models

B.1 VIB

One of the key innovations of VAEs is that, under an objective of maximising the log likelihood of data samples $\log p(\mathbf{x})$, they can learn a set of stochastic latent variables $\mathbf{z}$ that model the generative factors of the data, drawn from an approximate posterior $e_\theta(\mathbf{z}|\mathbf{x})$ (regarding density estimation, they model the data density via a learnt distribution over latent variables, along with a transformation from latent space). Crucially, this process is driven by an *encoder* $e_\theta(\mathbf{z}|\mathbf{x})$ for latent variable inference, and a *decoder* $q_\psi(\mathbf{x}|\mathbf{z})$ for generation from latent variables, with parameters θ, ϕ learned as neural networks.

The VIB approach replicates this idea in a supervised learning formulation, maximising the log likelihood of outputs $\log p(\mathbf{y}|\mathbf{x})$, to learn a set of stochastic latent variables that model the factors of inputs that are generative for outputs. This is identifiable from a conventional supervised neural network, which acts discriminatively. Note that stochastic $\mathbf{z}$ also mean outputs $\mathbf{y}$ are treated as random variables, again with an encoder for latent variable inference and a decoder for output value generation. Additionally, to reduce the dimensionality of latent encodings (and motivated by the success of information theoretic constraints in the β-VAE architecture [53]) the VIB approach constrains them to have low complexity, but be effective for output computation, using an information bottleneck objective during training - as shown in equation B.1.

$$\max\ I(\mathbf{z};\mathbf{y}) \text{ subject to } I(\mathbf{z};\mathbf{x}) \leq I_c \tag{B.1}$$

The IB objective states that the mutual information between inputs and latent encodings, shown as $I(\mathbf{z};\mathbf{x})$, should not exceed some constant I_c - while latent encodings should still be maximally informative, and thereby effective, for computation of outputs. Effectiveness is enforced in the objective by maximising mutual information between encodings and outputs $I(\mathbf{z};\mathbf{y})$. These constraints supervise over latent variables, and allow encodings to be significantly lower dimensional than would otherwise be the case, by penalising the presence of excess information from inputs. The IB objective can be written as an objective function using a langrange multiplier β:

$$\max\ I(\mathbf{z};\mathbf{y}) - \beta I(\mathbf{z};\mathbf{x}) \tag{B.2}$$

where $\beta \geq 0$ controls the amount information from inputs is penalised (i.e. the "size of the information bottleneck" [19]).

In practise, calculating $I(\mathbf{z};\mathbf{y})$ and $I(\mathbf{z};\mathbf{x})$ analytically requires computation of difficult and intractable integrals (see Appendix B.1 for more details). As a result, the original VIB architecture is made up of a stochastic encoder $e_\theta(\mathbf{z}|\mathbf{x})$ from which latent variables (i.e. encodings) can be sampled, a stochastic decoder $q_\psi(\mathbf{y}|\mathbf{z})$ from which outputs can be generated, and a variational marginal distribution over latent variables $m(\mathbf{z})$ [18]. These factors can be combined for optimisation under a single objective:

$$\max_{\theta,\phi,\psi} \mathbb{E}_{p(\mathbf{x},\mathbf{y})e_\theta(\mathbf{z}|\mathbf{x})}\left[\log q_\psi(\mathbf{y}|\mathbf{z}) - \beta \log \frac{e_\theta(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})}\right] \tag{B.3}$$

Calculating $I(\mathbf{z};\mathbf{y})$ analytically relies on calculating $p(\mathbf{y}|\mathbf{z})$, defined under the following equation:

$$p(\mathbf{y}|\mathbf{z}) = \int \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z})} d\mathbf{x} \tag{B.4}$$

This is intractable in deep learning problems, as $p(\mathbf{y}|\mathbf{x})$ is not known a priori. As a result, the VIB model uses a variational approximation to $p(\mathbf{y}|\mathbf{z})$ that acts as the decoder, as in VAEs. Similarly, the mutual information term between latent variables and inputs $I(\mathbf{z};\mathbf{x})$ relies on computing a difficult marginal distribution over latent variables $p(\mathbf{z}) = \int p(\mathbf{z}|\mathbf{x})p(\mathbf{x})d\mathbf{x}$. As a result, the VIB approach uses a variational marginal over latent variables $m(\mathbf{z})$ [18].

The final VIB architecture is therefore doubly stochastic - formed from a stochastic encoder $e_\theta(\mathbf{z}|\mathbf{x})$ from which latent variables (i.e. encodings) can be sampled, a stochastic decoder $q_\psi(\mathbf{y}|\mathbf{z})$ from which outputs can be generated, and a variational marginal $m(\mathbf{z})$ that constrains the distribution of latent encodings over the encoding space. These factors can be combined for optimisation under a single objective:

$$\max_{\theta,\phi,\psi} \mathbb{E}_{p(\mathbf{x},\mathbf{y})e_\theta(\mathbf{z}|\mathbf{x})}\left[\log q_\psi(\mathbf{y}|\mathbf{z}) - \beta \log \frac{e_\theta(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})}\right] \tag{B.5}$$

Where the term $\log \frac{e_\theta(\mathbf{z}|\mathbf{x})}{m(\mathbf{z})}$ is used (under an expectation) when an analytic solution for the Kullback-Leibler (KL) Divergence $\mathrm{KL}\left[e_\theta(\mathbf{z}|\mathbf{x})\|m(\mathbf{z})\right]$ is not possible. The KL-divergence can be thought of as a metric for the difference between two distributions - such that minimising $\mathrm{KL}\left[e_\theta(\mathbf{z}|\mathbf{x})\|m(\mathbf{z})\right]$ means training the encoder distribution to be easily encoded using information theoretic arguments under the marginal. Note that in practise Alemi et al. enforce this doubly-stochastic architecture using feedforward neural networks for both encoder and decoder, before placing Gaussian (varying between diagonal and fully covariant) and Categorical (sometimes softmax) distributions over the encoder and decoder network outputs respectively. To train the encoder using gradient descent, since gradients cannot be propagated through the sampling process $\mathbf{z} \sim e_\theta(\mathbf{z}|\mathbf{x})$, Alemi et al. use the *reparameterisation trick* as introduced for training VAEs in [39]. This expresses samples $\mathbf{z}$ as a differentiable transformation of some random variable $\boldsymbol{\varepsilon}$ (which we do not require gradients for), to be generated by evaluating:

$$e_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_e, \boldsymbol{\sigma}_e^2) = \boldsymbol{\mu}_e + \boldsymbol{\sigma}_e^2 \cdot \boldsymbol{\varepsilon}; \boldsymbol{\varepsilon} \sim \mathcal{N}(0,1) \tag{B.6}$$

We can then calculate gradients with respect to the encoder parameters $\boldsymbol{\mu}_e, \boldsymbol{\sigma}_e$ via standard backpropagation, since they are used as deterministic variables during sampling.

For the variational marginal, in their original 2017 paper [18] Alemi et al. used a simple, non-parametric distribution - generally a K-dimensional spherical Gaussian such that $m(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0,I)$. However, to allow for uncertainty estimation, in their following 2018 paper [18] they used a parametric marginal such that it could be learned in parallel with the encoder-decoder model parameters during training. This means that, at the end of training, the marginal now models the distribution of latent encodings over the training data (meaning $m_\phi(\mathbf{z})$, where ϕ are the marginal parameters, models the density of the training data distribution in the lower dimensional encoding space). Alemi et al. then use this density model for UQ by taking the KL-divergence between it and the encoder distribution for a given test input - i.e. $\mathrm{KL}\big[e_\theta(\mathbf{z}|\mathbf{x})\|m_\phi(\mathbf{z})\big]$ [19]. They call the output of this computation the *per-instance rate* (or *rate* for short). Since $m_\phi(\mathbf{z})$ models network behaviour within the training data distribution, and at test time we are computing the likelihood of encodings under this distribution, as an uncertainty estimate the rate measures epistemic uncertainty.

Appendix C

Source Code

C.1 VIB Code

C.1.1 Original VIB Code

This subsection holds code imported from Alex Alemi's GitHub demonstration[1].

```
from tensorflow.examples.tutorials.mnist import input_data
mnist_data = input_data.read_data_sets('/tmp/mnistdata', validation_size
    =0)
images = tf.placeholder(tf.float32, [None, 784], 'images')
labels = tf.placeholder(tf.int64, [None], 'labels')
one_hot_labels = tf.one_hot(labels, 10)

def encoder(images):
    net = layers.relu(2*images-1, 1024)
    net = layers.relu(net, 1024)
    params = layers.linear(net, 512)
    mu, rho = params[:, :256], params[:, 256:]
    encoding = ds.NormalWithSoftplusScale(mu, rho - 5.0)
    return encoding

prior = ds.Normal(0.0, 1.0)

def decoder(encoding_sample):
    net = layers.linear(encoding_sample, 10)
    return net

with tf.variable_scope('encoder'):
    encoding = encoder(images)
with tf.variable_scope('decoder'):
    logits = decoder(encoding.sample())
with tf.variable_scope('decoder', reuse=True):
    many_logits = decoder(encoding.sample(12))
```

[1] Available from https://github.com/alexalemi/vib_demo/blob/master/MNISTVIB.ipynb

```
class_loss = tf.losses.softmax_cross_entropy(
    logits=logits, onehot_labels=one_hot_labels) / math.log(2)
BETA = 1e-3
info_loss = tf.reduce_sum(tf.reduce_mean(
    ds.kl_divergence(encoding, prior), 0)) / math.log(2)
total_loss = class_loss + BETA * info_loss
```

C.1.2 Adapted VIB code

This subsection holds code adapted from Alex Alemi's GitHub demonstration within the context of the position-only baseline model.

```
def create_pos_only_model_7(M_WINDOW, H_WINDOW, BETA_h=1e-5, BETA_c=1e-6)
    :
    # Defining model structure
    encoder_inputs = Input(shape=(M_WINDOW, 2))
    decoder_inputs = Input(shape=(1, 2))
    lstm_layer_enc = LSTM(1024, return_sequences=True, return_state=True)
    lstm_layer_dec = LSTM(512, return_sequences=True, return_state=True)
    h_state_mapper = Dense(1024, activation='linear', name='h_state_dist')
    c_state_mapper = Dense(1024, activation='linear', name='c_state_dist')
    decoder_dense_mot = Dense(2, activation='sigmoid')
    decoder_dense_dir = Dense(2, activation='tanh')
    decoder_params = Dense(2,activation='relu')
    To_Position = Lambda(toPosition)

    # Encoding
    encoder_outputs, state_h, state_c = lstm_layer_enc(encoder_inputs)
    state_h = h_state_mapper(state_h)
    state_c = c_state_mapper(state_c)
    mu_h, sigma_h = state_h[:, :512], state_h[:, 512:]
    mu_c, sigma_c = state_c[:, :512], state_c[:, 512:]
    state_h_distribution = tfp.distributions.MultivariateNormalDiag(mu_h,
        softplus(sigma_h - 5))
    state_h = state_h_distribution.sample()
    state_h_sample = state_h
    state_c_distribution = tfp.distributions.MultivariateNormalDiag(mu_c,
        softplus(sigma_c - 5))
    state_c = state_c_distribution.sample()
    state_c_sample = state_c
    states = [state_h, state_c]

    states = [state_h, state_c]

    # Decoding
    all_outputs = []
    inputs = decoder_inputs
    outs_dists = []
    for curr_idx in range(H_WINDOW):
```

```
        # Run the decoder on one timestep
        decoder_pred, state_h, state_c = lstm_layer_dec(inputs,
            initial_state=states)
        outputs_delta = decoder_dense_mot(decoder_pred)
        outputs_delta_dir = decoder_dense_dir(decoder_pred)
        outputs_pos = To_Position([inputs, outputs_delta,
            outputs_delta_dir])
        # Store the current prediction (we will concantenate all
            predictions later)
        all_outputs.append(outputs_pos)
        # Reinject the outputs as inputs for the next loop iteration as
            well as update the states
        inputs = outputs_pos
        states = [state_h, state_c]

    # Concatenate all predictions
    decoder_outputs = Lambda(lambda x: K.concatenate(x, axis=1))(
        all_outputs)
    marginal_h = marginal_tril_dist('h')
    marginal_c = marginal_tril_dist('c')

    # VIB LOSS
    def vib_loss(y_true, y_pred):
        class_loss = metric_orth_dist(y_true, y_pred)
        info_loss_h = state_h_distribution.log_prob(state_h_sample)
        info_loss_c = state_c_distribution.log_prob(state_c_sample)
        marginal_loss_h = marginal_h.log_prob(tf.cast(state_h_sample, tf.
            float32))
        marginal_loss_c = marginal_c.log_prob(tf.cast(state_c_sample, tf.
            float32))
        return class_loss + BETA_h * (info_loss_h - marginal_loss_h) +
            BETA_c * (info_loss_c - marginal_loss_c)

# Use custom model class so that gradients can be propagated back to
    marginal distribution

class MarginalGradModel(Model):
    def train_step(self, data):
        # Unpack the data.
        x, y = data
        x1, x2 = x[0], x[1]
        with tf.GradientTape() as tape:
            y_pred = self([x1,x2], training=True) # Forward pass
            # Compute the loss value
            loss = self.compiled_loss(y, y_pred, regularization_losses=self
                .losses)

        # Compute gradients
        trainable_vars = self.trainable_variables
```

```
sess = K.get_session()
# Get marginal parameters in computational graph
for v in tf.compat.v1.trainable_variables():
    if "mu" in v.name:
        if "c" in v.name:
            mus_c = v
        else:
            mus_h = v

    elif "rho" in v.name:
        if "c" in v.name:
            rhos_c = v
        else:
            rhos_h = v

    elif "mix_logits" in v.name:
        if "c" in v.name:
            mix_logits_c = v
        else:
            mix_logits_h = v

gradients = tape.gradient(loss, trainable_vars)

# calculate gradients w.r.t marginal parameters
mus_h_gradients = tape.gradient(loss, mus_h)
rhos_h_gradients = tape.gradient(loss, rhos_h)
mix_logits_h_gradients = tape.gradient(loss, mix_logits_h)

mus_c_gradients = tape.gradient(loss, mus_c)
rhos_c_gradients = tape.gradient(loss, rhos_c)
mix_logits_c_gradients = tape.gradient(loss, mix_logits_c)


# Update parameters
self.optimizer.apply_gradients(zip(gradients, trainable_vars))
self.optimizer.apply_gradients(zip(mus_h_gradients, mus_h))
self.optimizer.apply_gradients(zip(rhos_h_gradients, rhos_h))
self.optimizer.apply_gradients(zip(mix_logits_h_gradients,
    mix_logits_h))
self.optimizer.apply_gradients(zip(mus_c_gradients, mus_c))
self.optimizer.apply_gradients(zip(rhos_c_gradients, rhos_c))
self.optimizer.apply_gradients(zip(mix_logits_c_gradients,
    mix_logits_c))
# Update metrics (includes the metric that tracks the loss)
self.compiled_metrics.update_state(y, y_pred)
# Return a dict mapping metric names to current value
return {m.name: m.result() for m in self.metrics}
```

C.1.3 Original Marginal Distribution Code

This subsection contains code supplied during email correspondence with Ian Fischer [55] that implements the VIB parametric marginal distribution.

```
def marginal_tril_dist():
  n = 200
  d = 3
  min_variance = 1e-5
  # Compute the number of parameters needed for the lower triangular
      covariance matrix
  tril_components = (d * (d + 1)) // 2

  # Parameterize the categorical distribution for the mixture
  mix_logits = tf.get_variable('mix_logits', [n])
  mix_dist = tfd.Categorical(logits=mix_logits)

  # Parameterize the means of Gaussian distribution
  mus = tf.get_variable('mus', [n, d], initializer=tf.initializers.
      random_normal())

  # Parameterize the lower-triangular covariance matrix for the Gaussian
      distribution
  rhos = tf.get_variable('rhos', [n, tril_components], initializer=tf.
      initializers.random_normal(-(1.0 / n), (1.0 / n)))
  # The diagonal of the lower-triangular matrix has to be positive, so
      transform the diagonal with a softplus and then translate it by
      min_variance.
  scale_tril = tfb.FillScaleTriL(diag_bijector=tfb.Chain([tfb.Softplus(),
      tfb.Shift(min_variance)]))(rhos)

  # Make the fully covariant Gaussian distribution
  comp_dist = tfd.MultivariateNormalTriL(loc=mus, scale_tril=scale_tril)

  # Make the mixture distribution
  dist = tfd.MixtureSameFamily(
      components_distribution=comp_dist,
      mixture_distribution=mix_dist,
  )
  return dist
```

C.1.4 Adapted Marginal Distribution Code

The major adaption of marginal distribution code was in transferring it from Tensorflow version 1 to 2, and in making it compatible for training with the position-only-baseline model:

```
# Define learnable marginal distribution for UQ
def marginal_tril_dist(h_or_c='h'):
    n = 100
```

```
    d = 512
    min_variance = 1e-5
    # Compute the number of parameters needed for the lower triangular
        covariance matrix
    tril_components = (d * (d + 1)) // 2

    # Parameterize the categorical distribution for the mixture
    init_vals = np.random.rand(n)
    init_probs = [float(i)/sum(init_vals) for i in init_vals]
    mix_logits = tf.cast(np.log(init_probs), dtype=tf.float32)
    mix_logits = trainable_dist_layer(mix_logits, name='mix_logits_%s'%(
        h_or_c))

    mix_dist = tfp.distributions.Categorical(logits=mix_logits.return_vars
        ())

    # Parameterize the means of Gaussian distribution
    mu_init = tf.initializers.RandomNormal()
    mus = tf.cast(mu_init(shape=[n,d]), dtype=tf.float32)
    mus = trainable_dist_layer(mus, name = 'mus_%s'%(h_or_c))

    # Parameterize the lower-triangular covariance matrix for the Gaussian
         distribution
    lower_tril_init = tf.initializers.RandomNormal(-(1.0 / n), (1.0 / n))
    rhos = tf.cast(lower_tril_init(shape=[n,tril_components]), dtype=tf.
        float32)
    rhos = trainable_dist_layer(rhos, name='rhos_%s'%(h_or_c))

    # The diagonal of the lower-triangular matrix has to be positive, so
        transform the diagonal with a softplus and then translate it by
        min_variance.
    scale_tril = tfp.bijectors.FillScaleTriL(diag_bijector=tfp.bijectors.
        Chain([tfp.bijectors.Softplus(), tfp.bijectors.Shift(min_variance)
        ]))(rhos.return_vars())

    # Make the fully covariant Gaussian distribution
    comp_dist = tfp.distributions.MultivariateNormalTriL(loc=mus.
        return_vars(), scale_tril=scale_tril)

    # Make the mixture distribution
    dist = tfp.distributions.MixtureSameFamily(
        components_distribution=comp_dist,
        mixture_distribution=mix_dist,
        name='marginal_dist_%s'%(h_or_c),
        )
    return dist

# Wrapper layer class for use with marginal distribution parameters, so
    they are trainable.
```

```
class trainable_dist_layer(tf.keras.layers.Layer):
    def __init__(self, init_value, name):
        super(trainable_dist_layer, self).__init__()
        self.trainable_vars = tf.Variable(initial_value=init_value, name =
            name)
    def return_vars(self):
        return self.trainable_vars
    def call(self):
        return self.trainable_vars
```

Appendix D

Experiments & Results

D.1 Hard Synthetic Trajectories Dataset Examples

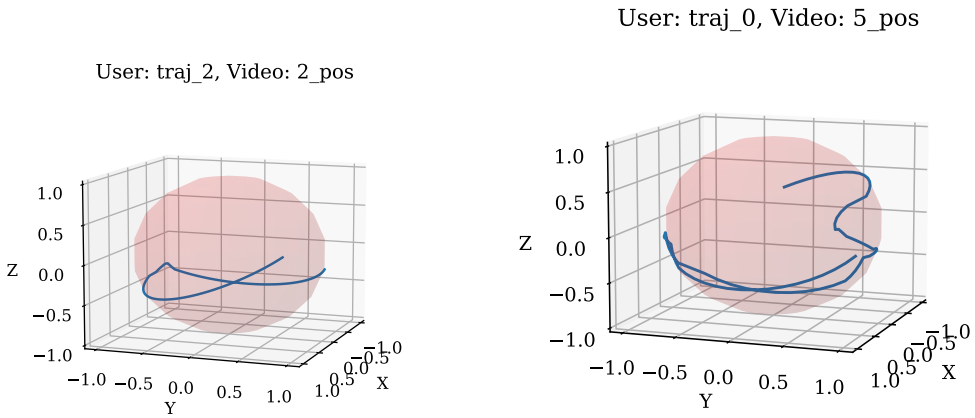

Figure D.1: Trajectories with (left) 2 selected positions and (right) 5 selected positions

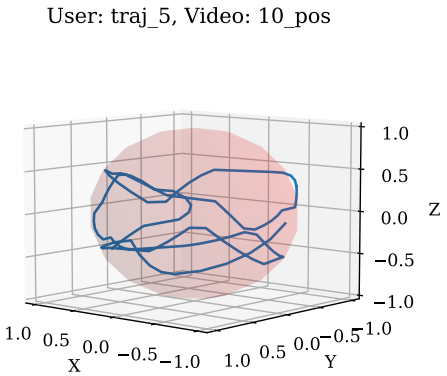

Figure D.2: Trajectory with 10 selected positions

Figure D.3: Example traces (represented in the unit sphere) that follow the (x, y, z) coordinates of synthetic trajectories that included 2, 5 and 10 selected positions. Each of these trajectories was extracted from a separate synthetic test set.

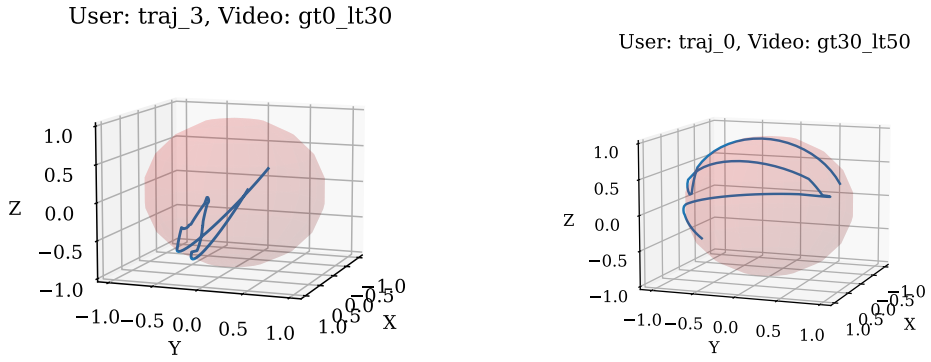

Figure D.4: Angular constraints (left) $0^\circ < \theta < 30^\circ$ and (right) $30^\circ < \theta < 50^\circ$

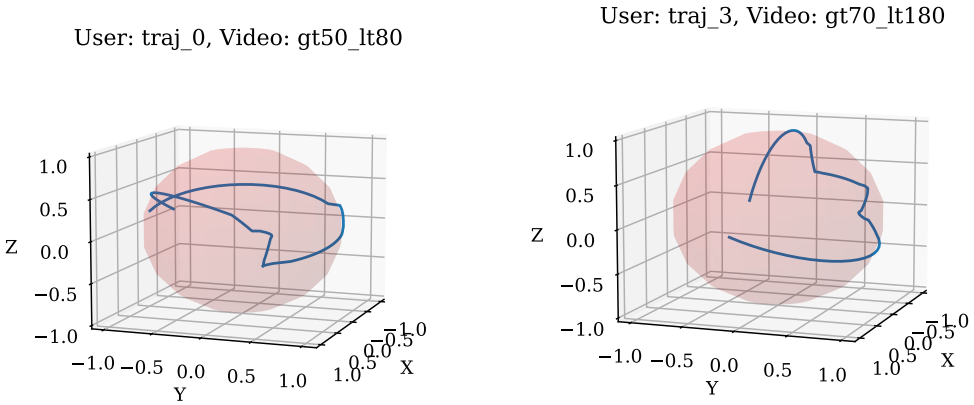

Figure D.5: Angular constraints (left) $50^\circ < \theta < 80^\circ$ and (right) $70^\circ < \theta < 180^\circ$

Figure D.6: Example traces (represented in the unit sphere) that follow the (x, y, z) coordinates of synthetic trajectories where selected positions are constrained to have angles between trajectory segments (see Section **??** for more detail) in ranges $0 - 30^\circ, 30 - 50^\circ, 50 - 80^\circ$ and $70 - 180^\circ$. Each of these trajectories was extracted from a separate synthetic test set.

D.2 Simple Synthetic Trajectories Dataset Examples

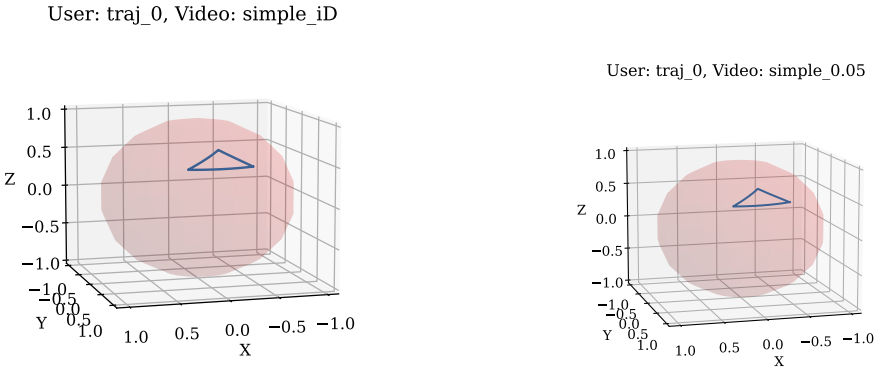

Figure D.7: Trajectories with (left) $\sigma_{pos} = 0.01$ and (right) $\sigma_{pos} = 0.05$

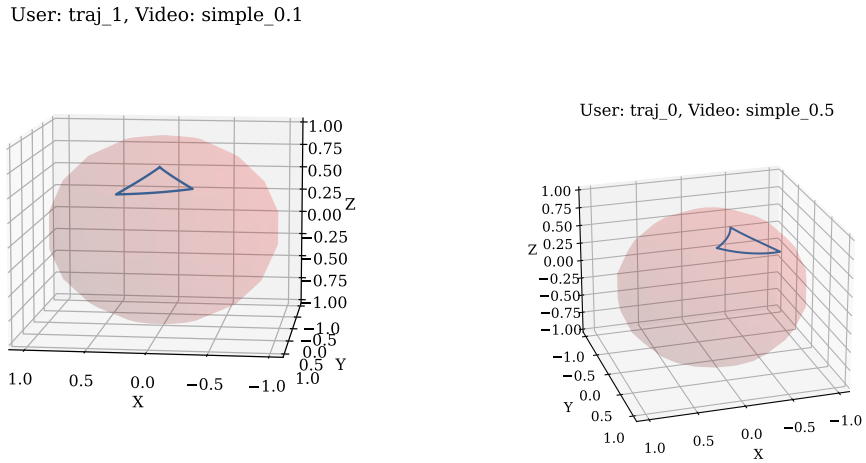

Figure D.8: Trajectories with (left) $\sigma_{pos} = 0.1$ and (right) $\sigma_{pos} = 0.5$

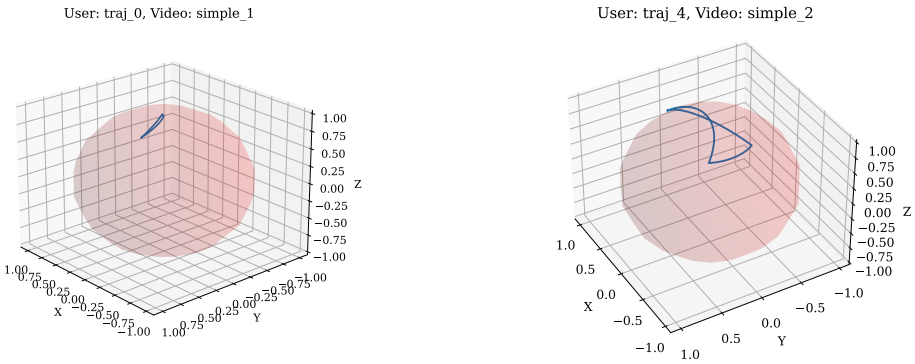

Figure D.9: Trajectories with (left) $\sigma_{pos} = 1$ and (right) $\sigma_{pos} = 2$

Figure D.10: Example traces (represented in the unit sphere) that follow the (x, y, z) coordinates of synthetic trajectories (extracted from the SSTD - see Section 4.1.2) where positions are generated from distributions centered on three standard points, with varying standard deviations σ_{pos}. Each of these trajectories was extracted from a separate SSTD test set.

D.3 Visualisations of iD and OoD activation distributions

D.3.1 t-SNE Visualisation Method

The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm [59] is a method for visualizing high-dimensional data in two or three dimensional space. It functions by constructing a probability density function over pairs of data points in high dimensional space, such that the distribution of data points P is generated from joint probabilities $p_{i,j}$ - with similar points being assigned high probabilities. t-SNE then generates a low-dimensional embedding, with a similar distribution Q over pairs of data points $q_{i,j}$, and enforces this lower dimensional embedding to accurately represent high dimensional data by minimizing the Kullback-Leibler (KL) divergence between P and Q. The KL divergence as a cost function here heavily penalizes using low dimensional pairs of points $q_{i,j}$ that are similar to represent dissimilar high dimensional pairs $p_{i,j}$. A t-SNE mapping from high to low dimensional data is trained using gradient descent, with an individual t-SNE model trained for each dataset being visualised. Since the t-SNE method is based on pairwise comparisons of data points, it is capable of capturing local structure in high dimensional space, and representing it in lower dimensional embeddings. This often presents as clusters of data points in the lower dimensional space.

The entropy of distributions P and Q is controlled by a 'perplexity' hyperparameter. The authors of the original t-SNE paper [59] indicate perplexity values from 5-50 are suitable for visualizing most datasets. However, it is well known that perplexity values, dependent on the characteristics of the high dimensional data at hand, can artificially generate structure in the low dimensional embedding where there is in fact none. As a result, it is important to generate multiple visualisations using a t-SNE model with different perplexity values, and verify that the structure of the low dimensional embedding does not change significantly with varying perplexity settings.

For our visualisations, we fit t-SNE models to each set of iD and OoD $\mathbf{z_h}$ and $\mathbf{z_c}$ activations (treating iD and OoD activations as belonging to a single high-dimensional dataset), for each OoD test case from our David-MMSys18, HSTD and SSTD experiments (see Section 4.3 for more detail). However, we also first reduce activations to be 50 dimensional using principal components analysis, as recommended by the authors [62], before fitting t-SNE models to 50 dimensional activation sets. We train t-SNE models for 1000 iterations and visualise the resulting two dimensional embeddings, with repeated training and visualisation for different perplexity values in the range 5-50. We observe that visualisations do not vary in their fundamental structure with perplexity - indicating that t-SNE visualisations are not introducing artificial structure amongst activations. For all visualisations used in this thesis we use a perplexity value of 20.

D.3.2 SSTD t-SNE Visualisations

This section includes visualisations of sampled $\mathbf{z_h}$ and $\mathbf{z_c}$ vectors (as defined in Section 3.2.1) extracted from the position-only-VIB model, generated from iD and OoD

input trajectories as defined under SSTD OoD test cases (see Section 4.3). Visualisations were generated using the t-distributed Stochastic Neighbor Embedding algorithm, under the implementation described in Appendix D.3.1.

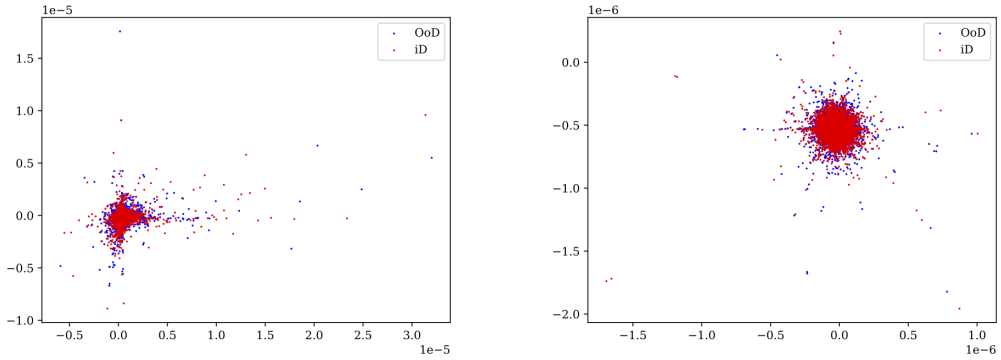

Figure D.11: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the SSTD OoD test case $\sigma_{pos} = 0.05$

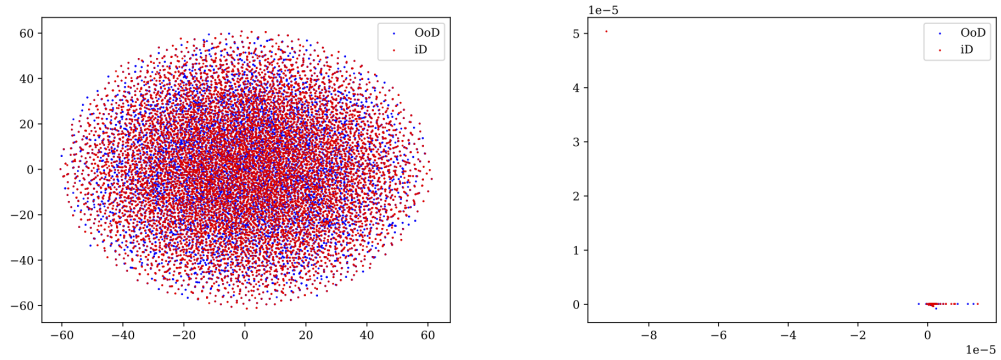

Figure D.12: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the SSTD OoD test case $\sigma_{pos} = 0.1$

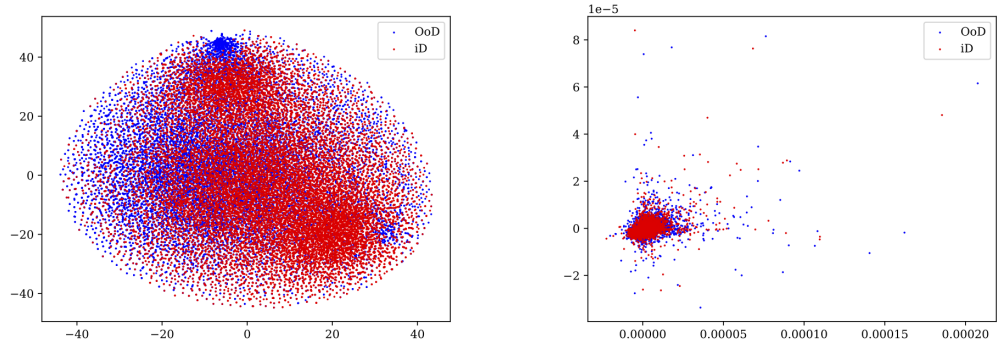

Figure D.13: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the SSTD OoD test case $\sigma_{pos} = 0.5$

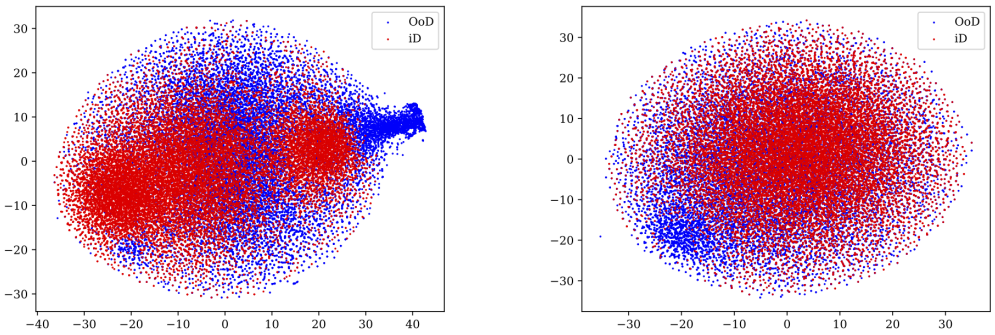

Figure D.14: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the SSTD OoD test case $\sigma_{pos} = 1$

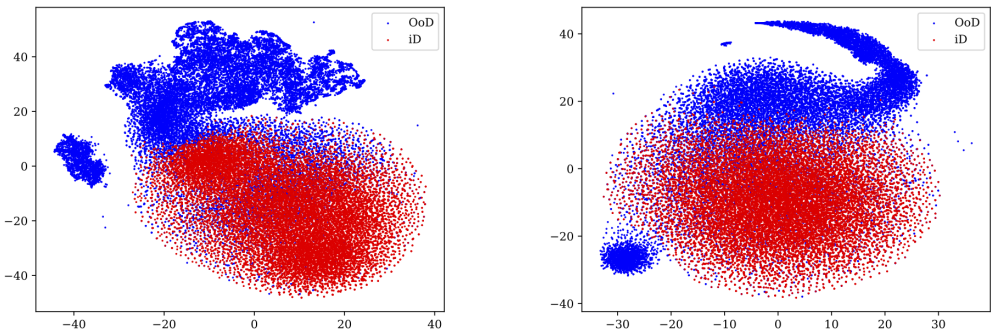

Figure D.15: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the SSTD OoD test case $\sigma_{pos} = 2$

D.3.3 SSTD per-instance rate distributions

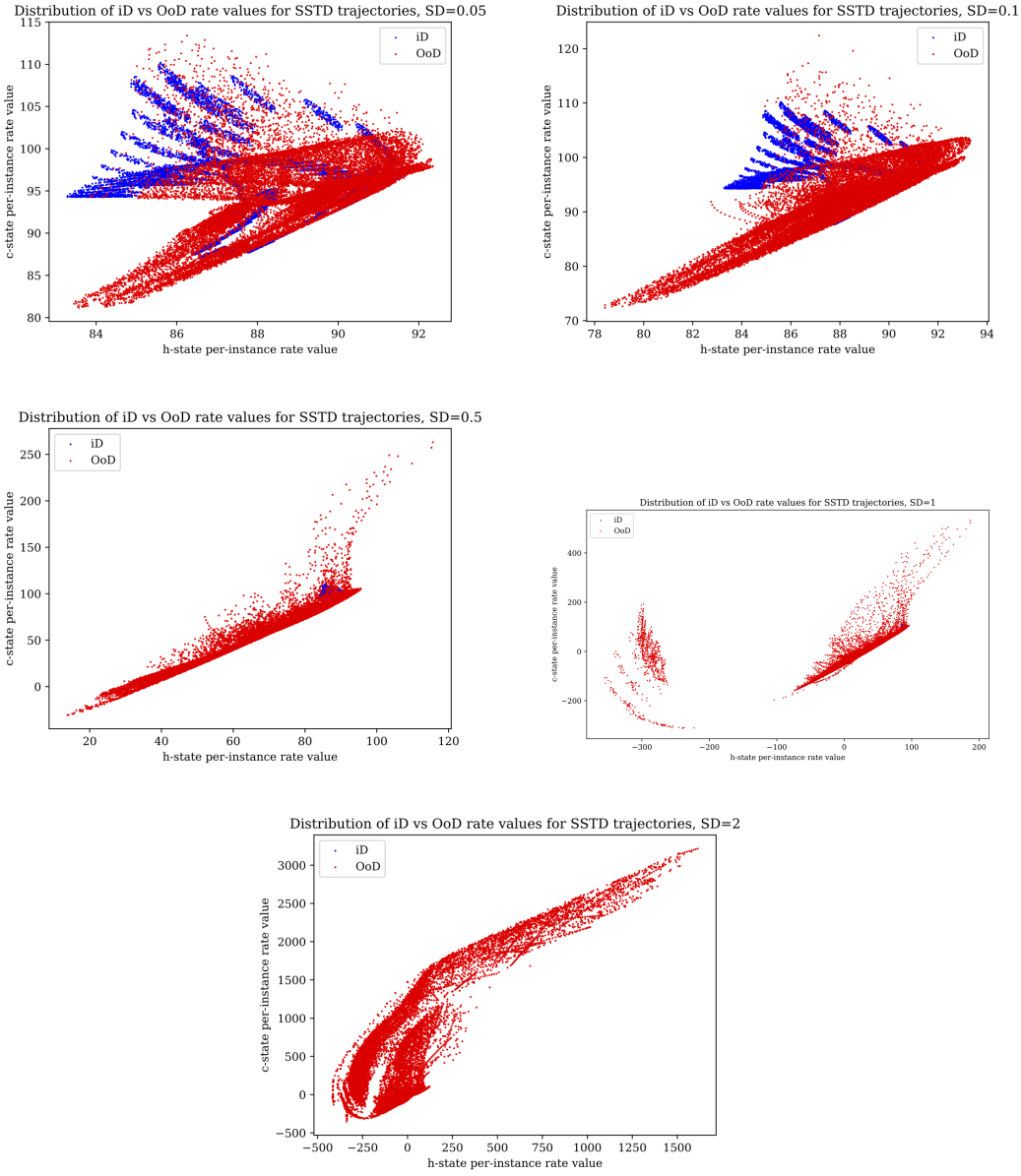

Figure D.16: Figures showing scatter plots of generated iD and OoD per-instance rate values in response to SSTD OoD test cases, with rate values computed from the $\mathbf{h}$ encoding distribution ($e^h_\theta(\mathbf{x})$) and marginal (m^h_ϕ) plotted on the x axis and values computed from the $\mathbf{c}$ encoding distribution ($e^c_\theta(\mathbf{x})$) and marginal (m^c_ϕ) on the y axis. Scatter plots for SSTD test cases with $\sigma_{pos} \in [0.05, 0.1, 0.5, 1, 2]$ are ordered with increasing σ_{pos} from the top to bottom of the figure, with two plots per row.

D.3.4 SSTD per-instance rate distributions - demonstrating the manifold

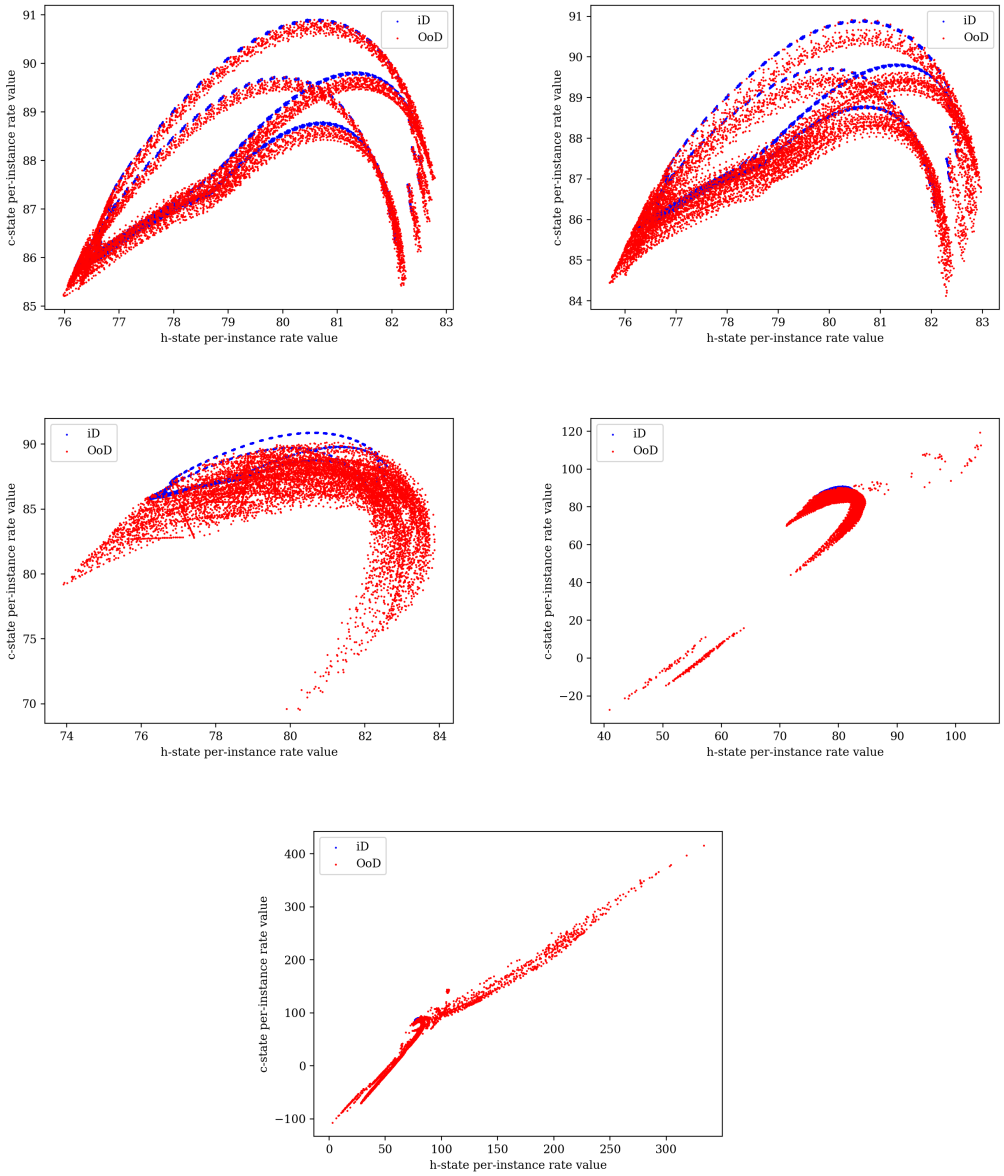

Figure D.17: Figures showing scatter plots of generated iD and OoD per-instance rate values in response to SSTD OoD test cases, with rate values computed from the $\mathbf{h}$ encoding distribution ($e^h_\theta(\mathbf{x})$) and marginal (m^h_ϕ) plotted on the x axis and values computed from the $\mathbf{c}$ encoding distribution ($e^c_\theta(\mathbf{x})$) and marginal (m^c_ϕ) on the y axis. These plots are generated using the position-only-VIB model with information bottleneck coefficient $\beta = 1 \times 10^{-3}$, and display the 'iD manifold' described in Section 5.1. Scatter plots for SSTD test cases with $\sigma_{pos} \in [0.05, 0.1, 0.5, 1, 2]$ are ordered with increasing σ_{pos} from the top to bottom of the figure, with two plots per row.

D.3.5 HSTD t-SNE Visualisations

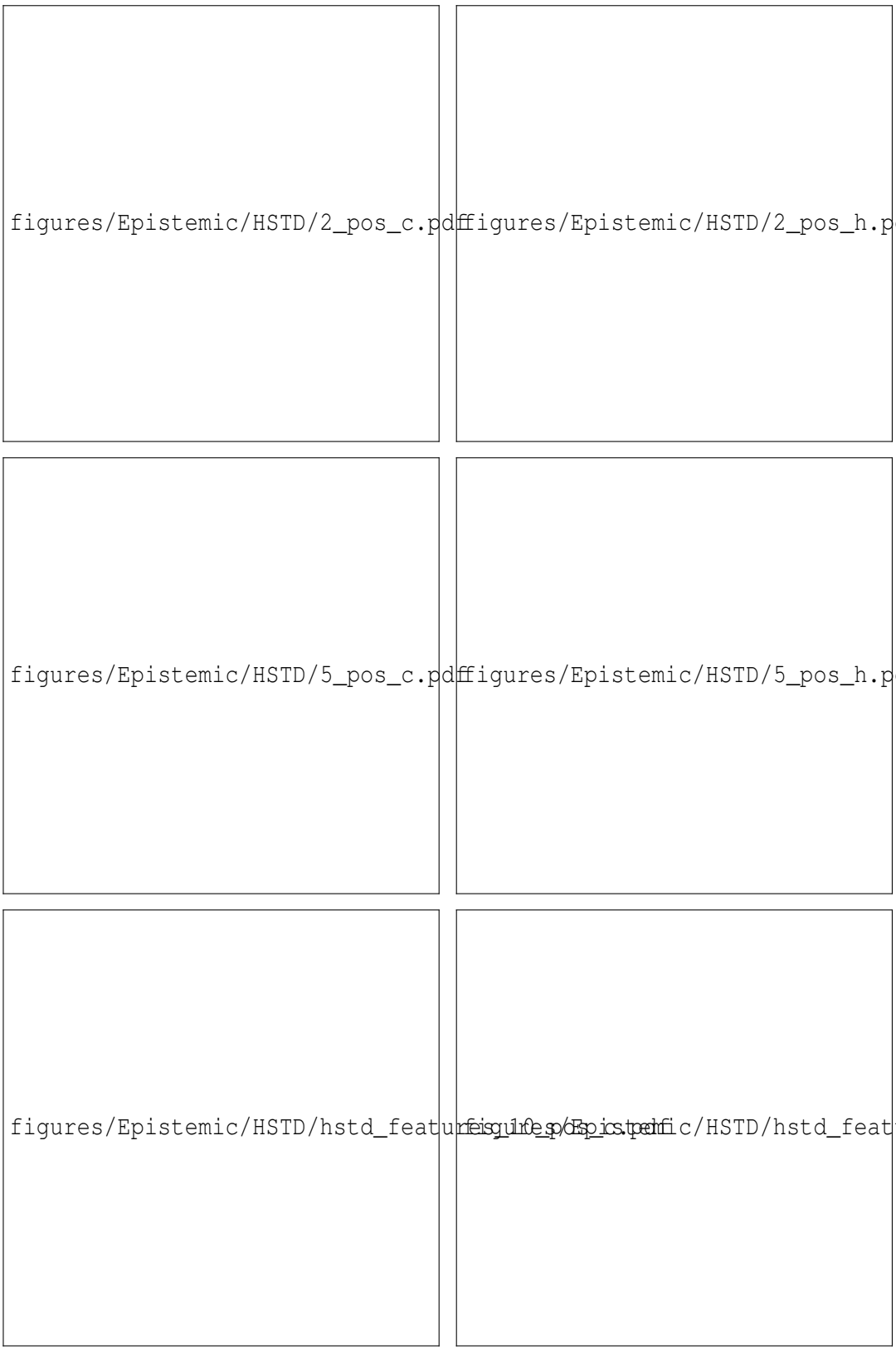

Figure D.18: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for varying HSTD 'Speed' constraint OoD test cases, with speed values $\in [2, 5, 10]$ displayed from the top to bottom of the figure (see Sections 4.3 and 4.1.1 for definitions of OoD test cases and speed constraints respectively)

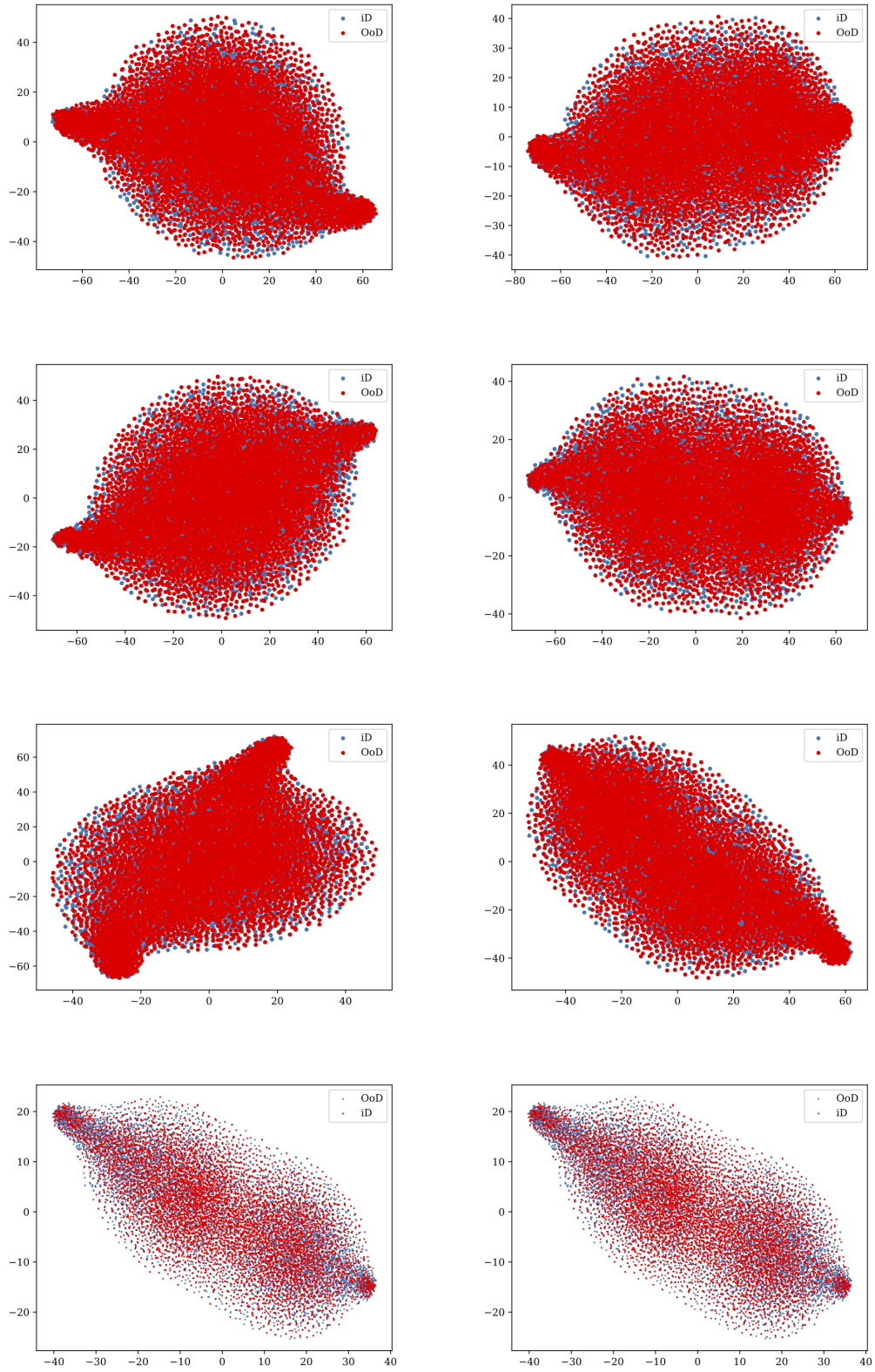

Figure D.19: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for varying HSTD 'angular' constraint OoD test cases, with angles constrained to be in ranges $[0-30°, 30-50°, 50-80°, 70-180°]$ displayed from the top to bottom of the figure (see Sections 4.3 and 4.1.1 for definitions of OoD test cases and angular constraints respectively)

D.3.6 HSTD per-instance rate distributions

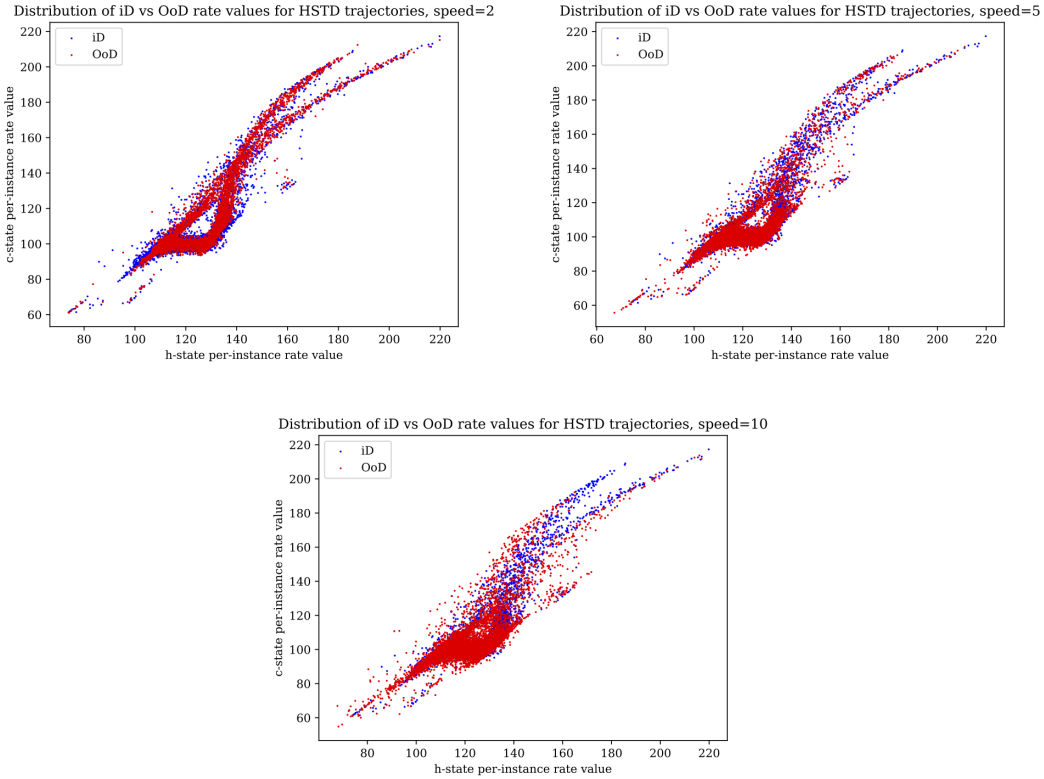

Figure D.20: Figures showing scatter plots of generated iD and OoD per-instance rate values in response to varying HSTD 'speed' constraint OoD test cases, with rate values computed from the $\mathbf{h}$ encoding distribution ($e^h_\theta(\mathbf{x})$) and marginal (m^h_ϕ) plotted on the x axis and values computed from the $\mathbf{c}$ encoding distribution ($e^c_\theta(\mathbf{x})$) and marginal (m^c_ϕ) on the y axis. Scatter plots for HSTD 'speed' constraints $\in [2, 5, 10]$ are displayed clockwise in this order.

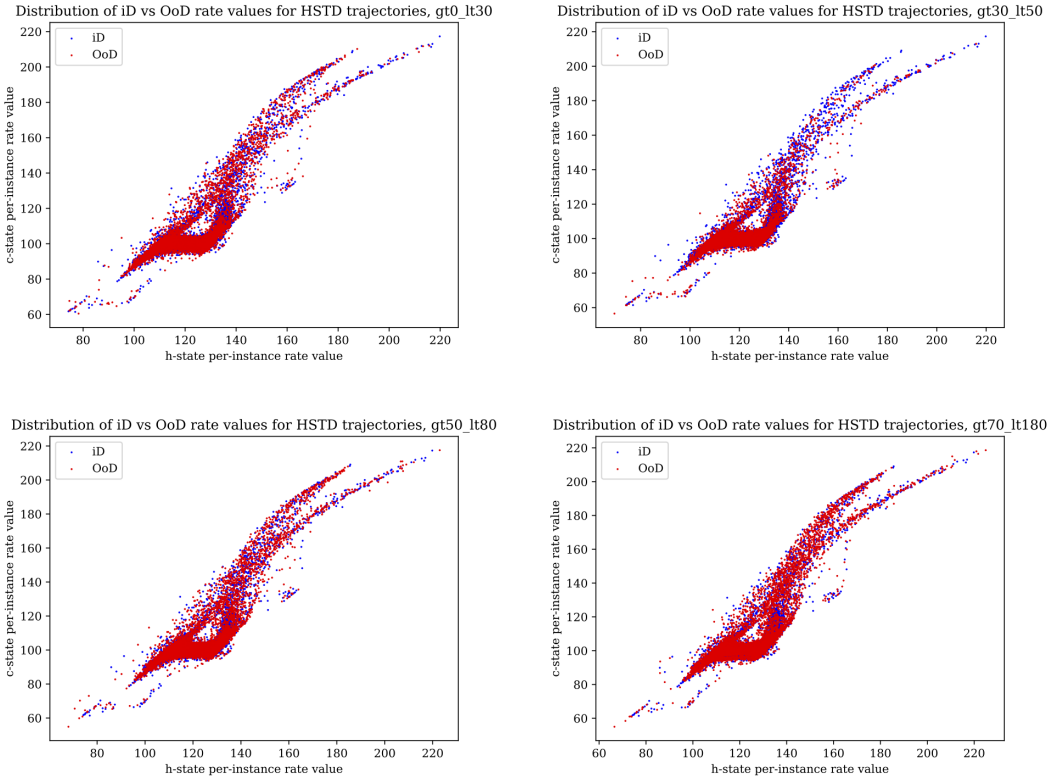

Figure D.21: Figures showing scatter plots of generated iD and OoD per-instance rate values in response to varying HSTD angular constraint OoD test cases, with rate values computed from the $\mathbf{h}$ encoding distribution ($e^h_\theta(\mathbf{x})$) and marginal (m^h_ϕ) plotted on the x axis and values computed from the $\mathbf{c}$ encoding distribution ($e^c_\theta(\mathbf{x})$) and marginal (m^c_ϕ) on the y axis. Scatter plots for HSTD angular constraint ranges $[0-30°, 30-50°, 50-80°, 70-180°]$ are displayed in this order from the top to bottom of the figure, with two plots per row.

D.3.7 David-MMSys18 t-SNE Visualisations

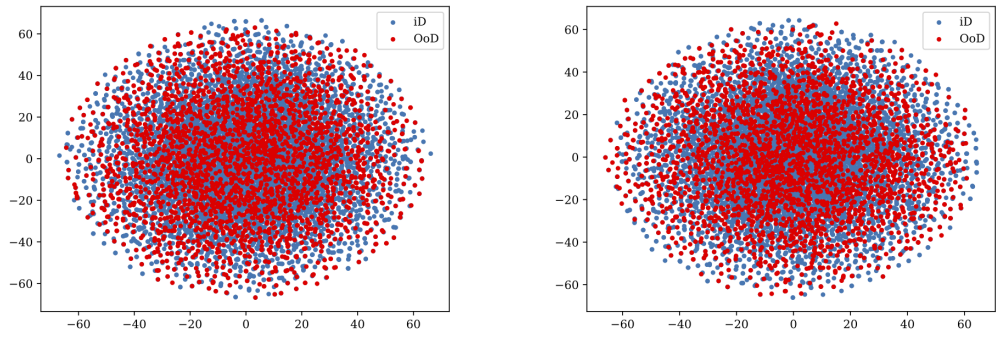

Figure D.22: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the David-MMSys18 'Videos' OoD test case (see Section 4.3 for definitions of OoD test cases)

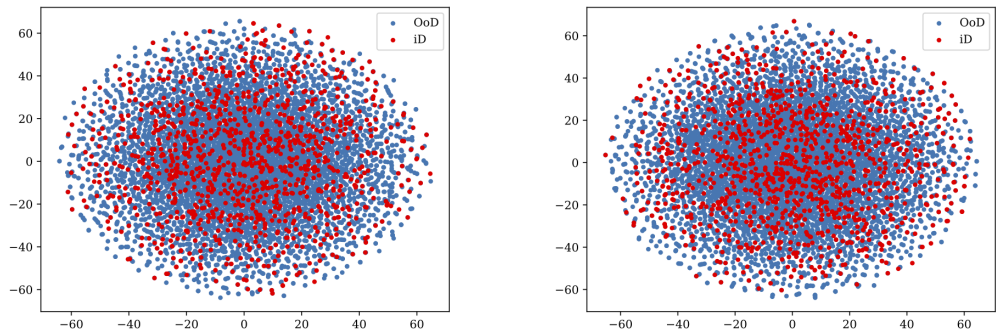

Figure D.23: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the David-MMSys18 'Users' OoD test case

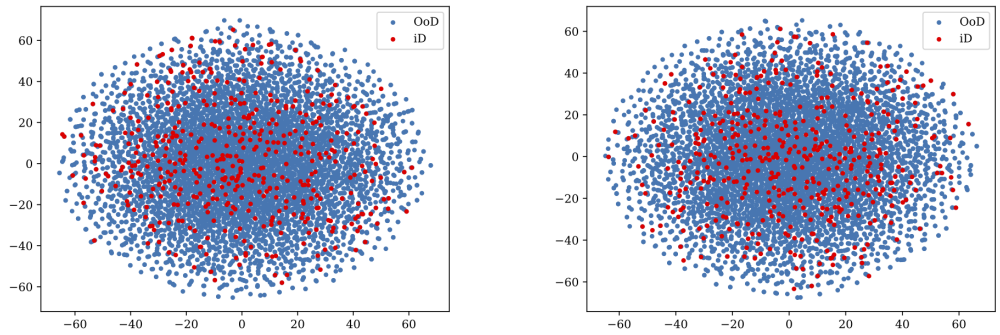

Figure D.24: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the David-MMSys18 'Videos & Users' OoD test case

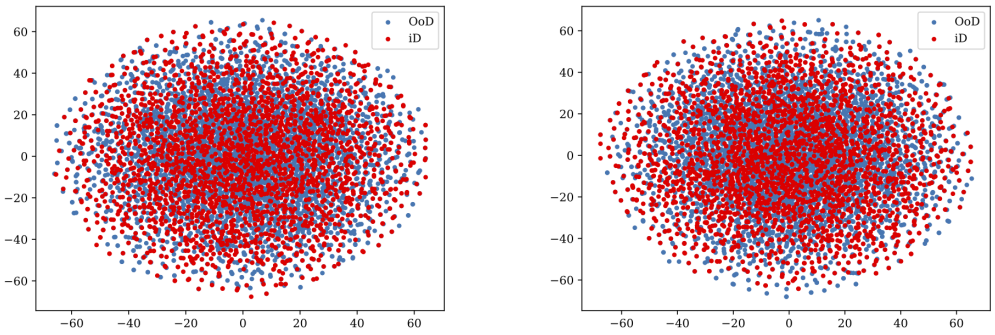

Figure D.25: $\mathbf{z_c}$ (left) and $\mathbf{z_h}$ (right) samples for the David-MMSys18 'True OoD' OoD test case

D.3.8 David-MMSys18 per instance rate distributions

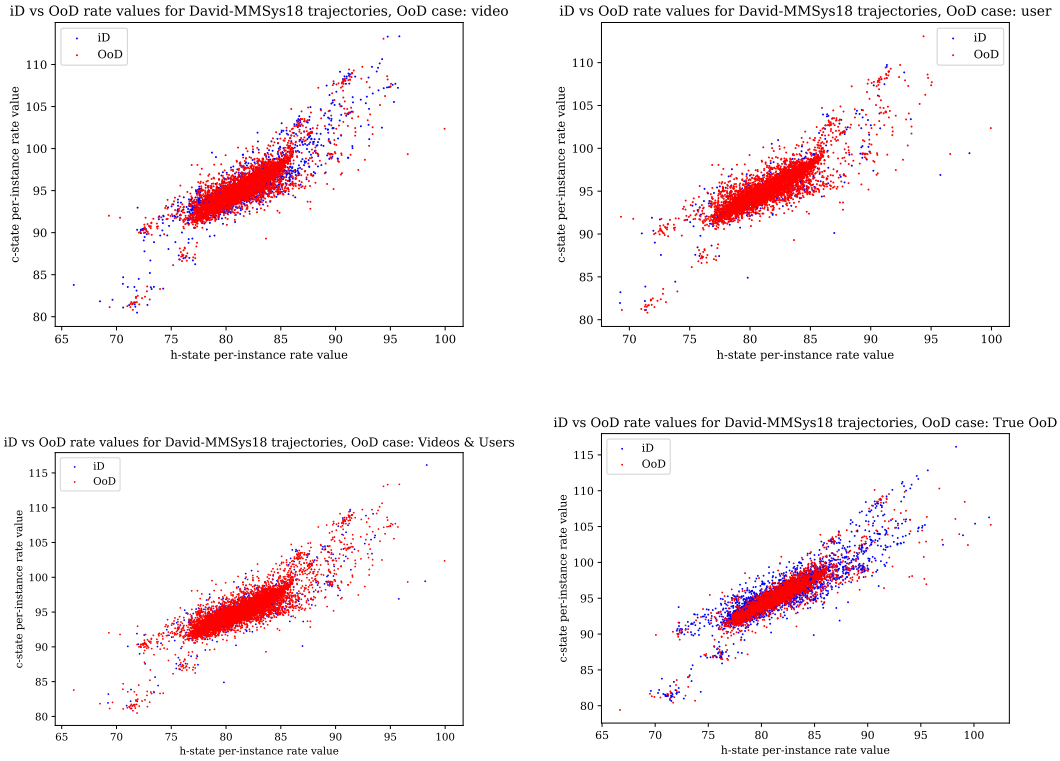

Figure D.26: Figures showing scatter plots of generated iD and OoD per-instance rate values in response to David-MMSys18 OoD test cases, with rate values computed from the $\mathbf{h}$ encoding distribution ($e^h_\theta(\mathbf{x})$) and marginal (m^h_ϕ) plotted on the x axis and values computed from the $\mathbf{c}$ encoding distribution ($e^c_\theta(\mathbf{x})$) and marginal (m^c_ϕ) on the y axis. Scatter plots for David-MMSys18 test cases 'Videos', 'Users', 'Videos & Users', 'True OoD' are displayed in this order from the top to bottom of the figure, with two plots per row.

D.4 Aleatoric Uncertainty Results

Video		**Num. uncertainty estimates over threshold**				
	Threshold	0.013	0.014	0.015	0.018	0.02
1_PortoRiverside		*375*	*328*	*225*	*225*	*225*
3_PlanEnergyBioLab		*204*	*204*	*202*	*201*	*182*
5_Waterpark		*61*	*61*	*61*	*61*	*60*
14_Warship		*230*	*228*	*223*	*204*	*173*
16_Turtle		*106*	*105*	*91*	*91*	*91*

Table D.1: Counts of uncertainty estimates made above thresholds $[0.013, 0.014, 0.015, 0.018, 0.02]$ used to calculate AUROC results in section 4.4.

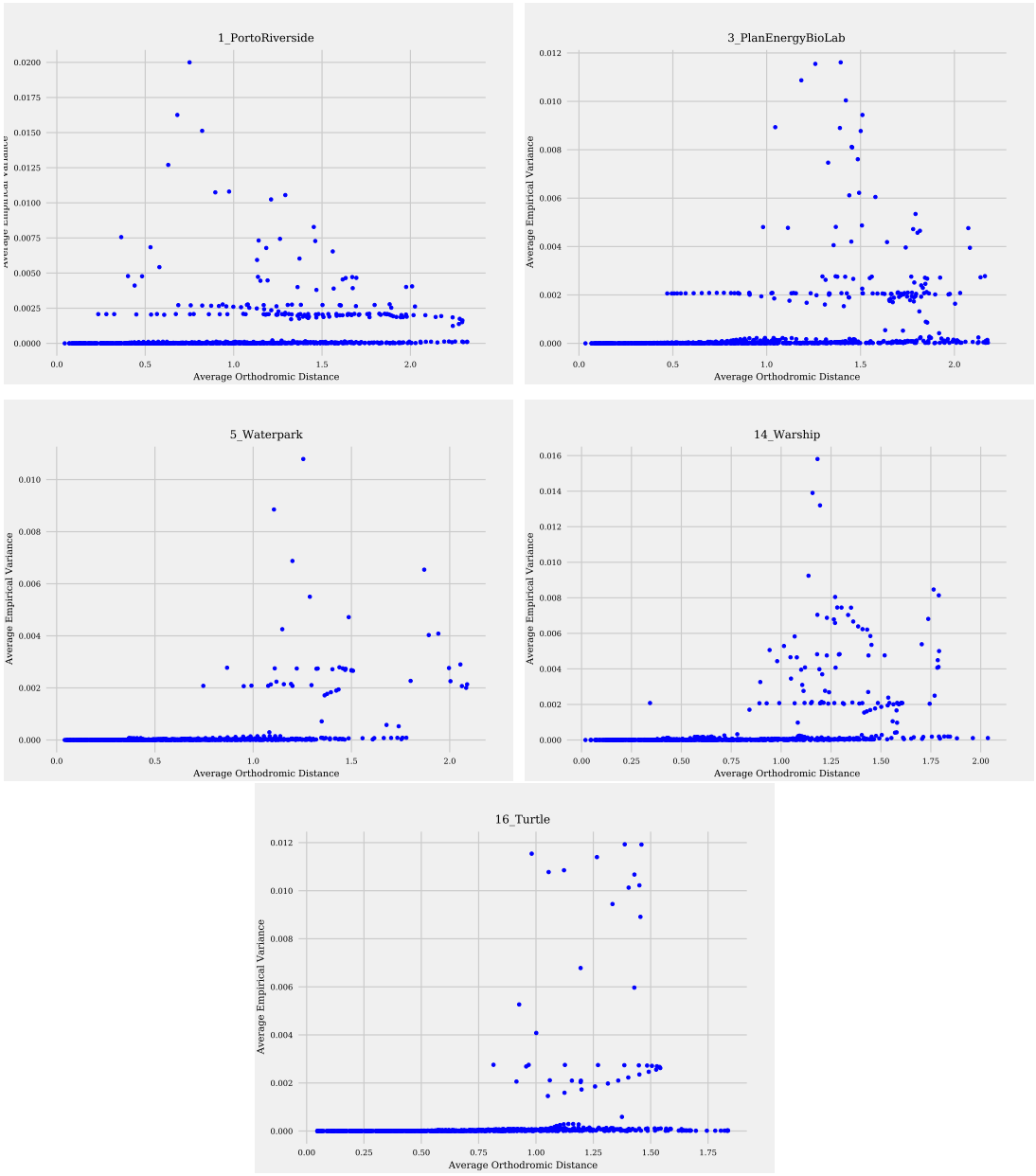

Figure D.27: Scatter plots for aleatoric uncertainty estimates vs Orthodromic distance on David-MMSys18 videos