SUPPLEMENTARY MATERIAL

For Multi-Vehicle Trajectories Generation for Vehicle-to-Vehicle Encounters

Wenhao Ding, Wenshuo Wang and Ding Zhao

A Network Architecture

A.1 Multi-vehicle Trajectory Generator

We display the specific structure of our MTG in the following tables. Tab 1 displays the encoder module. The two sequences are processed at the same time and the final output is an embedding vector with $2 \times z_{dim}$. In our experiments, we set the dimension $z_dim = 10$. Since we use bi-directional GRU and the size of hidden state is 256, the size of output state is 256×2. After obtaining the embedding vector, we divide it into two parts. The first of it represents the mean value of Gaussian distribution and the second part represents the standard deviation.

Table 1: Encoder of MTG

Description		Output dimension
$Sequence_1(x_1, y_1)$	$Sequence_2(x_1, y_1)$	$50 \times 2, 50 \times 2$
Concatenated sequence (x_1, y_1, x_2, y_2)		50×4
Input layer (MLP)		50×64
ReLU		-
Encoder (GRU)		1×512
Encoder (MLP)		512×20
z_{mean}	z_{std}	$1 \times 10, 1 \times 10$

Table 2: Decoder of MTG

Description		Output dimension
z	z	$1 \times 10, 1 \times 10$
Hidden layer (MLP)	Hidden layer (MLP)	$1 \times 256, 1 \times 256$
ReLU		-
Hidden2StartPoint Layer (MLP)	Hidden2StartPoint Layer (MLP)	$1 \times 32, 1 \times 32$
ReLU		-
$Startpoint_1$	$Startpoint_2$	$1 \times 2, 1 \times 2$
Input layer (MLP)	Input layer (MLP)	$1 \times 32, 1 \times 32$
ReLU		-
Concatenate	Concatenate	$1 \times 64, 1 \times 64$
$Decoder_1(GRU)$	$Decoder_2(GRU)$	$1 \times 256, 1 \times 256$
$Decoder_1(MLP)$	$Decoder_2(MLP)$	$1 \times 2, 1 \times 2$
$Ponit_1$	$Ponit_2$	$1 \times 2, 1 \times 2$
Collector	Collector	$50 \times 2, 50 \times 2$

As an important part, the reparameterization trick is then operated on z_{mean} and z_{std}, which guarantees the possibility of back-propagation. The sampled latent code z can be obtained after this process. The structure of our decoder is shown in Tab 2. During the generating process, we first utilize a Multiple Layers Perception (MLP) to map the latent code z into the initial hidden state of GRU. The input of GRU consists of two parts and each of them has the dimension of 32. The first part comes from the hidden state (through the *Hidden2Startpoint (H2S) Layer*), and the second part comes from the output of last state. Since the first state has no previous state, an arbitrate start point is used to determine where to start. In Tab 2, the size of input state is set to 64, and it comes

from the concatenation of start point (1×32) and Hidden2StartPoint (1×32). The *Collector* means the sequence is generated through a loop and one point is generated in each iteration.

A.2 Variational Autoencoder Baseline

In this subsection, we will describe our VAE baseline. The structure of encoder and decoder are displayed in Tab 3. In the encoder, we utilize a uni-directional GRU to process two sequences simultaneously. In the decoder, two sequences are also processed at the same time, which means two points are generated in each iteration.

Table 3: Architecture of original VAE

Description		Output dimension
$Sequence_1(x_1, y_1)$	$Sequence_2(x_1, y_1)$	$50 \times 2, 50 \times 2$
Concatenated sequence (x_1, y_1, x_2, y_2)		50×4
Input layer (MLP)		50×64
ReLU		-
Encoder (GRU)		1×256
Output layer (MLP)		256×20
z_{mean}	z_{std}	$1 \times 10, 1 \times 10$
z (Reparameterization)		1×10
Hidden layer (MLP)		1×256
ReLU		-
Start point (x_1, y_1, x_2, y_2)		1×4
Input layer (MLP)		1×64
ReLU		-
Decoder (GRU)		1×256
Output layer (MLP)		1×4
Two points		1×4
Collector		50×4
$Sequence_1$	$Sequence_2$	$50 \times 2, 50 \times 2$

A.3 InfoGAN Baseline

The InfoGAN baseline is almost the same as the MTG. The generator of InfoGAN has the same structure as the decoder of MTG. The discriminator of InfoGAN utilize a bi-directional GRU and output a 1-dimensional number. The *Q module* of InfoGAN is implemented with three MLPs.

B Training Details

We display some hyper-parameters and training methods used in the training stage in this section. We conduct our experiment with *Pytorch* and the hardware platform contains $NVIDIA^{®}$ $GTX1070$ ($8G$) and $Intel^{®}$ $i7\text{-}7700K$. Values of all hyper-parameters are shown in Tab 4. We use Adaptive Moment Estimation (Adam) algorithm [1] as our optimizer and the hyper-parameter *beta* of β-VAE is implemented with annealing method. The formula of the annealing process is the same as [2]:

$$\lambda_{step} = 1 - (1 - \lambda min) \times R_{step} \tag{1}$$

$$\beta = \beta_{base} \times \mu_{step} \tag{2}$$

$$\mathcal{L}_{VAE} = \mathcal{F}(S_1, \bar{S_1}) + \mathcal{F}(S_2, \bar{S_2}) + \beta \times D_{KL}(q_\phi || p(z)) \tag{3}$$

The loss curve of MTG is displayed in Fig 1. We show the curve of K-L divergence and reconstruction error separately for better understanding the training stage. The reconstruction error always decreases during the training while K-L divergence increases at the beginning and decreases during the later stage of training.

Table 4: Hyper-parameters during Training Stage

Parameter Name	Parameter Value
Batch size	400
Max iteration	10e5
z_{dim}	10
noise dimension (infoGAN)	32
Sequence length	50
learning rate (VAE)	0.0001
β_1 of Adam (β-VAE)	0.9
β_2 of Adam (β-VAE)	0.999
β_{base} in β-VAE	2
λmin in β-VAE	0.1
R_{step} in β-VAE	0.9995
learning rate (Discriminator in infoGAN)	0.0001
β_1 of Adam (Discriminator in infoGAN)	0.5
β_2 of Adam (Discriminator in infoGAN	0.99
learning rate (Generator in infoGAN)	0.0001
β_1 of Adam (Generator in infoGAN)	0.5
β_2 of Adam (Generator in infoGAN)	0.99

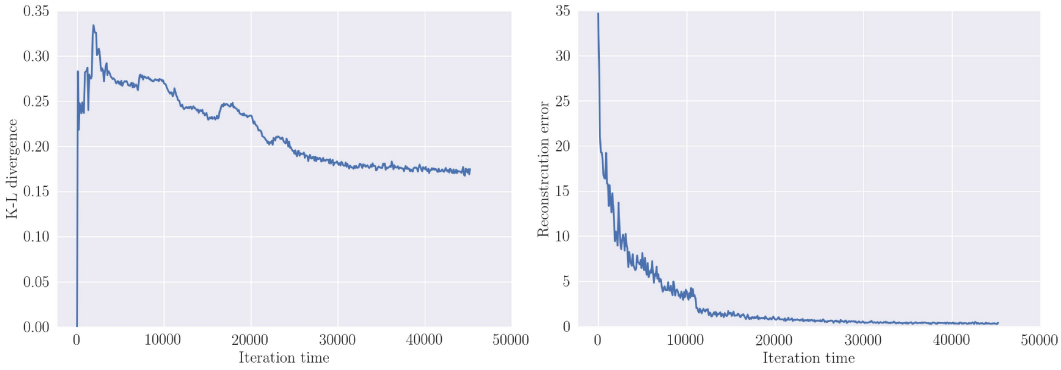

Figure 1: Loss curve during the training stage.

C More Experiment Results

C.1 More Generation Trajectories

Fig 2 shows more generated results. The method of generating these samples are same as the main text.

C.2 Zoomed Analysis

We display some zoomed trajectories to show more details about the generated results in Fig 3 and Fig 4. In Fig 3, more interpretable trajectories are displayed. Each group (row) contains 4 figures, and from left to right there exists continuous changes. In Fig 6, we also display some failure cases. Some of them do not show the traffic rationality, containing some sharp turn.

C.3 Time domain Analysis

Since we only display one time domain analysis result of MTG in main text, we add 8 more results in Fig 5. The three columns represent the corresponding distance, variant of speed and variant of direction respectively.

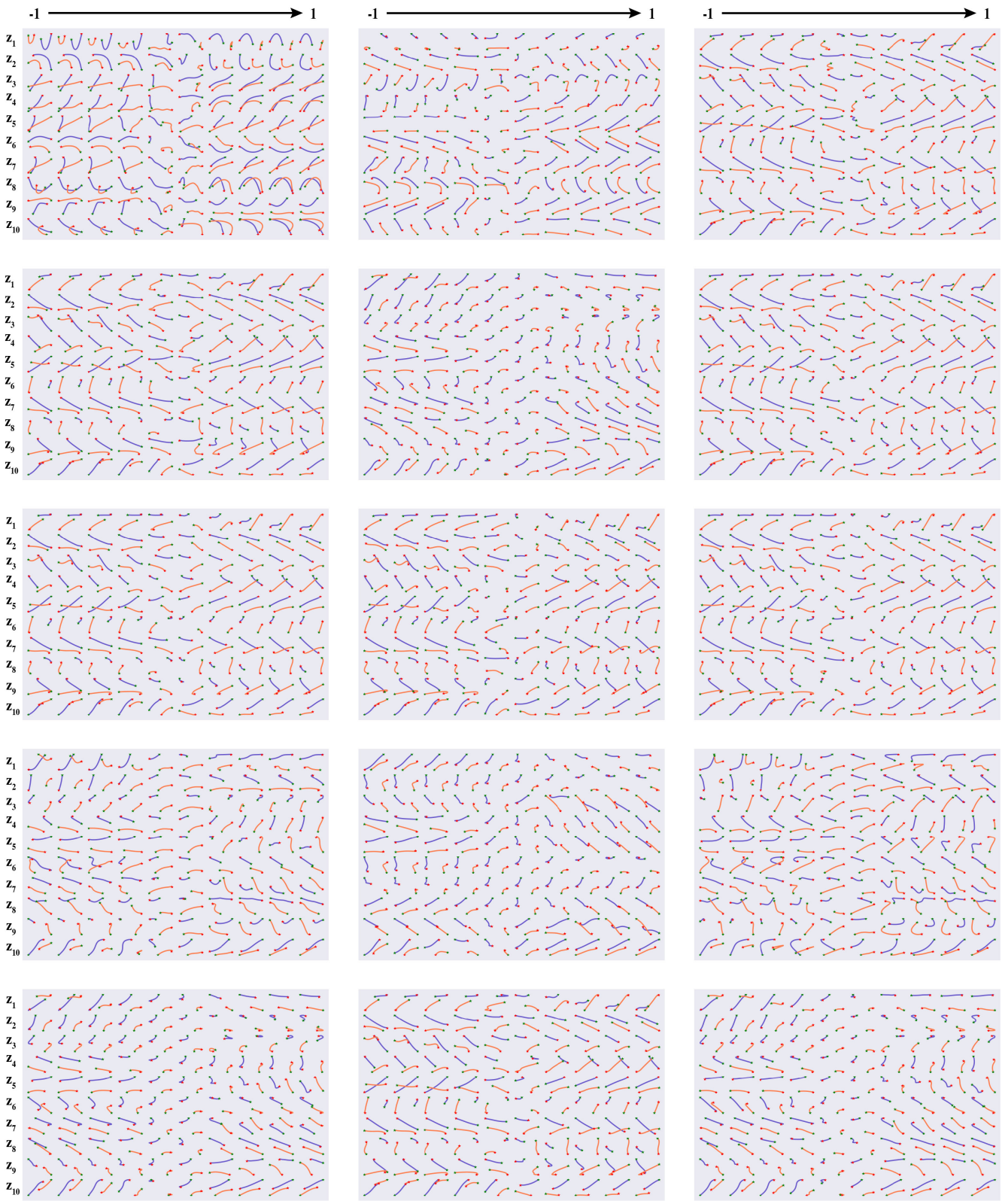

Figure 2: Generated results of MTG. In each figure, 100 trajectories are displayed. Each row of these figures is dominated one latent code (varies from -1 to 1).

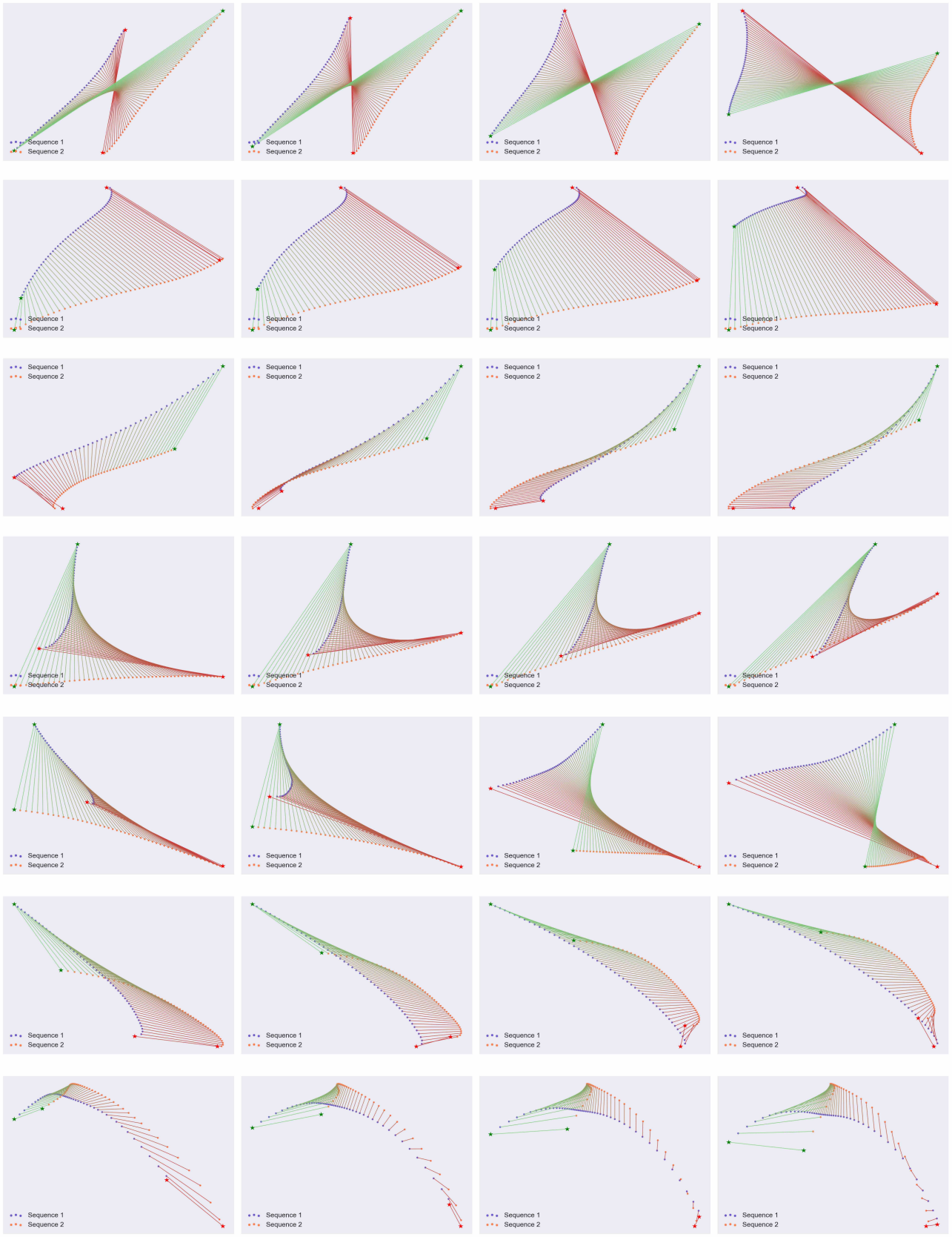

Figure 3: Zoomed generated results of MTG. Seven groups are displayed here, each of which contains 4 figures. In every row, one latent code is selected and set to 4 different values, then 4 figures are generated to show the variant of the encounter trajectories.

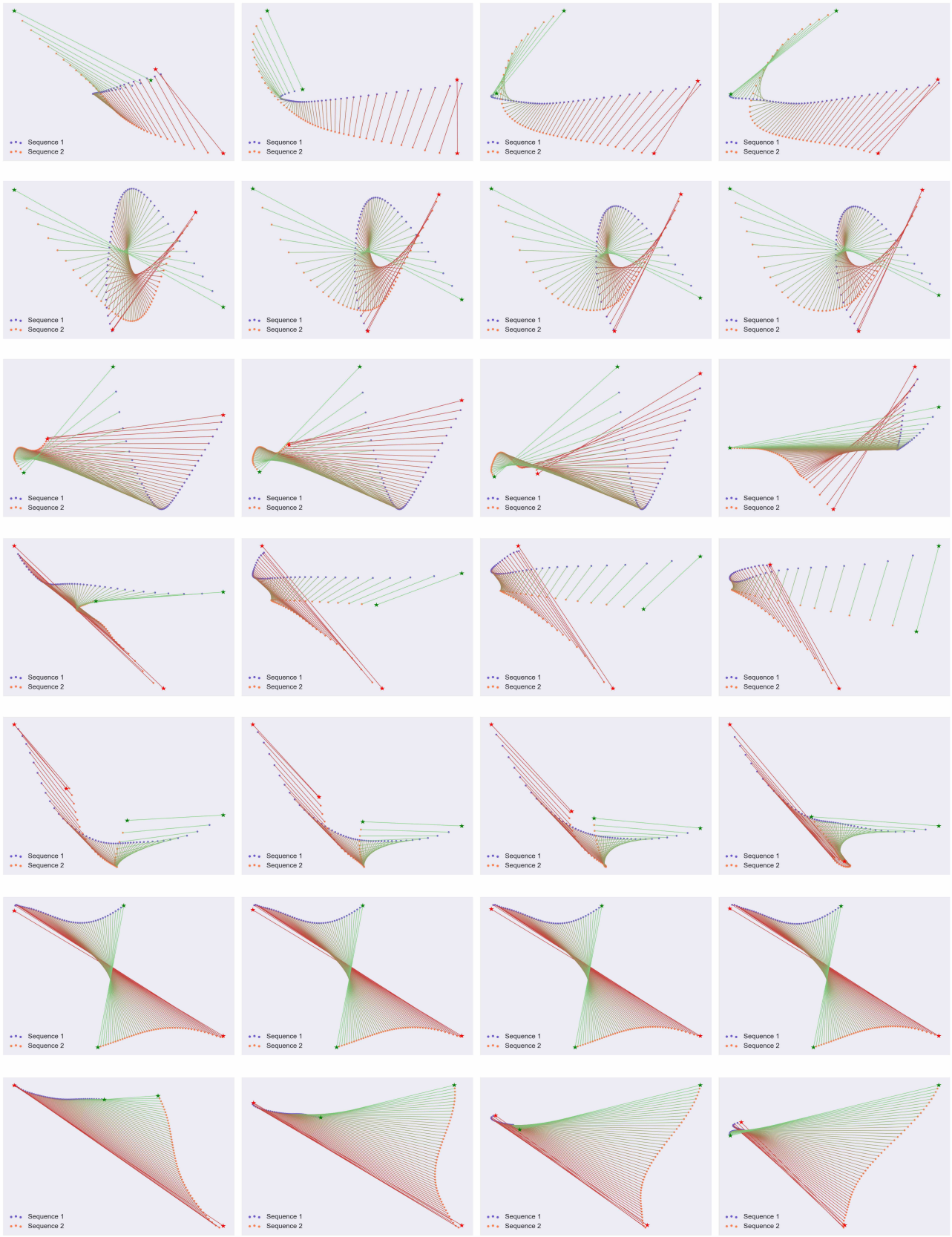

Figure 4: Failure cases of MTG. Seven failure cases are displayed, each of which contains 4 figures. In every row, one latent code is selected and set to 4 different values, then 4 figures are generated to show the variant of the encounter trajectories.

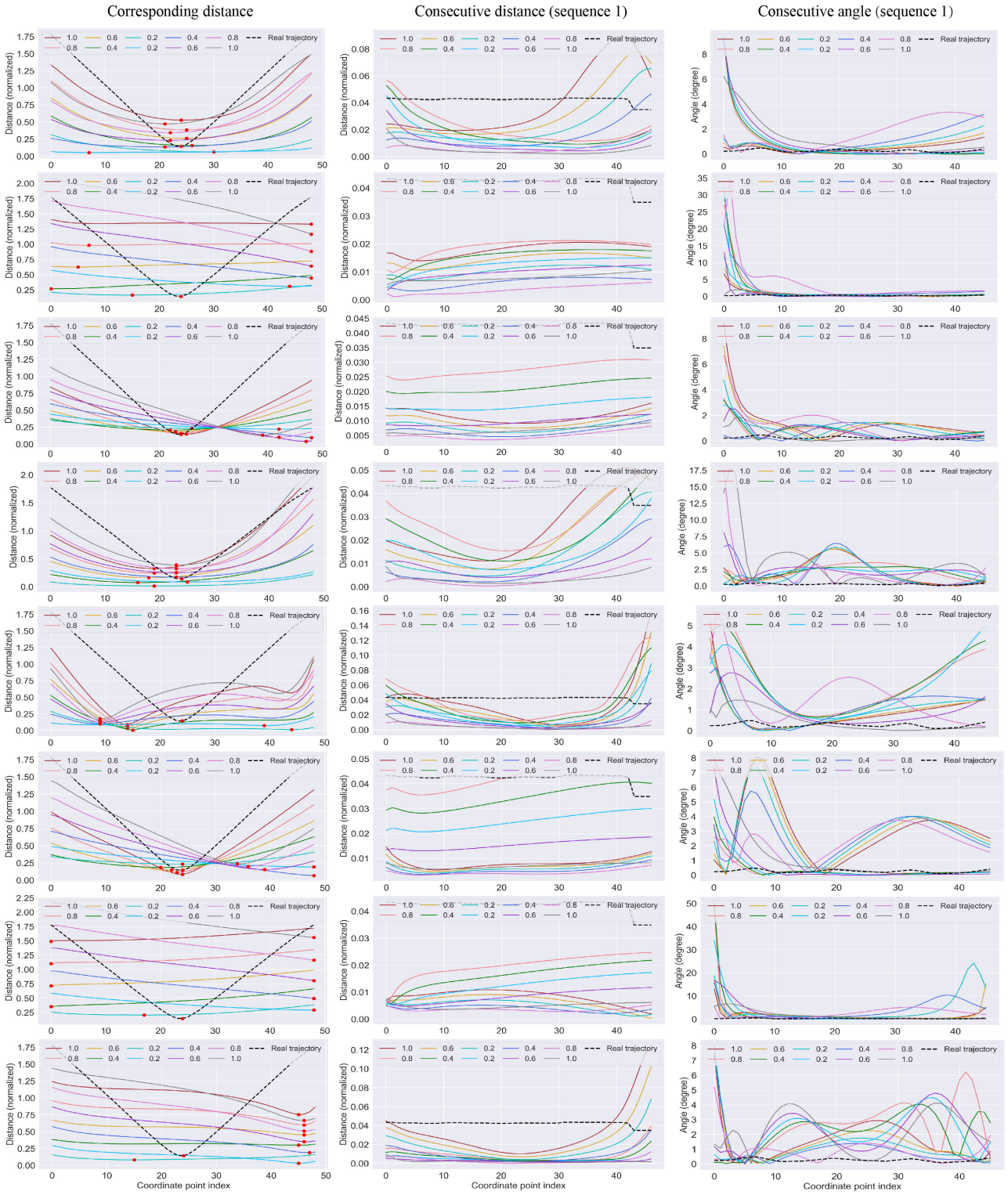

Figure 5: Time domain analysis of MTG results. The first column represents the distance between corresponding points, the second column represents the variant of speed and the third column represents the variant of direction.

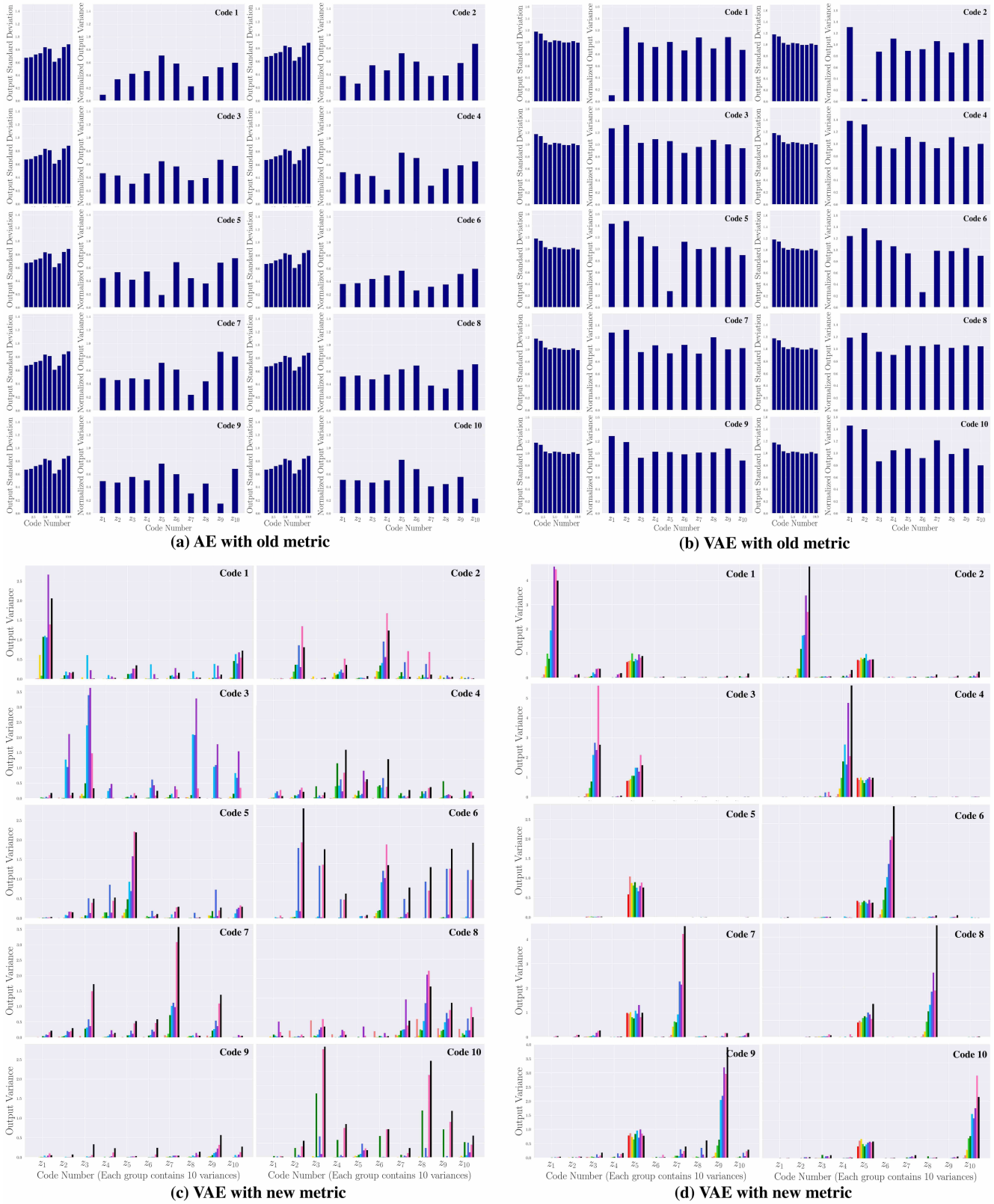

Figure 6: Comparison between old and new disentanglement metric. Results of Autoencoder tested on old metric and new metric are shown in (a) and (c) respectively. Results of VAE tested on old metric and new metric are shown in (b) and (d) respectively.

C.4 Disentanglement Metric

In order to compare our metric with the old one [3], we conduct 4 experiments with two metrics on both autoencoder and VAE. Restricted by space, we only display the result of z_6 in main text, and the rest codes are displayed in Fig 6.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computer Science*, 2014.

[2] D. Ha and D. Eck, "A neural representation of sketch drawings," *arXiv preprint arXiv:1704.03477*, 2017.

[3] H. Kim and A. Mnih, "Disentangling by factorising," *arXiv preprint arXiv:1802.05983*, 2018.