Scenario-based trajectory generation and density estimation towards risk analysis of autonomous vehicles

Citation for the original published paper (version of record):
Johansson, E., Sönnergaard, M., Selpi, S. et al (2023). Scenario-based trajectory generation and density estimation towards risk analysis of autonomous vehicles. IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC: 1375-1380. http://dx.doi.org/10.1109/ITSC57777.2023.10422694

Accepted by IEEE ITSC 2023

Scenario-based trajectory generation and density estimation towards risk analysis of autonomous vehicles

Edvin Johansson[1], Matilda Sönnergaard[1], Selpi[1], Sadegh Rahrovani[2] and Parsia Basimfar[2]

***Abstract*— A large amount of testing is needed to determine when autonomous vehicles are sufficiently safe. To achieve this goal, test cases should be representative of real-world driving but also designed to provide sufficient coverage of both frequent and rare events. This is a crucial step in finding potential high-consequence events, failure borders of the Autonomous Driving (AD) function and accurate estimation of the corresponding residual risks.**

In this paper, we propose a new method to adapt generative models to generate vehicle trajectories that are representative of the ones collected from the real world. The method uses Non-Uniform Rational B-Splines (NURBS) combined with normalizing flows to build a statistical scenario model. The method allows us to estimate a joint probability density that can be used to evaluate the likelihood of different trajectory occurrences.

We demonstrate the method for statistical modeling on the cut-in traffic scenario and we give an example of how the estimated joint probability distribution can be used to assess the risk (trajectory occurrence probability and criticality) for different test cases. The results can be used for accelerated testing purposes, where the aim is to sample the rare tests more frequently, but can also be used to calculate the failure probability of AD functions.

I. INTRODUCTION

Testing autonomous vehicles is crucial in ensuring their safety. An autonomous vehicle that has not been sufficiently tested could result in severe collisions. To make sure that such collisions do not occur the vehicle has to be tested in many situations. A prominent approach to testing autonomous vehicles is to divide these situations into scenarios, such as cut-ins, overtaking, car-following, etc. Autonomous vehicles can then be tested on these different scenarios individually, which is known as scenario-based testing.

Often the shape of the vehicle trajectory can be used to distinguish between different scenarios. Dividing collected data into these different scenarios can be done in different ways, for example by rule-based knowledge, clustering techniques [1] or a combination of them [2]. In this work, we divide the data into different scenarios using rule-based knowledge.

After dividing data into scenarios, different generative models can be used to generate more data within these scenarios, which can be used to further test the autonomous vehicle. This has the benefit of being cheaper than collecting more real data. There exist many generative models, such as Generative Adversarial Networks (GANs) [3], Variational Autoencoders (VAEs) [4], diffusion models [5] and normalizing flows [6], [7]. These models are constantly evolving and new methods are being introduced rapidly.

Some of these models have been applied to trajectory generation. For instance, Time-GANs [8] use an embedding and recovery function to reconstruct and learn temporal dynamics of data. TrajVAE and TrajGAN introduced by [9] use Long Short Term Memory (LSTM) cells and an embedding to lower the dimensionality. Lastly, in the work by [10], two types of GANs are adapted for trajectory generation. In each of these papers, a different method has been used in order to adapt generative models for trajectory generation.

Furthermore, in [11], a normalizing flows model is used to generate scenarios for testing autonomous vehicles. This has the benefit of making it possible to estimate a probability distribution of the data, which they used to estimate collision rate in the car-following scenario. However, in [11] they did not generate trajectories, but initial parameters for the scenario.

Therefore, in this paper, we describe a method of using Non-Uniform Rational B-Splines (NURBS) to approximate trajectory data into curve representations, which can be used more easily in generative models. Then, we show how these NURBS approximations can be used to train a normalizing flows model on cut-in trajectory data. An example of a cut-in trajectory can be seen in Fig. 1.

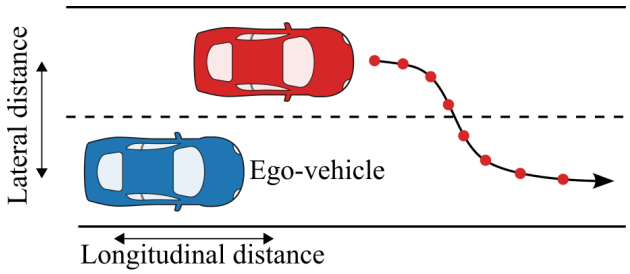

Fig. 1. An example of a cut-in trajectory. The target vehicle (red) cuts in front of the ego-vehicle (blue). The red dotted line is the shape of a cut-in trajectory.

We decided to use normalizing flows instead of other generative models because it can estimate a density, a probability distribution over multiple variables. For trajectory generation, this distribution would describe how likely a given trajectory is and how this likelihood varies as certain variables are changed. Then, using the definition of risk =

[1]Edvin Johansson, Matilda Sönnergaard and Selpi are with the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden edvjoh@student.chalmers.se, matsonn@student.chalmers.se, selpi@chalmers.se

[2]Sadegh Rahrovani and Parsia Basimfar are with the Autonomous Drive Department in VolvoCars, Sweden sadegh.rahrovani@volvocars.com, parsia.basimfar@volvocars.com

severity $*$ probability as defined in [12], it is possible to estimate the risk associated with trajectories, given some measure of severity. This can help decide which type of trajectories the autonomous vehicle needs to be tested more on, as a way to accelerate testing.

We chose to use NURBS curves to represent the trajectories because of three main advantages. Firstly, it lets us use a fixed number of parameters for each trajectory. This means that cut-in trajectories of different lengths are sent into the generative model with the same number of dimensions. Secondly, it reduces the dimensionality of the data. Thirdly, NURBS also have a smoothening effect on trajectories, which is useful, as the trajectory data collected from real-world driving is often noisy.

II. METHODOLOGY

An overview of the methodology used is given in Fig. 2: The trajectories are approximated by NURBS. The NURBS are then normalized to improve the normalizing flows model performance. The normalized values of the NURBS together with the total length in seconds of the trajectory t are then used to train the normalizing flows model. After training, the model can be used to both generate NURBS using the generative direction and estimate the probability of NURBS using the normalizing direction. Each of these steps will be explained in detail in the rest of this section.

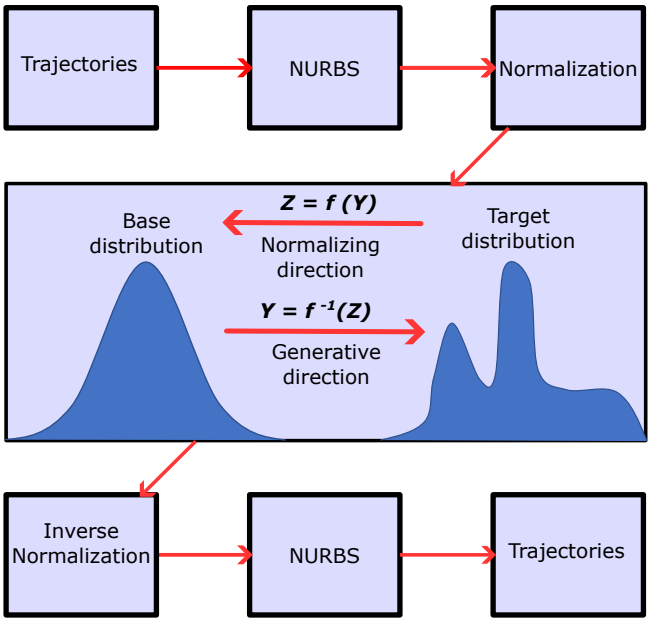

Fig. 2. An overview of the method presented in this paper. Trajectory data points are transformed into NURBS. The parameters of the NURBS are normalized and then sent into the normalizing flows model. To generate trajectories, the inverse normalization is needed.

A. Non-uniform rational B-splines

Before the data is sent into the normalizing flows model, it is approximated by NURBS. In general, other methods can also be used to fit different kinds of approximations of the trajectory. One example is polynomials, where the coefficients can be used to represent the trajectory. We found however that NURBS have many benefits. For instance, in comparison with polynomials, they have a designated start and end point.

This section gives a brief overview of NURBS. More details can be read from *The NURBS Book* [13]. Since we are working with trajectories, when we mention NURBS we specifically mean NURBS curves and not NURBS surfaces.

NURBS are defined by three things: a degree p, n control points $\mathbf{P}_i = (x_i, y_i, w_i)$ in which (x_i, y_i) coordinates are combined with weights w_i, and a knot vector $\mathbf{U} = (u_1, \ldots, u_m)$ in non-descending order of length $m = n+p+1$. Intuitively, the curve is pulled towards the control points. The degree determines how many control points pull a given point on the curve and the knot vector decides the limits at which different control points should pull the curve.

Fitting a NURBS curve of a specific degree to a set of t trajectory points $\{\mathbf{Q}_1, \ldots, \mathbf{Q}_t\}$ where $\mathbf{Q}_i = (x_i, y_i)$, is a nonlinear optimization problem if no additional conditions are specified [13]. One algorithm described in *The NURBS Book* [13, Chapter 9.4.1] makes the problem linear by first specifying the number of control points, setting all weights to 1 and specifying the knot vector according to some heuristic. Then, solving the x and y values of the control points can be done using least squares approximation.

There exist different heuristics to specify the knot vector. Here, we will only describe the heuristic used in this work. Furthermore, we decided that our NURBS should start at the first point and end at the last point of our trajectory. This is done by placing the first and last control points at the first and last points of the trajectory, and setting the first $p+1$ (degree plus one) and last $p+1$ values of the knot vector to 0s and 1s respectively. The values in between these 0s and 1s are then called the internal knot vector. To simplify notation we specify the internal knot vector as $k_1, \ldots k_l = u_{p+2} \ldots u_n$, which implies that $l = n - p - 1$ is the number of internal knot vector values.

The heuristic used for specifying the knot vector is then to have the internal knot vector values spaced evenly relative to the Euclidean distance between data points. More formally, given the normalized cumulative sum of distances between points

$$d_q = \frac{\sum_{i=1}^{q} |\mathbf{Q}_{i+1} - \mathbf{Q}_i|}{\sum_{j=1}^{t-1} |\mathbf{Q}_{j+1} - \mathbf{Q}_j|}, \tag{1}$$

computed for each $1 \leq q \leq t-1$, the internal knot vector values $k_1 \ldots k_l$ are spaced evenly by the index $s = \text{floor}(j(t-1)/(l+1))$ such that

$$k_j = (1-a)\, d_s + a d_{s+1}, \tag{2}$$

where $a = j(t-1)/(l+1) - s$.

The intuitive interpretation of this is that the internal knot vector describes percentage distance to percentage time along the trajectory. For instance, with an internal knot vector of length $l = 3$ and with a value $k_2 = 0.6$, we have reached $60\% = k_2$ of the distance when we are at $50\% = 2/(l+1)$

of the total time. If the trajectory is that of a car, it would indicate that the car is driving slightly slower than average at the later half of the trajectory.

B. Normalizing flows

The generative model we used to generate trajectories is the normalizing flows model, mostly because of its potential to estimate a probability density over trajectories. In this section, a brief overview of normalizing flows is given and the specific type of normalizing flows we used is explained.

Normalizing flows models use a number of flows, which are invertible and differentiable functions $f_1, \ldots, f_n$. These transform a complicated distribution into, usually, a normal distribution, hence the name normalizing flows. These flows together form the transformation $f = f_1 \circ \ldots \circ f_n$, which is then also invertible and differentiable. Then, let $\mathbf{Z}$ be a random variable with a simple density function $p_{\mathbf{Z}}$, such as the normal distribution, and the random variable $\mathbf{Y} = f^{-1}(\mathbf{Z})$. The change of variables formula can be used to calculate the density function $p_{\mathbf{Y}}$ [14]:

$$p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Z}}(f(\mathbf{y})) \left| \det \left(\frac{\partial f}{\partial \mathbf{y}} \right) \right|. \tag{3}$$

A normalizing flows model then estimates the transformation f to fulfill this equation. The flows are parameterized by parameters ϕ such that $f(\mathbf{y}; \phi)$. Also, if not using the normal distribution, $p_{\mathbf{Z}}(\mathbf{z})$ can be parameterized by ψ such that $p_{\mathbf{Z}}(\mathbf{z}; \psi)$. To train a normalizing flows model, the log-likelihood is maximized with respect to the parameters $\theta = \{\phi, \psi\}$. In other words, during training

$$\begin{aligned} \sum_{i=1}^{N} \log p_{\mathbf{Y}}(\mathbf{y}_i; \theta) = \sum_{i=1}^{N} \Bigg(& \log p_{\mathbf{Z}}(f(\mathbf{y}_i; \phi); \psi) \\ & + \log \left| \det \left(\frac{\partial f}{\partial \mathbf{y}}(\mathbf{y}_i; \phi) \right) \right| \Bigg) \end{aligned} \tag{4}$$

is maximized by changing the parameters θ.

A trained normalizing flows model can then be used to both estimate the probability of samples and generate new ones. Estimating the probability of a sample is done by propagating the sample using (3). Generation is done by sampling from $p_{\mathbf{Z}}$ and using the inverse f^{-1} to get a point $\mathbf{y}$. $p_{\mathbf{Z}}$ is called the base distribution and $p_{\mathbf{Y}}$ the target distribution.

There has a lot of research on the different types of flows that can be used in normalizing flows models. For a comparison between them, please see [14]. In our work, we used the autoregressive variant of neural spline flows introduced by [15]. These specific flows were used because they showed good performance on a range of different datasets in [14].

The flows in [15] consist of multiple compositions of linear flows and rational-quadratic neural spline flows. The rational quadratic neural splines perform an advanced transformation of the input and are mostly used to introduce non-linearities. For a detailed description of them, see [15]. The linear flows are defined by a function $g(\mathbf{z}) = A\mathbf{z} + b$ on the input $\mathbf{z}$, where A is a matrix and b is a number. To ensure that A is invertible, we use LU decomposition, defining A as $A = LU$. Here, L and U are lower and upper triangular matrices respectively, where U has ones on the diagonal and L has a positive diagonal. The elements of L, U and b are trained and used to reconstruct A. It is common to decompose $A = PLU$ where P is a permutation matrix with fixed values, however, we did not find that this improved performance.

C. Preparation and training

In order to combine normalizing flows with NURBS, the first step is to approximate all trajectories with NURBS, which is done with the algorithm described in Sect. II-A. The normalized values of $[t, k_1, \ldots, k_{n-p-1}, x_1, y_1, \ldots, x_n, y_n]$ are then sent into the normalizing flows model, where t is the total time in seconds of the trajectory, $k_1, \ldots, k_{n-p-1}$ is the internal knot vector and $x_1, y_1, \ldots, x_n, y_n$ are the x and y coordinates of the control points.

These values can be used to fully reconstruct NURBS. Furthermore, because the total time t is sent into the model, $n - p - 1$ time stamps can be estimated along the NURBS. This is done by using the way that the internal knot vector was approximated, explained in Sect. II-A. For example, with a knot value of $k_2 = 0.6$ and the number of internal knots $3 = n - p - 1$, integration along the NURBS curve can be done until $60\% = k_2$ of the total distance is reached. The timestamp at this position then becomes $t_2 = t \times 2/(n-p) = t/2$. Thus, by using NURBS we do not only get the shape of the trajectory but also estimated temporal properties.

The normalization is done so that each input z is transformed into $z' = (z - \mu_z)/\sigma_z$, where μ_z is the mean value of the data and σ_z the standard deviation. Generation is then done by applying the inverse transform of the normalization ($z = z'\sigma_z + \mu_z$). The normalization greatly increased training convergence speed and somewhat the performance of the normalizing flows model. This is likely due to the large difference in ranges of the parameter values. For instance, the standard deviation of the first value of the internal knot vector is $\sigma_{k_1} \approx 0.01$, while the standard deviation for the x coordinate of the first control point is $\sigma_{x_1} \approx 30$.

D. Metrics for evaluation

Evaluating a normalizing flows model is often done by comparing the test log-likelihood, as done in [14], [16]. While this compares between models well, it does not give a concrete value of how well the models perform. Because of this, we used two metrics, *matching* and *coverage*, which were introduced in [10]. Matching measures how "realistic", in terms of how similar the generated trajectories are to the real data. Coverage measures the model's ability to generate trajectories similar to all different types of trajectories in the real data.

Formally, matching is defined as

$$\text{matching} = \frac{\sum_i^m \min_j(\text{dist}(G_i, R_j))}{m} \tag{5}$$

where G_m is the set of m generated trajectories, R_n the set of n real trajectories, and dist is some function defining the distance between two trajectories. In our case, as in [10], we used Dynamic Time Warping (DTW) [17].

Note that multiple generated trajectories can be matched to the same trajectory in the real data. This means that if many generated trajectories are closely matched to the same real trajectory, we can get a low matching score, even if the model is giving an inaccurate representation of the data. Hence we also use coverage, which is a measure of how many real trajectories are "covered" by a set of generated trajectories. Mathematically, it is defined as follows:

$$\text{coverage} = \frac{|\text{argmin}_j(\text{dist}(G_i, R_j)), \forall i = \overline{1, \text{m}}|}{n}. \quad (6)$$

Similarly to matching, it is not enough to simply have a good (high) coverage score, as this can sometimes be achieved by generating trajectories with high diversity, even if they have a low resemblance to the real trajectories. Conversely, if the trajectories in the real data are very similar, many generated trajectories may cover the same real trajectory. This leads to low coverage, even if the generated trajectories are similar to all of the real trajectories.

In general, however, finding a model with low matching and high coverage should correspond to a model that is able to capture a majority of the trajectories in the real data. The exact values are however dependent on the data, and as such, it is useful to measure the matching and coverage between two distinct sets of non-generated data as a baseline. We should then be able to consider a model that gets both matching and coverage close to this baseline as accurate.

E. Risk estimation

With normalizing flows models, it is possible to estimate the likelihood of both generated and real trajectories. This has multiple uses for risk estimation and can be used with the definition: risk $=$ severity $*$ likelihood. As a primitive example, we have defined the severity of a trajectory as

$$\text{severity} = \exp\left(-\frac{\min_i\left(x_i^2 + y_i^2\right)}{\alpha}\right) \quad (7)$$

for some parameter α and where the minimum is taken over all points in the trajectory. This is based on the Distance of Closest Encounter [18] and ensures that the severity reaches a maximum value of 1 only in the case of a collision. The parameter α affects the distribution of the severity; a smaller value concentrates the severity such that only the closest trajectories get a high severity, whereas a greater value gives a more even distribution. This definition of severity was used as an example in this paper, but can easily be changed to other severity measures such as those suggested in [18], [19].

III. EXPERIMENTS

In this section, we first show an example of a distribution deduced from the normalizing flows model adapted for trajectory generation. The example is shown in Fig. 3 which shows the estimated distribution of the initial absolute relative velocity and the initial longitudinal distance. Then, we evaluate the adapted normalizing flows model by performing two experiments. Firstly, we compare the result from changing the standard deviation when sampling from the base distribution. Secondly, we show how the probability distribution estimated by the model can be used to estimate the risk of trajectories.

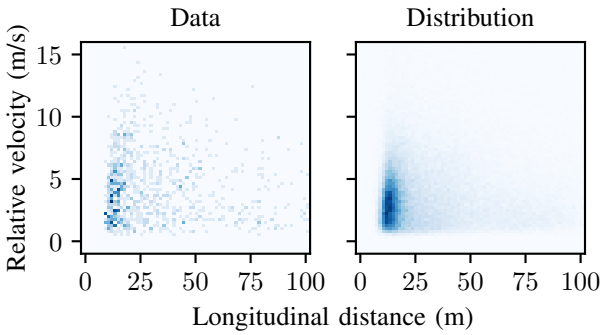

Fig. 3. An example of distributions that can be estimated with the normalizing flows model. Here, the distribution of initial relative absolute velocity between the vehicles against the initial longitudinal distance between them is shown, together with the values for the original data.

The normalizing flows model was trained on a set of roughly 15000 cut-in trajectories. After 200 iterations, the model showed signs of overfitting, indicating that more data was required or that the model had already learned the overall shape of the distribution. After roughly 1000 iterations, training was stopped because the log test-loss converged to around -21 nats.

After training, the model was evaluated by sampling from the base distribution with different standard deviations, essentially generating trajectories with varying likelihood. Fig. 4 shows some examples of generated trajectories sampled from the base distribution using different standard deviations. The lower the standard deviation, the higher the likelihood of the generated trajectories. The more likely trajectories look more like what we might think of as ideal trajectories, because of their smooth and simple shape. This is especially clear with a standard deviation $\sigma = 0.1$ in Fig. 4, where all of the trajectories have a very smooth shape.

To evaluate the model, matching and coverage were also calculated for samples generated with different standard deviations. Matching and coverage were calculated with a generated dataset of 400 trajectories and a real dataset of 100 trajectories so that the results could be compared with those of [10]. To estimate the accuracy of the result, the calculation of matching and coverage was done 50 times for different samples in the generated and real dataset. The results can be seen in Fig. 5. To compare the matching and coverage values to a baseline, they were also calculated between two distinct real datasets.

It can be seen from Fig. 5 that, as we decrease the standard deviation, the matching decreases since the generated trajectories become more ideal and similar to the real ones. At

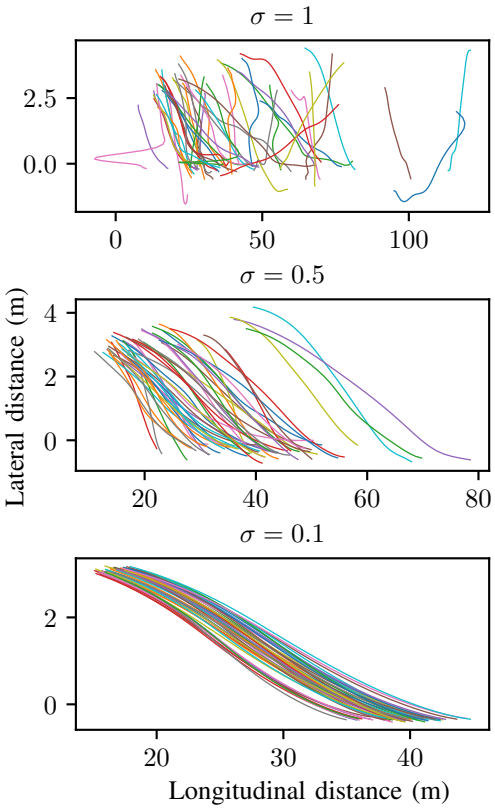

Fig. 4. Generated trajectories from the normalizing flows model sampled with different standard deviations σ.

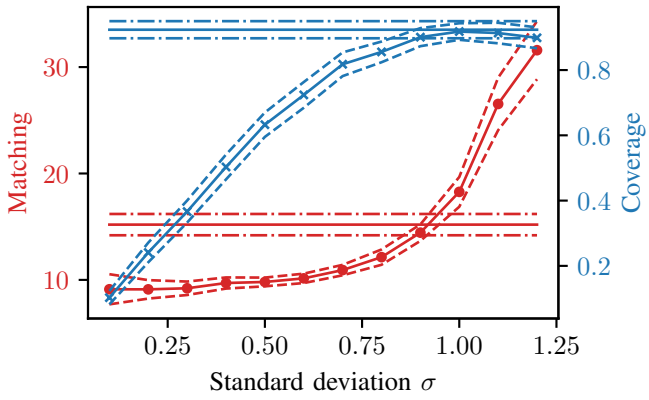

Fig. 5. Matching and coverage for samples generated with different standard deviations σ. The horizontal lines are matching and coverage values between two distinct real datasets. The dotted lines are the matching (circles) and coverage (crosses) values for different sampling standard deviations σ. The tests were computed 50 times each and the dashed and dashed-dotted lines represent the standard deviation in those tests.

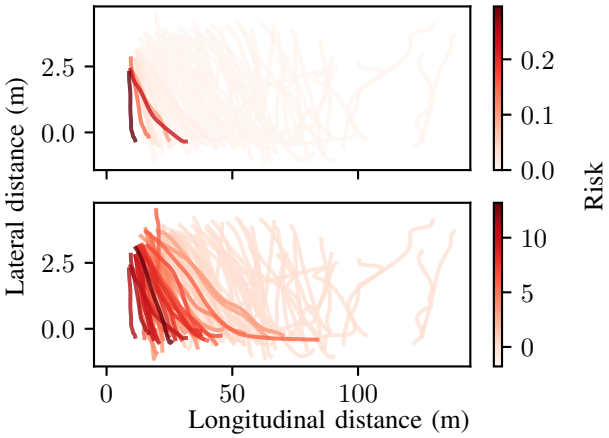

Fig. 6. Generated trajectories, color coded by estimated *risk*, calculated for different values of α (top: $\alpha = 20$, bottom: $\alpha = 200$). A deeper color indicates a higher risk. Note that the colors are normalized for each plot and are therefore not directly comparable between the plots.

the same time, the coverage decreases because the generated trajectories become very similar to each other and do not cover many of the less likely real trajectories.

To further evaluate and demonstrate our method, the risk = severity $*$ likelihood was calculated by using the severity defined in (7) and the likelihood from the normalizing flows model. Fig. 6 shows a set of generated trajectories color-coded by this risk computed using different values of α. This demonstrates how risk can be calculated by using different severity measures together with our normalizing flows model. As α increases, the trajectories are weighted more by how likely they are, rather than how severe they are.

IV. DISCUSSION

The work in this paper shows how NURBS can be used to represent and approximate trajectories for generative models. NURBS are well suited for representing trajectories, as they can be fitted quickly and accurately to complex trajectories and are able to embed time (and hence velocity). However, they are not without issues. NURBS are good at representing smooth curves, but this also means that it is difficult to fit them to sharp corners. As we have mainly worked with trajectories of cut-ins, there are often such corners at the beginning or end of the trajectories.

Furthermore, because the linear fitting algorithm described in Sect. II-A assumes all control point weights to be equal, certain shapes are impossible to represent exactly, such as circles. While being unable to represent certain shapes is not likely to affect trajectory approximation significantly, it may be worth investigating how using the weights can affect the fitting accuracy, especially as we have a fixed number of control points for all trajectories. With that said, the goal is not necessarily to minimize the fitting error, as the trajectory data can contain noise that we do not want to overfit.

For generation, the normalizing flows model seems to generate realistic trajectories. However, unrestricted sampling from the normal distribution can generate unrealistic and physically insensible trajectories. For instance, as time is decided by a single parameter, trajectories are occasionally generated with negative total time t. Thus it is important to consider the probability of the generated trajectories. By sampling from a narrower base distribution, as shown in Fig. 4, many unrealistic trajectories can be avoided.

The normalizing flows model we trained estimates the

probability of trajectories. Combining the probability with some measure of severity, criticality or importance helps decide which type of trajectories the autonomous vehicle needs to be further tested on. In this paper, we showed an example of how the probability can be combined with a severity measure. However, there exist many other methods of deciding which trajectories are severe or important. For instance, in [11], importance sampling is used together with normalizing flows to generate more collision scenarios. Also, in [20], trajectories are perturbed to make collisions more likely. Both of these methods and others could make use of knowing the probability of the trajectories. Because then we would not only generate interesting scenarios but also know how likely they are to appear in normal traffic.

V. CONCLUSION

We have presented a new method that uses NURBS to approximate trajectories and then uses normalizing flows for trajectory generation. The method seems to perform well, based on visualizations and the use of matching and coverage scores. It is however challenging to assess the performance of a generative model concretely.

We have also shown how the new method can be used to estimate a probability distribution of trajectories, which in turn can be used together with a severity measure to estimate the risk associated with the trajectories. The latter helps in assessing which trajectories autonomous vehicles need to be tested on.

The new method has only been applied on cut-in scenarios. Future work could look at how the method performs for other traffic scenarios.

ACKNOWLEDGEMENT

We would like to acknowledge Volvo Cars for providing the data and the computational resources. We also thank Mikael Andersson from Volvo Cars, Team Risk Estimation & Monitoring, who provided good discussion on the topic of NURBS and helped us throughout the project.

REFERENCES

[1] F. S. Hoseini, S. Rahrovani, and M. H. Chehreghani, "A Generic Framework for Clustering Vehicle Motion Trajectories," Sept. 2020. [Online]. Available: http://arxiv.org/abs/2009.12443

[2] S. Jarl, L. Aronsson, S. Rahrovani, and M. H. Chehreghani, "Active learning of driving scenario trajectories," *Engineering Applications of Artificial Intelligence*, vol. 113, p. 104972, Aug. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0952197622001750

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3422622

[4] L. P. Cinelli, M. A. Marins, E. A. Barros da Silva, and S. L. Netto, *Variational methods for machine learning with applications to deep networks*. Cham: Springer, 2021, oCLC: 1251443306.

[5] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, July 2015, pp. 2256–2265. [Online]. Available: https://proceedings.mlr.press/v37/sohl-dickstein15.html

[6] E. G. Tabak and C. V. Turner, "A Family of Nonparametric Density Estimation Algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, Feb. 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cpa.21423

[7] D. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, July 2015, pp. 1530–1538. [Online]. Available: https://proceedings.mlr.press/v37/rezende15.html

[8] J. Yoon, D. Jarrett, and M. Van der Schaar, "Time-series generative adversarial networks," *Advances in neural information processing systems*, vol. 32, 2019.

[9] X. Chen, J. Xu, R. Zhou, W. Chen, J. Fang, and C. Liu, "TrajVAE: A Variational AutoEncoder model for trajectory generation," *Neurocomputing*, vol. 428, pp. 332–339, Mar. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231220312017

[10] A. Demetriou, H. Alfsvåg, S. Rahrovani, and M. Haghir Chehreghani, "A Deep Learning Framework for Generation and Analysis of Driving Scenario Trajectories," *SN Computer Science*, vol. 4, no. 3, p. 251, Mar. 2023. [Online]. Available: https://link.springer.com/10.1007/s42979-023-01714-3

[11] H. Zhang, J. Sun, and Y. Tian, "Accelerated Testing for Highly Automated Vehicles: A Combined Method Based on Importance Sampling and Normalizing Flows," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. Macau, China: IEEE, Oct. 2022, pp. 574–579. [Online]. Available: https://ieeexplore.ieee.org/document/9922218/

[12] I. ISO, "26262: 2018:"Road vehicles—Functional safety"," *British Standards Institute*, vol. 12, 2018.

[13] L. Piegl and W. Tiller, *The NURBS Book*, ser. Monographs in Visual Communications. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. [Online]. Available: http://link.springer.com/10.1007/978-3-642-97385-7

[14] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing Flows: An Introduction and Review of Current Methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964–3979, Nov. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9089305/

[15] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," *Advances in neural information processing systems*, vol. 32, 2019.

[16] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing Flows for Probabilistic Modeling and Inference," *J. Mach. Learn. Res.*, vol. 22, no. 1, July 2022, publisher: JMLR.org.

[17] A. Mueen and E. Keogh, "Extracting Optimal Performance from Dynamic Time Warping," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, Aug. 2016, pp. 2129–2130. [Online]. Available: https://dl.acm.org/doi/10.1145/2939672.2945383

[18] L. Westhofen, C. Neurohr, T. Koopmann, M. Butz, B. Schütt, F. Utesch, B. Neurohr, C. Gutenkunst, and E. Böde, "Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art," *Archives of Computational Methods in Engineering*, vol. 30, no. 1, pp. 1–35, Jan. 2023. [Online]. Available: https://link.springer.com/10.1007/s11831-022-09788-7

[19] S. S. Mahmud, L. Ferreira, M. S. Hoque, and A. Tavassoli, "Application of proximal surrogate indicators for safety evaluation: A review of recent developments and research needs," *IATSS Research*, vol. 41, no. 4, pp. 153–163, Dec. 2017. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0386111217300286

[20] D. Rempe, J. Philion, L. J. Guibas, S. Fidler, and O. Litany, "Generating Useful Accident-Prone Driving Scenarios via a Learned Traffic Prior," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 17 305–17 315.