



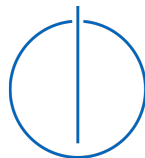
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Guided Research

Traffic Trajectory Prediction Framework within Providentia++

Author: Jurek Olden
Supervisors: M.Sc. Christian Creß, M.Sc. Walter Zimmer
Submission Date: 30.07.2021



Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 1.1 | The Providentia++ Project | 2 |
| 1.2 | Trajectory Prediction | 2 |
| 1.3 | Project Goals | 3 |
| 2 | Related Work | 3 |
| 3 | Framework | 4 |
| 3.1 | Overview | 4 |
| 3.2 | Transferring Data To InfluxDB | 5 |
| 3.3 | Integrating PyTorch Models | 5 |
| 3.4 | Visualization Using CARLA | 6 |
| 3.5 | Real Time Validation | 7 |
| 4 | Data Annotation and Preprocessing | 9 |
| 5 | Prediction Models | 10 |
| 5.1 | Constant Velocity Model | 10 |
| 5.2 | Vanilla LSTM | 10 |
| 5.3 | FloMo | 11 |
| 6 | Outlook | 13 |
| 6.1 | Modelling | 13 |
| 6.2 | Robustness | 14 |
| | References | 14 |

ABSTRACT

The purpose of the Providentia++ project is to set up digital imagery and infrastructure along the german A9 highway in order to create a real time “digital twin” of the traffic. Ultimately, the goal is to also provide services to various stakeholders such as autonomous cars or smart traffic lights, among others. One obvious service is the prediction of short-term trajectories of traffic participants, which may be used for e.g. collision avoidance or lane change optimization in autonomous cars. In this report we present the implementation of a trajectory prediction framework within the Providentia++ backend. This framework allows researchers to easily export training data from the digital twin and to integrate models into the backend, while also providing real-time visualization and model performance metrics. As an implementation and performance reference, we integrated three models: two baseline models - a constant velocity model and a vanilla LSTM - and a state of the art motion prediction model,

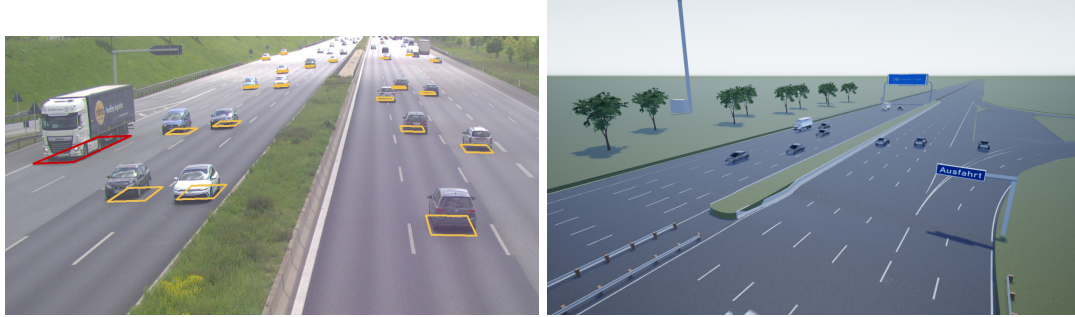


Figure 1: Providentia++ object detection and visualization (different scenes)

FloMo [1]. The whole framework can be integrated into the live system using a docker image and be used to provide live visualization and metrics.

1 Introduction

1.1 The Providentia++ Project

The core of the Providentia++ project [2] consists of camera and sensor hardware along a test stretch, which includes a part of highway A9 near Garching in Germany and is being extended into the city of Garching, and a collection of software modules performing various tasks such as handling video streams, performing object detection and merging data from the roadside measuring stations. These modules run on different infrastructure such as computers on site at a measuring station or servers. Communication between modules, or nodes, is handled by the *Robot Operating System (ROS, [3])*. Each software module is called a *ROS node* that may publish messages on a topic or subscribe to a topic. The *ROS core* coordinates message delivery over TCP/IP. The final output of the Providentia++ backend publishes messages encapsulating the current state of all detected objects (cars, trucks etc.) on the test stretch, merged over all measuring stations, along with some metadata such as the weather. This object data is what our framework is able to export as training data and what is used as input to the integrated models in order to produce live predictions. The live predictions are then published as *ROS* messages of their own. The Providentia++ project already provides visualisation of the digital twin based on the CARLA simulator [4] (Figure 1, page 2), which we will extend to also show predicted trajectories.

1.2 Trajectory Prediction

Trajectory prediction is the task of "predicting future positions of dynamic agents" [5]. It is an important concept in autonomous driving: after object detection and tracking, it allows autonomous agents to incorporate the future state of a scene in order to optimize

motion planning before entering the control phase. Eventually, trajectory prediction combined with other predictors may be used to avoid potentially dangerous situations, in particular when driving alongside human actors.

1.3 Project Goals

In order to integrate a flexible trajectory prediction framework to the Providentia++ backend, we identified three main responsibilities:

- relay object data to external tools as training data
- provide a flexible interface for plugging in externally created models, e.g. from a Python environment
- validate models in real time against the ground truth which is presented as new states of the digital twin

On top of implementing the trajectory prediction framework, integration tasks needed to be solved:

- adding visualized trajectories to the existing visualisation (Figure 1, page 2).
- provide metrics of the current model

Finally, to integrate the framework itself into the Providentia++ infrastructure, we need to pack all modules and their dependencies, including the chosen model, into a docker image for flexible deployment.

On top of the framework, we integrated three models in order to give baseline performance metrics and a reference implementation. One model is 'FloMo', the state of the art motion prediction model which was created in the context of Providentia++.

2 Related Work

Human motion trajectory prediction is an active field of research with many different approaches and ideas [6][7][8][1]. Rudenko et al. give an overview of current (2019) approaches [5]. Autonomous driving projects and digital augmentation of traffic, similar to Providentia++, is also a major field of research and a 'hot topic'; The work of Erdelean et al [9] gives a catalogue of "connected and automated driving test sites" up until February 2019.

Nachiket et al. propose a LSTM based encoder-decoder architecture using convolutional social pooling layers "as an improvement to social pooling layers for robustly learning interdependencies in vehicle motion". Additionally, predictions are based on 'maneuver classes' and the model produces a multimodal output [6].

Liu et al. address the issue of generalization "when the training set lacks examples collected from dangerous scenarios". To this end, samples are encoded such that ones

with positive future events may distinguished from ones with negative future events using contrastive loss [7].

Mangalam et al. model uncertainty from agents long-term goals and uncertainty stemming from randomness and the intent of other agents through multimodality in long-term goals and multimodality in waypoints and paths. They introduce a model exploiting this structure [8].

Schöller et al. assert that current models based on generative neural networks either do not reliably learn the true trajectory distribution or "do not allow likelihoods to be associated with predictions". They formulate trajectory prediction as a "density estimation problem with normalizing flow between a noise sample and the future motion distribution" and propose a model based on maximum likelihood estimation [1]. This model was created in the context of the Providentia++ project and was integrated into the Providentia++ backend as part of this work.

3 Framework

3.1 Overview

Our framework, analog to the formulated goals (Section 1.3, page 3), consists of five ROS nodes. The 'transfer' node is responsible for data export, it subscribes to the digital twin output ROS topic and converts and relays this data. The 'prediction' node is able to load external prediction models, input the digital twin output to the model and publish the resulting predictions as a ROS message on a separate topic. The 'validation' node subscribes to both the digital twin output as well as the prediction output and generates the RMSE score of the currently running prediction model. This score is then published via ROS messages for further usage, e.g. to display metrics on the Providentia++ homepage. Additionally, we used the prediction output to extend the existing CARLA visualization module to display the predicted trajectory for each object in real time, updated with each new inference iteration. We also implemented

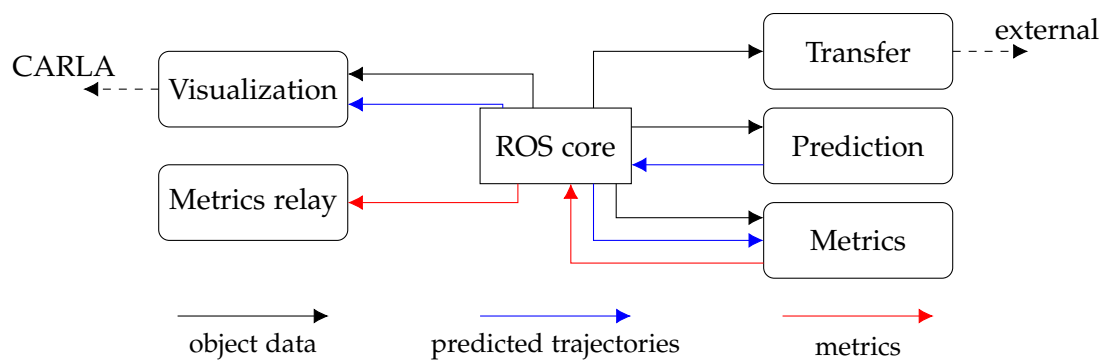


Figure 2: ROS message flow: only framework nodes are shown

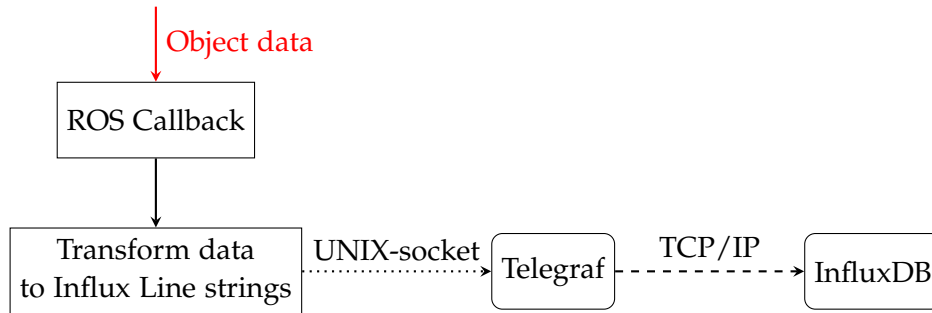


Figure 3: Outgoing data flow

a node which receives the metric messages, formats them and relays the data to the Providentia++ homepage server via HTTP POST, such that they can be displayed online. The overview (Figure 2, page 4) shows the message flow between the ROS core and our nodes, note that individual nodes can run on their own, though the metrics module depends on the prediction module to function in a meaningful way - in turn, the node responsible for relaying metrics needs the metrics module to publish messages.

3.2 Transferring Data To InfluxDB

InfluxDB is a modern time-series database [10] which integrates very well with Python. Data can easily be imported in Pandas DataFrame format using Python, which makes it suitable for machine learning prototyping. *Telegraf* [11] is a ready to use data collection agent with a plethora of input options, built to relay data to an *InfluxDB* instance. We decided to build the data exporting module on top of *Telegraf* and to input data via a UNIX socket using the *InfluxDB Line Protocol* [12] (Figure 3, page 5). This setup is very flexible, since the system running the module only needs a current version of *Telegraf* to work. It also makes the module independent from *InfluxDB* version updates. Additionally, it allows users to easily configure any *InfluxDB* instance as the recipient.

3.3 Integrating PyTorch Models

We decided on supporting the *Torch* machine learning library [13]. In order to use PyTorch models in the toolchain, they need to be traced and stored as a *torchscript* file. This process records all operations the model performs in its forward step. Then, we can use *libtorch*, Torchs C++ component, to import the *torchscript* file and call the models forward function in a C++ environment. Our implementation supports the addition of new models through polymorphism, in particular, each model needs to implement a custom *model interface*, derived from a base class. The model interface handles all necessary state and processing needed for calling the models forward function. This could be downsampling, keeping object position history or state such as LSTM hidden/cell states. In any case, the interface encapsulates all *libtorch* related

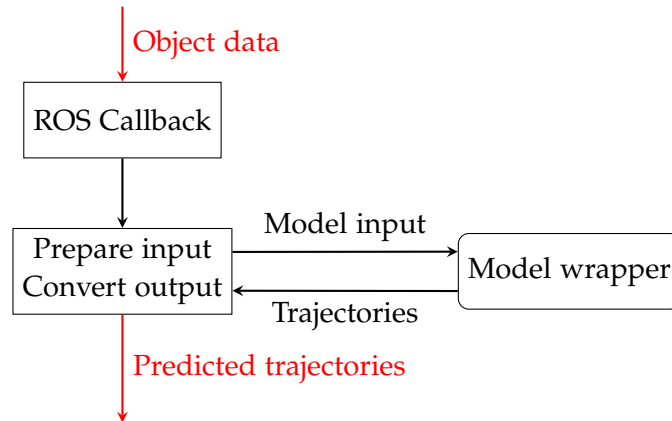


Figure 4: Prediction data flow

objects, such as tensors, and makes sure that the models input has the proper shape. The module passes object data to the model interface on each received message, which is expected to return a set of trajectories. The module then converts these trajectories into a **ROS** message and publishes it (Figure 4, page 6). Models can return zero, one or multiple trajectories per object.

3.4 Visualization Using CARLA

The *CARLA* simulator is a widely used autonomous driving framework based on the unreal engine [4], which lends itself nicely to visualize the digital twin, since all necessary features are already available. The Providentia++ project contains a custom HD map to use with the *CARLA* Simulator (Figure 1, page 2). The toolchain already includes a *CARLA* interface module able to spawn and update vehicles according to the current digital twin state in the *CARLA* simulator. With the predicted trajectories being published as **ROS** messages by the prediction module, we extended this *CARLA* interface to draw predicted trajectories dynamically into the simulation (Figure 5, page 7). The feature can be turned on via command line options and it can also be toggled during runtime using an environment variable. After integrating the framework to the live system, these features can be used to dynamically enable and disable trajectory visualization on the project homepage. A remaining challenge is latency induced by model inference. The simulator is synchronized with the digital twins output, and if trajectory visualization is enabled, the next frame is only drawn once all trajectories are visualized. If the model inference is running slower than the digital twin (25Hz), the model in use can not support real time visualization.

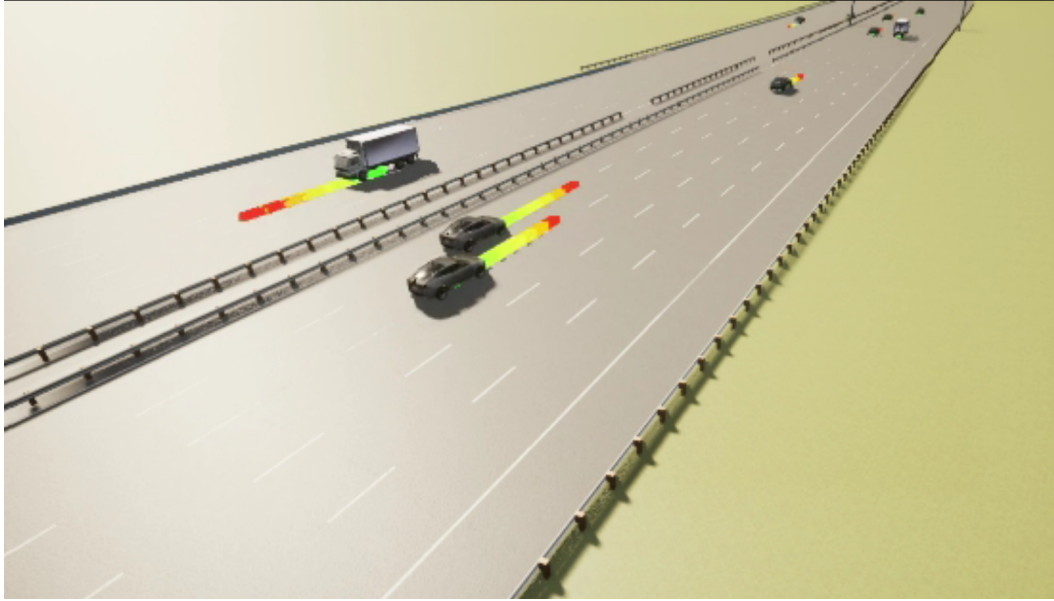


Figure 5: Trajectories visualized using the framework and Carla

3.5 Real Time Validation

The Providentia++ project along with this framework can be a powerful tool to test models in a real world scenario on a continuous stream of unseen data. In order to evaluate model performance, the framework automatically calculates the models total loss w.r.t. each predicted position. This data is published on a *ROS* topic and could optionally displayed on the Providentia++ status page via the Providentia++ statistics module. It can also be exported to an InfluxDB instance via the Telegraf Adapter (Section 3.2, page 5). The validation module works by storing each predicted trajectory for each object, each time new digital twin positions, i.e. the ground truth, is received, deviations from the predicted positions are calculated and a metrics message is published (Figure 7, page 8). Note that there is a full predicted trajectory for each frame, hence the predicted trajectories overlap.

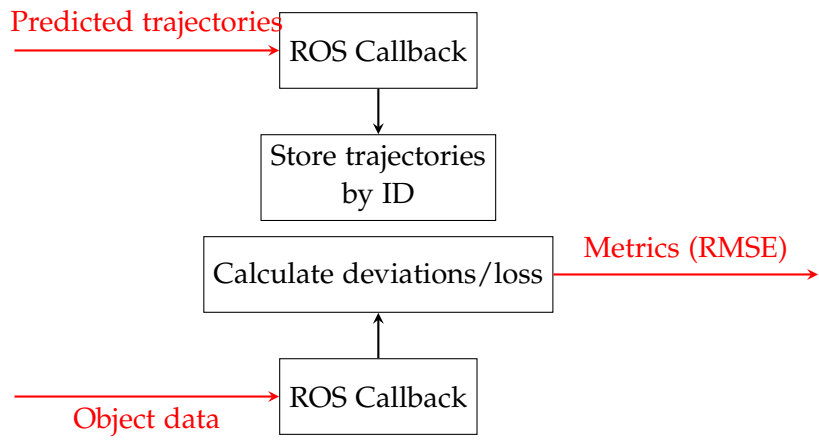


Figure 6: Validation data flow

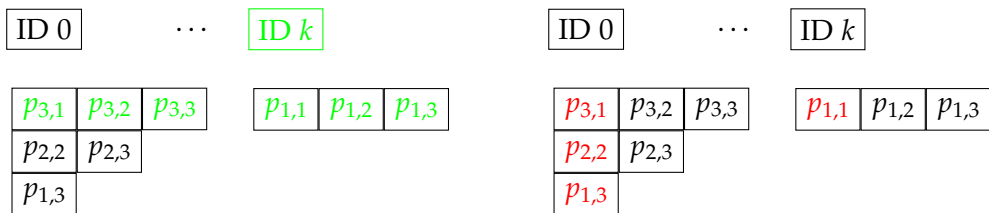


Figure 7: Left: new trajectory message stored. Right: New ground truth position

4 Data Annotation and Preprocessing

Since the Providentia++ object data is not guaranteed to perfectly mirror reality due to inaccuracies in object detection and tracking, some preprocessing steps were performed on the data used for training the baseline LSTM model. Additionally, we manually labelled 60 camera frames as a contribution to the Providentia++ dataset which is yet to be published. Common artifacts in the data include trajectories that are very short as well as physically impossible trajectories. Hence we filter out trajectories that:

- are shorter than a threshold
- contain unreasonable accelerations/jumps
- contain unreasonable turn rates

After this filtering pass, the remaining trajectories are long enough for training sequence to sequence models and, by visual inspection, seem to contain no or very little artifacts from object detection and tracking (Figure 8, page 10).

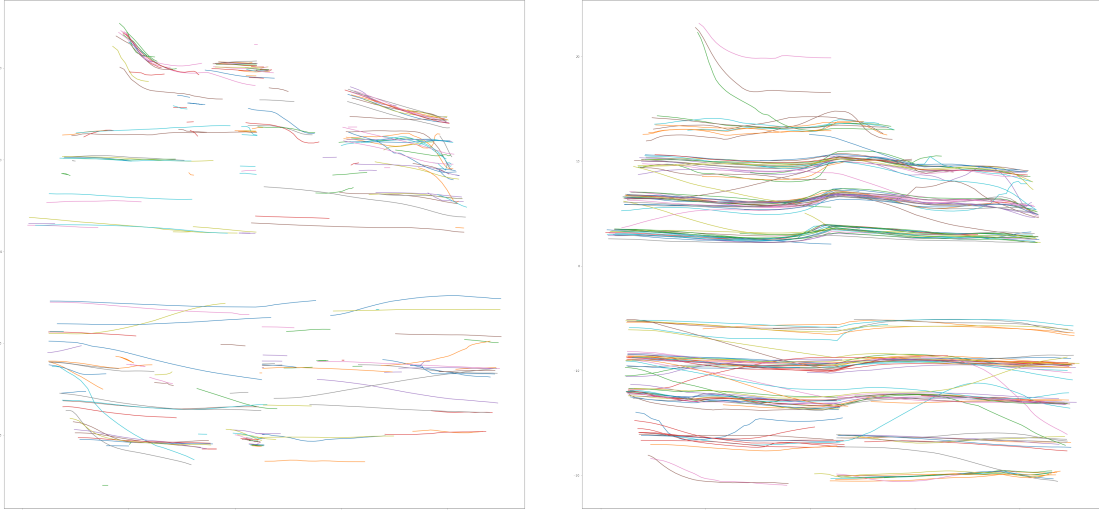


Figure 8: Left: filtered trajectories, right: remaining trajectories

5 Prediction Models

The main focus of this work was creating the framework, but we included a set of models to provide baseline performance metrics and a reference for future model integration tasks.

5.1 Constant Velocity Model

The Constant Velocity Model (CVM) assumes that an object will continue its current trajectory without acceleration of any kind: Let $p : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^3, t \mapsto p(t)$ be the position of the object at time t . Then for some time offset $\delta \in \mathbb{R}^{\geq 0}$ and timestep $k \in \mathbb{N}$ we have:

$$\hat{p}(t + k\delta) = p(t) + k \underbrace{(p(t) - p(t - \delta))}_{\text{spatial difference}} \quad (1)$$

where \hat{p} is the predicted position (Figure 9, page 11). This model works surprisingly well, especially in the highway setting. The high inertia of cars and the straight nature of highway lanes promotes good performance of the constant velocity model. This model is integrated into the toolchain without using libtorch: The model interface saves the last known position for each object we see, on the second encounter the interface returns a predicted trajectory based on the positional difference and updates the last known position for that object.

5.2 Vanilla LSTM

Long-short term memory [14] is the well known extension of the basic recurrent neural network architecture. In order to use LSTM for sequence-to-sequence prediction, a

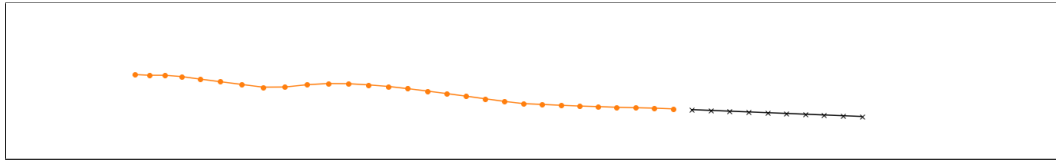


Figure 9: Example CVM prediction - \hat{p} in black

simple method is to feed back predicted values into the model and repeating this process until an output of the desired length is generated. This concept has been used in many different domains and often yields good results. A decent model specifically designed to solve trajectory prediction should be able to outperform this general approach, making it a great baseline model to benchmark against and to give a reference on how future models are to be traced and integrated. The model interface needs to store cell and hidden state per object, manage the model input (batching) and convert from C++ objects to libtorch tensors and vice versa.

5.3 FloMo

The FloMo model [1] tries to approximate the true trajectory distribution by using normalizing flows. It then draws one or more trajectories from this approximated distribution, giving it the ability to output multiple trajectories. FloMo is a state of the art model developed in the context of the Providentia++ project. Its model interface is responsible for keeping track of a history of positions for each object - FloMos input shape consists of eight 2-dimensional positions per object -, to properly format the input as a libtorch tensor, optionally handle CUDA and convert the models tensor output.

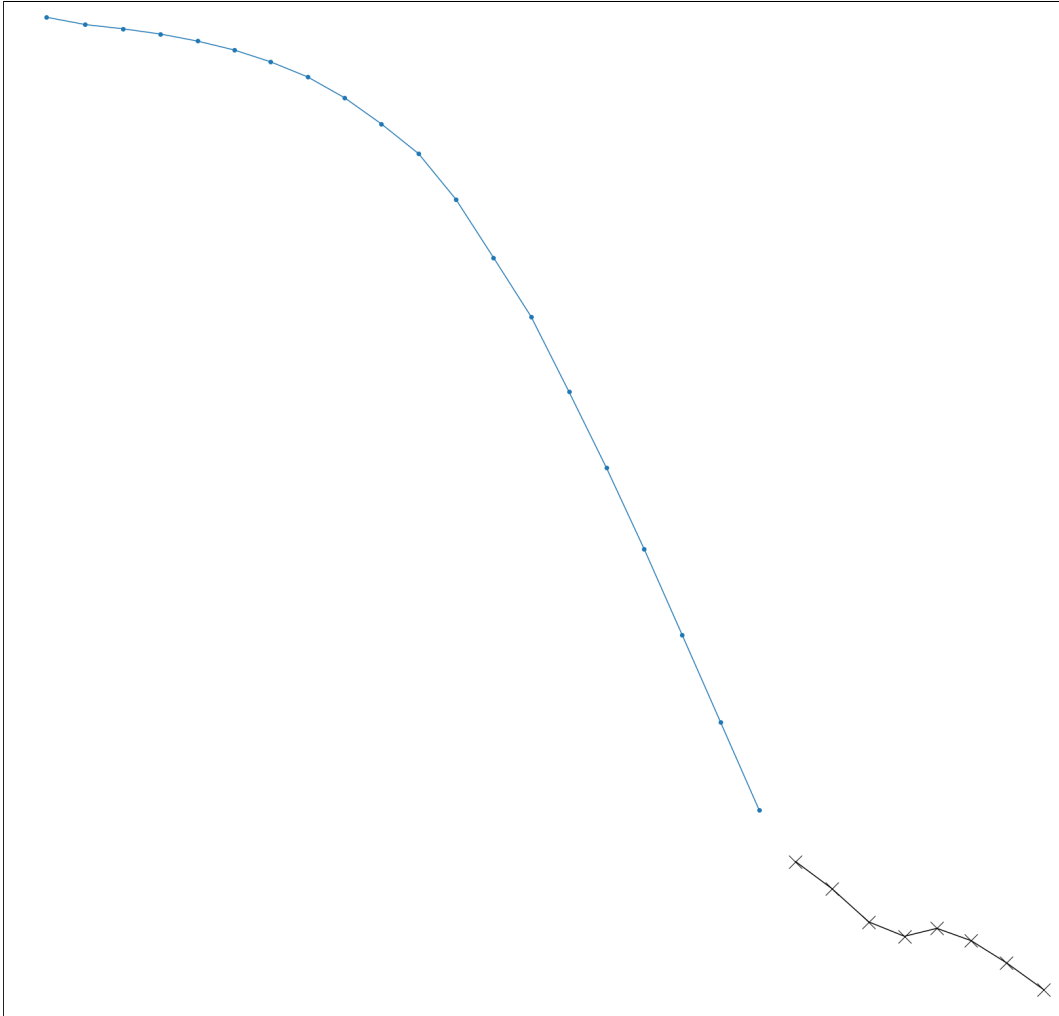


Figure 10: Example LSTM prediction

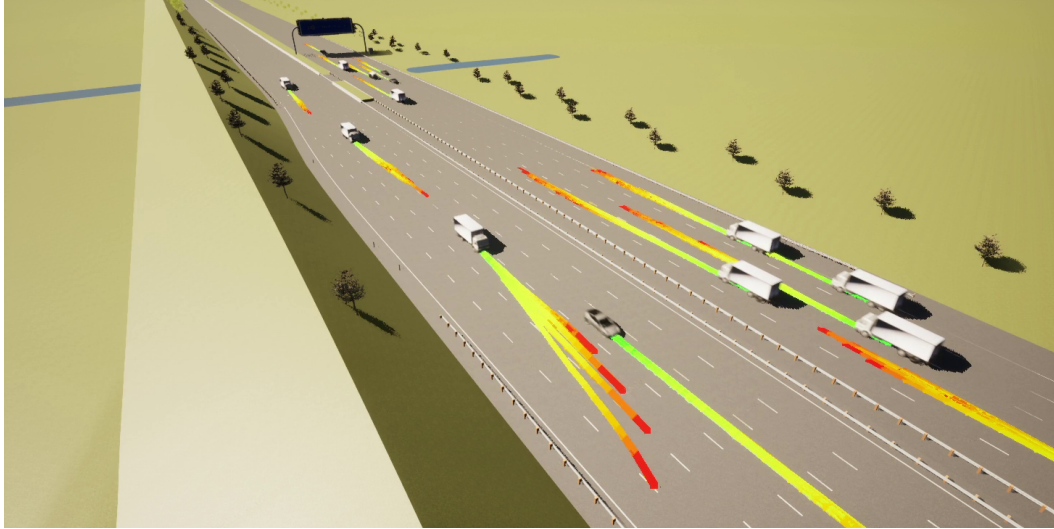


Figure 11: FloMo trajectories visualized using the framework and Carla

6 Outlook

We are confident that this framework is a solid foundation for future work on trajectory prediction within Providentia++. The code is easy to extend and adapt and tries to make it as easy as possible to integrate new models without concerning oneself with implementation details.

6.1 Modelling

Trajectory prediction is an extremely hard task due to the different sources of uncertainty and high variance in possible future positions. It is our belief that the output domain needs to be aggressively restricted by e.g. using a physics based motion model and predicting parameters of this model, incorporating scene data such as lane boundaries or restricting the potential future maneuvers of objects. Some publications already presented approaches based on one or more of these ideas, but motion prediction is still a very open field of research. Another challenge, in particular in the context of Providentia++, is model performance. Since the Providentia++ infrastructure is designed to run at 25Hz frame frequency, a model and the hardware it is running on should optimally be able to complete an iteration before the next frame.

While the short term prediction presented in this work should prove useful to e.g. augment the planning phase of autonomous cars, traffic prediction on a larger scale could hold large potential in the context of cooperative navigation and optimizing traffic flow.

6.2 Robustness

This framework was successfully tested in ‘laboratory conditions’, but it makes some (implicit) assumptions about the ordering and existence of ROS messages; This is especially true for the validation module. Future work and extensions of this framework should consider improving on the robustness of the modules by implementing checks on message order, While the validation and prediction modules work and their results seem valid and as expected, no formal verification against known values was performed.

References

- [1] Christoph Schöller and Alois C. Knoll. “FloMo: Tractable Motion Prediction with Normalizing Flows.” In: *CoRR* abs/2103.03614 (2021). arXiv: 2103.03614. URL: <https://arxiv.org/abs/2103.03614>.
- [2] BMVI. *Providentia++*. URL: <https://www.bmvi.de/SharedDocs/DE/Artikel/DG/AVF-projekte/providentia-plusplus.html>.
- [3] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. URL: <https://www.ros.org>.
- [4] Alexey Dosovitskiy et al. *CARLA: An Open Urban Driving Simulator*. 2017. arXiv: 1711.03938 [cs.LG].
- [5] Andrey Rudenko et al. “Human motion trajectory prediction: A survey.” In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935.
- [6] Nachiket Deo and Mohan M Trivedi. “Convolutional social pooling for vehicle trajectory prediction.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1468–1476.
- [7] Yuejiang Liu, Qi Yan, and Alexandre Alahi. “Social nce: Contrastive learning of socially-aware motion representations.” In: *arXiv preprint arXiv:2012.11717* (2020).
- [8] Karttikeya Mangalam et al. “From Goals, Waypoints & Paths To Long Term Human Trajectory Forecasting.” In: *arXiv preprint arXiv:2012.01526* (2020).
- [9] Isabela Erdelean et al. “Catalogue of connected and automated driving test sites: Deliverable No2. 1.” In: (2019).
- [10] influxdata. *InfluxDB*. URL: <https://www.influxdata.com/>.
- [11] influxdata. *Telegraf*. URL: <https://www.influxdata.com/time-series-platform/telegraf/>.
- [12] influxdata. *InfluxDB Line Protocol*. URL: <https://docs.influxdata.com/influxdb/cloud/reference/syntax/line-protocol/>.

- [13] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [14] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [15] Chuhua Wang et al. “Stepwise Goal-Driven Networks for Trajectory Prediction.” In: *arXiv preprint arXiv:2103.14107* (2021).