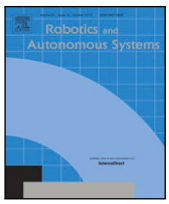
Robotics and Autonomous Systems 172 (2024) 104585

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



RobustStateNet: Robust ego vehicle state estimation for Autonomous Driving

Pragyan Dahal [a,*], Simone Mentasti [b], Luca Paparusso [a], Stefano Arrigoni [a], Francesco Braghin [a]

[a] Department of Mechanical Engineering, Politecnico di Milano, Italy
[b] Department of Electronics, Information Engineering and Bioengineering, Politecnico di Milano, Italy

ARTICLE INFO

Keywords:
Ego motion prediction
Ego state estimation
Sensor fusion
Robust state estimation
Autonomous driving
Sensors failure

ABSTRACT

Control of an ego vehicle for Autonomous Driving (AD) requires an accurate definition of its state. Implementation of various model-based Kalman Filtering (KF) techniques for state estimation is prevalent in the literature. These algorithms use measurements from IMU and input signals from steering and wheel encoders for motion prediction with physics-based models, and a Global Navigation Satellite System(GNSS) for global localization. Such methods are widely investigated and majorly focus on increasing the accuracy of the estimation. Ego motion prediction in these approaches does not model the sensor failure modes and assumes completely known dynamics with motion and measurement model noises. In this work, we propose a novel Recurrent Neural Network (RNN) based motion predictor that parallelly models the sensor measurement dynamics and selectively fuses the features to increase the robustness of prediction, in particular in scenarios where we witness sensor failures. This motion predictor is integrated into a KF-like framework, RobustStateNet that takes a global position from the GNSS sensor and updates the predicted state. We demonstrate that the proposed state estimation routine outperforms the Model-Based KF and KalmanNet architecture in terms of estimation accuracy and robustness. The proposed algorithms are validated in the modified NuScenes CAN bus dataset, designed to simulate various types of sensor failures.

1. Introduction

Ego Vehicle state estimation is the first step in an Autonomous Driving software application. For the safe implementation of the tasks further down the pipeline, such as motion planning and control, it is crucial for the estimated ego vehicle state to be accurate. Ego motion estimation and ego state estimation are two different tasks performed using different sensors. Global Navigation Satellite System (GNSS), Inertial Measurement Unit (IMU), Wheel Encoders, Lidar, Camera, etc., are combined to build the Autonomous Driving sensor suite. The majority of works in literature use exteroceptive sensors like Camera, Lidar, or proprioceptive sensors like IMU for ego-motion estimation. In combination with this, sensors like GNSS in standalone mode or Lidar in Simultaneous Localization and Mapping (SLAM) [1] mode are popular for state estimation or localization. Works with camera relocalization [2] are also getting traction in this field. Ego vehicle state estimators based on Kalman Filters utilizing the vehicle dynamics for motion prediction and GNSS for state update are significantly common in literature [3]. These traditionally developed filtering estimators are prone to inaccuracies due to sensor failures and GNSS measurement unavailability in urban scenarios.

Estimation algorithms implement sensor fusion with a different permutation of the aforementioned sensors. The motivation for sensor fusion in such algorithms is to increase state observability by exploiting different sensors' complementary nature and introducing redundancy into the estimation algorithms. Deep learning-based motion estimators, Visual Odometry (VO) solvers, Visual Inertial Odometry solvers (VIO) [4], etc., perform end-to-end motion estimation by directly fusing the sensor features and compute estimates through complex learned mapping functions. These fusion strategies, developed without explicitly modeling the error sources in sensors, can be prone to robustness issues [5]. Some recent works, [5,6], study the involvement of IMU and Camera failures and propose a fusion strategy to weight features to robustly estimate ego vehicle motion.

While the majority of works in the literature focus on increasing the accuracy of the estimation routine, be it motion or state estimation, very little attention is given to increasing the robustness to the sensor failure. In this work, we focus on the robustness and accuracy of estimation as a single problem. We study various failure scenarios of proprioceptive sensors like IMU, wheel encoders, and steering encoders. We propose a novel deep learning-based state estimator that utilizes a Recurrent Neural Network (RNN) for motion prediction and state update and is structured in a similar fashion to the Kalman Filter. The overall architecture of the proposed model is illustrated in Fig. 1. A comparative study is performed between the proposed algorithm,

* Corresponding author.
E-mail address: pragyan.dahal@polimi.it (P. Dahal).

https://doi.org/10.1016/j.robot.2023.104585

Available online 7 December 2023

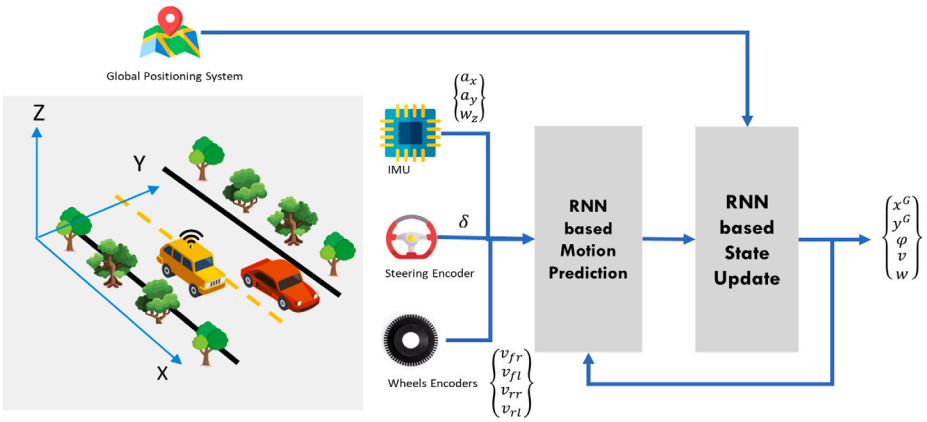


Fig. 1. Overall architecture of the proposed ego vehicle state estimator.

the baseline model-based Kalman Filter, and the deep learning-based filtering framework KalmanNet [7]. The algorithm is validated on the modified NuScenes Dataset. Sensor failures are introduced into the pre-existing data and simulated noise into the measurements to validate the algorithm. The contribution of the paper is three-fold:

- We propose a novel motion predictor for the ego vehicle state estimation utilizing Recurrent Neural Networks. We demonstrate that the predictor is robust to sensor measurement and input failures and provides reliable state prediction.
- The proposed motion predictor is integrated into a novel deep learning-based update framework, which consistently provides ego vehicle state estimation, even when multiple sensors fail.
- We create a new challenging dataset by incorporating sensor failure in the existing NuScene dataset. The code to generate the failures will be released upon paper acceptance.

The remainder of the article is organized in the following fashion. Section 2 presents a survey of works in the literature related to ego-motion and state estimation. In Section 3, we explain the problem statement the paper is trying to solve. Section 4.1 details the proposed motion prediction architecture while 4.3 explains the architecture of the proposed RobustStateNet. Section 4.5 provides the details regarding the training of the proposed models. In Section 5, we discuss the experimental setup and validate the proposed models. Section 7 concludes the paper.

2. Related works

2.1. Ego motion estimation

Ego motion estimation can be categorized into either odometry computation using the fusion of proprioceptive sensors like IMU and exteroceptive sensors like Camera and Lidar or into state computation using vehicle motion models. Visual Odometry (VO) [8,9], Visual Inertial Odometry(VIO) [4–6,10], and Visual, Inertial, and Lidar Odometry (VILO) generally compute the relative pose between two consecutive time frames and compute the trajectory of the ego-motion by accumulating them. However, ego-motion computation using predefined models uses the knowledge of vehicle kinematics or vehicle dynamics. The motion prediction in this work is focused on computing the predicted state at each time instance, which comprises positional values in two dimensions (x,y, and yaw angle) along with their rate of change, hence resembling state computation procedures with the model-based methods. In many of the physics-based models, for example, single-track kinematic models, double-track models, and even models based on vehicle dynamics can fail in cases of sensor measurement degradation or failure. Strictly model-based implementations use signal-filtering techniques to filter out possible noise and outliers. Authors in [11] use a kinematic bicycle vehicle model to perform a single-step prediction, and this motion model is used in a KF framework to provide vehicle state estimates. However, no explicit modeling of the possible sensor failures or measurement degradation is done in the algorithm development. Implementation of these physics-based models relies heavily on the accuracy of the vehicle parameters which can be cumbersome to compute.

With the recent advancements in machine learning, learning-based techniques are widely adopted to study various aspects of vehicle dynamics. Authors in [12] use a simple Fully Connected Network (FCN) to estimate the vehicle side slip angle. A Long Short-Term Memory (LSTM) is used to estimate the vehicle lateral velocity in [13]. The authors use the sensor data collected in a simulation environment as input to this RNN model which regresses a vehicle lateral velocity as output. [14] use three different Nonlinear AutoRegressive Neural Networks with exogenous input (NARX) to estimate the vehicle side slip angle for three different road conditions. The authors then use a pattern recognition classifier to select the output corresponding to correct road conditions. While these works are focused on estimating one single parameter or an aspect of vehicle dynamics, learning a complete vehicle motion model through data is also taking traction. Authors in [15] integrate a vehicle dynamic model based on the FCNs into a feed-forward feedback control. They demonstrate that with such integration of learned dynamic models into the control setup, the controller is able to provide comparable results to the experienced human driver at the limits of the vehicle's capabilities. In [16], authors propose a Model Predictive Controller (MPC) that replaces the traditional vehicle motion models with a learned NN-based dynamic model. The input to the model is the history of past system states and vehicle inputs. In [17], authors propose a learning-based vehicle dynamics model based on Gated Recurrent Unit (GRU). The network predicts the state at the next time instance, as it takes as input the state of past instances and other system values like wheel torque and brake pressure.

2.2. Ego vehicle state estimation

Global localization of the ego vehicle can be done in multiple ways. Using exteroceptive sensors, Camera-based relocalization [2], and Lidar-based SLAM algorithms [1,18] are prevalent in the literature. A more simplistic approach would be to utilize the global positioning provided by a GNSS sensor or multiple GNSS sensors attached to the vehicle. In this setting, a variant of KF, Extended KF (EKF), or Unscented KF (UKF) is employed to provide consistent state estimates. A comparative study of these algorithms is presented in [3]. While these algorithms are efficient to implement, they require explicit modeling of motion model noise and measurement noise to be used in the development of the Kalman Filter. They are also prone to errors in cases of sensor failure and measurement degradation.

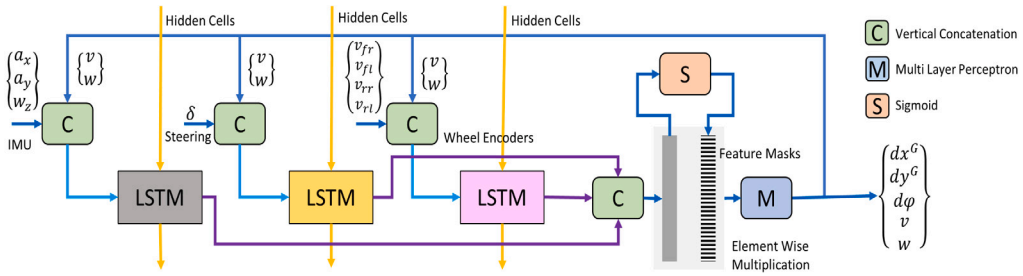


Fig. 2. Ego vehicle motion prediction architecture.

However, some recent works have started to address the issue of partial domain knowledge, i.e. develop state estimation algorithms without the knowledge of system noise parameters. Authors in [7] propose KalmanNet, two RNN-based state estimation models with an underlying assumption that the system motion model and measurement models are known. Even though the vehicle dynamics models are widely investigated, they are prone to failures in sensor failure scenarios. These assumptions could be further investigated, which is done in our proposed algorithm. In [19], the motion model and the noise parameters are learned from the data while the update step takes the exact form of the Model-Based KF. LSTMs and Transformers are used to estimate the parameters in [20], where authors demonstrate the superior performance of these models compared to the Expectation Maximization Kalman Filter (EM-KF).

The mentioned works in the literature, for both motion and state estimation, develop learning-based models completely focusing on increasing the accuracy of estimation. They are trained to utilize fully functional sensor measurements and are not equipped to deal with sensor failure or measurement degradation scenarios. Furthermore, the sensor measurements are concatenated and directly fed as input to the models, hence not utilizing the possible different dynamics of these sensor measurements and increasing the ability to have redundancy. Hence, in this work, we explore the possibility of modeling sensor dynamics in parallel and choosing the relevant features in a selective manner. We train our model with a dataset that includes sensor degradation and failure scenarios making it robust to such failures.

3. Problem statement

For motion prediction and control of the ego vehicle, its state at any given time must be well defined. In this work, we model the ego vehicle motion and state in two dimensions, *(x,y)*. The state at time instance k is represented as :

$$x_{k|k} = \left[x^G, y^G, \psi, v, w\right]_k \tag{1}$$

where, x^G and y^G are longitudinal and lateral positions in Global Reference Frame (G), ψ is yaw angle, v is longitudinal velocity and w is yaw rate. The reference point for the ego vehicle state estimation is taken at the center point of the vehicle's rear axle.

At the given time instance k, measurements from IMU, wheel encoders, and steering encoders are received. We assume a proper synchronization between sensors in the development of this algorithm. Longitudinal acceleration, a_x, lateral acceleration a_y, and yaw rate w_z are obtained from IMU. Steering angle δ is obtained from the steering encoder and wheel speeds from the four wheel encoders. The GNSS sensor provides global positioning along with the velocities in the Global Reference Frame. For the position of the different sensors and the reference frames, please refer to the NuScenes dataset [21]. Given the measurement from the IMU, wheel encoders, steering encoders and GNSS at time instance k, the task of the algorithm is to estimate ego vehicle state $x_{k|k}$.

4. RobustStateNet

RobustStateNet's architecture is designed to resemble that of the model-based Kalman Filter, hence comprising of two major components, motion prediction and state update. We implement RNN based motion prediction and update steps which are explained subsequently in Sections 4.1 and 4.3.

4.1. Ego vehicle motion prediction

In a traditional KF setting for ego vehicle state estimation, vehicle kinematics or dynamics are generally used to develop the motion model. They also require information regarding various model parameters, which at times can be difficult to attain. These motion models, even though highly accurate, are not robust to sensor failure and are very prone to wrong estimations. They fail in cases of missing measurements, sensor misalignment, or disconnections. With this in mind, we propose a motion predictor that can provide consistent and accurate state prediction even in cases of sensor failures or misreadings. Fig. 2 illustrates the architecture of this prediction model. We employ three parallel stateful LSTMs for the input from IMU, steering encoder, and wheels encoders as demonstrated in the figure.

The measurements obtained from the sensors are vertically concatenated with the velocity and yaw estimate of the previous time step, $k-1$. The inputs to these LSTM are represented in Eq. (2).

$$\begin{aligned} x_I^k &= \left[v, \psi, a_x, a_y, w_z\right]_k \\ x_S^k &= \left[v, \psi, \delta\right]_k \\ x_W^k &= \left[v, \psi, v_{fl}, v_{fr}, v_{rl}, v_{rr}\right]_k \end{aligned} \tag{2}$$

where, x_I^k, x_S^k and x_W^K are input to the IMU LSTM $\mathcal{N}_I$, Steering LSTM $\mathcal{N}_S$ and Wheel LSTM $\mathcal{N}_W$ respectively. These LSTMs model the temporal dependencies of the features or the sensor measurements through the progressive update of the hidden cell units, h_{Sensor}^{k-1} as well.

$$\begin{aligned} y_I^k, h_I^k &= \mathcal{N}_I(x_I^k, h_I^{k-1}) \\ y_S^k, h_S^k &= \mathcal{N}_S(x_S^k, h_S^{k-1}) \\ y_W^k, h_W^k &= \mathcal{N}_W(x_W^k, h_W^{k-1}) \end{aligned} \tag{3}$$

h_I^{k-1}, h_S^{k-1} and h_W^{k-1} are the hidden cell units of last time instance for the IMU, Steering, and Wheel Encoder LSTMs respectively, while y_I^k and h_I^k are the output feature and updated hidden unit of the IMU LSTM.(S and W subscript for the variables related to Steering and Wheel Encoder respectively.) The outputs of these LSTMs are vertically concatenated and fed to a feature selection layer, $\mathcal{N}_m$ to compute a mask for the selective fusion of the computed features. We investigate two different kinds of feature selection mechanisms, proposed by [2,5], for our implementation.

$$m_k = \mathcal{N}_m([y_I^k, y_S^k, y_W^k]) \tag{4}$$

The concatenated features, $[y_I^k, y_S^k, y_W^k]$ are represented by a gray box before $\mathcal{M}_m$ in Fig. 2. They are element-wise multiplied with the computed mask, m_k, and passed through a regressor, $\mathcal{N}_p$ based on

multilayer perceptions (MLPs), to compute the predicted motion of the ego vehicle. The motion predictor that computes the mask with Soft Masking is labeled $LSTM0$ while the one with Hard Mask is labeled $LSTM1$. Soft Mask computes a deterministic continuous value based on the Sigmoid layer while Hard Mask turns on and off different features coming from different sensors. This is done in a stochastic fashion. Readers are referred to [5] for a clear explanation of these masking techniques.

$$dx_k = \mathcal{N}_p([[y_I^k, y_S^k, y_W^k] * m_k]) \tag{5}$$

The output of the prediction model, dx_k is represented as :

$$dx_k = \left[dx_p^E, dy_p^E, d\phi_p, v_p, w_p\right]_k \tag{6}$$

The model predicts the change of the longitudinal dx_p^E and lateral position dy_p^E, of the ego vehicle in the Vehicle Reference Frame (E) and the change in the yaw angle $d\psi_p$. The motion is later transformed into the Global Reference Frame(G) to compute the predicted state.

$$\begin{aligned} dx_p^G &= dx_p^E * \cos\psi_p - dy_p^E * \sin\psi_p \\ dy_p^G &= dx_p^E * \sin\psi_p + dy_p^E * \cos\psi_p \end{aligned} \tag{7}$$

While the vehicle velocity v and yaw rate w are computed as a whole. Hence, from the model output, the predicted state of the ego vehicle, $x_{k|k-1}$ can be computed as:

$$\begin{aligned} x_{k|k-1}[1:3] = x_{k-1|k-1}[1:3] + \left[dx_p^G, dy_p^G, d\psi_p\right]_k \\ x_{k|k-1}[4:5] = \left[v_p, w_p\right]_k \end{aligned} \tag{8}$$

4.2. Baseline for motion prediction

We develop a Kalman Filter-based baseline for the comparative study. The model is based on the kinematic vehicle model and an Extended Kalman Filter (EKF) estimator. We use the same state representation as the RobustStateNet prediction model, $x_{k|k} = \left[x^G, y^G, \psi, v, w\right]_k$, which contains the vehicle position x^G, y^G, yaw ψ, velocity v and the yaw rate w values in the **G** frame. For developing the Kalman Filtering recursion, longitudinal acceleration, a_x, is used as input to the motion model, while the averaged value of the wheel encoder speeds, v_w and yaw rate from the IMU, w_z are used as a measurement. The system of equations for the baseline are Eqs. (9) and (10).

$$x_k = x_{k-1} + f_{k-1}(x_{k-1}, u_k, w_k)\delta t \tag{9}$$

$$y_k = Hx_k + v_k \tag{10}$$

The motion model is

$$f_{k-1} = \begin{cases} \dot{x}^G = v.cos\psi \\ \dot{y}^G = v.sin\psi \\ \dot{\psi} = \omega_z \\ \dot{v} = a_x \end{cases} \tag{11}$$

The input and the measurements are filtered using the Infinite Impulse Response (IIR) filter to remove the noises and failure values introduced while simulating the sensor measurement degradation. This is done to ensure a fair comparison with the LSTM-based predictor.

4.3. Ego vehicle state estimation: RobustStateNet

The RobustStateNet is based on the architecture of the standard Kalman Filter and utilizes data-based model development for motion model and Kalman Gain computation. The algorithm is developed with an assumption that the motion model $f(x)$ is not known while the measurement model $h(x)$ is known. We assume this to be the standard scenario in the Ego Vehicle State estimation when considering the proprioceptive sensors' failure or measurement degradation. The overall architecture of the RobustStateNet is illustrated in Fig. 3. It can be seen in the figure that the RobustStateNet consists of two steps to compute the state estimation at a given time instance k, namely, prediction and update.

Prediction The Prediction step of the RobustStateNet assumes that the dynamics of the vehicle motion is not known and uses the RNN-based prediction model described in Section 4.1. The velocity and yaw angle component of the estimated state of the previous time instance, $k-1$ are concatenated with the measurements from the IMU, steering, and wheel encoders and provided as input to the prediction model. Contrary to the model-based Kalman Filters, RobustStateNet does not compute the second-order statistical moment, i.e. the Covariance matrix. Hence, the assumptions related to noise distribution are not necessary for the development of the filtering recursion.

Update For the update of the predicted state, global positional and velocity information is assumed to be available from the GNSS sensor. Yaw angle and yaw rate from the IMU sensor are also used as measurement sources. The GNSS measurements are generated from the ground truth position values provided in the NuScenes dataset in the absence of the true GNSS data. We simulate the GNSS noise using the Circular Error Probable of 2 m, which indicates that 50% of the measurements are not further than 2 m from the correct ego position. By adding such noise, we simulate the use of cheap GNSS sensors with very low accuracy. The measurements from the GNSS and IMU sensors are used to create the measurement input, z_k.

$$z_k = \left[x^G, y^G, \psi, v, w\right]_k \tag{12}$$

The architecture of the update module is illustrated by Fig. 3. Three different kinds of features are taken as input to the Gated Recurrent Unit(GRU) whose output is the Kalman gain for the state update. The features are:

- State Prediction Difference, $\delta x = x_{k|k-1} - x_{k-1|k-1}$
- Measurement Innovation, $\delta z_p = z_k - Hx_{k|k-1}$
- Measurement Difference, $\delta z = z_k - z_{k-1}$

These features are weighted using a mask computed from a Sigmoid layer with an MLP, $\mathcal{N}_m$ as shown in Eq. (13). They provide appropriate weighting and increase the robustness to the failures introduced into the motion model and noise in the measurement model. The mask is elementwise multiplied to the vertically concatenated features and fed into the GRU.

$$m_k = \mathcal{N}_m([\delta x, \delta z_p, \delta z]) \tag{13}$$

$$f_k = m_k * [\delta x, \delta z_p, \delta z] \tag{14}$$

The Kalman Gain is computed as a function of these weighted features, f_k, and the hidden state of the GRU of the last time instance, h_{k-1}. The output of the GRU is then passed through an MLP to compute the Kalman Gain.

$$K = \mathcal{N}_g(GRU(f_k, h_{k-1})) \tag{15}$$

where, $\mathcal{N}_g$ is a MLP. The update step does not explicitly model the noise covariance regarding the vehicle motion and the measurements. Similarly to [7], we assume that the hidden state of the GRU keeps track of the second-order statistical moment. Computed KG is then transformed into a matrix, K, and is used to update the predicted state from the LSTM-based motion model.

$$x_{k|k} = x_{k|k-1} + K(z_k - Hx_{k|k-1}) \tag{16}$$

The training and inference of the RobustStateNet will be discussed in Section 4.5.

4.4. Baseline for the state estimation

We compare our proposed RobustStateNet against two baselines. First, we develop a Model-Based (MB) EKF, with known measurement noise parameters, i.e. the ground truth noise parameters are used to

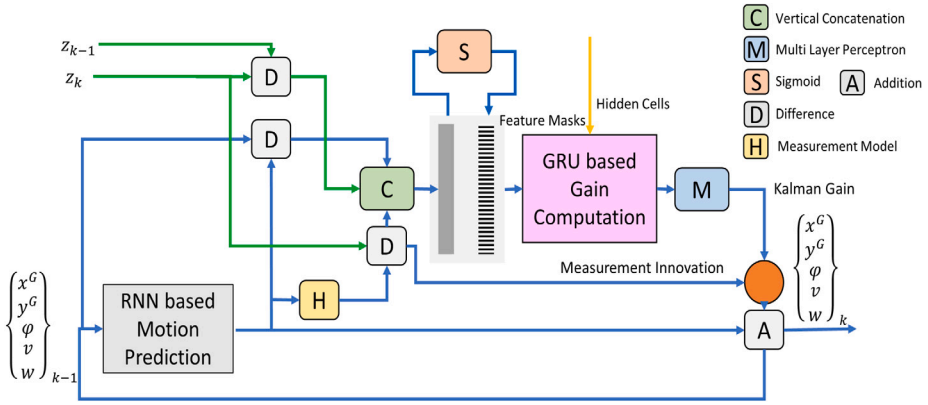


Fig. 3. Overall architecture of the proposed RobustStateNet estimator.

develop the filtering recursion. The motion model of the EKF is based on the kinematic vehicle model, Eq. (11) while the measurement model is a linear transformation from state space to measurement space. Second, we train the first version of KalmanNet, [7] with our own dataset. The KalmanNet assumes a known vehicle motion model and measurement model. We use the kinematic model as the motion model in the framework.

4.5. Training models

4.5.1. Training LSTM based motion predictor

LSTM motion predictor is trained on the modified NuScenes dataset in a supervised manner. The data collected on the vehicle CAN bus from the IMU sensor, steering encoder and wheel encoders are used to create the dataset. Various sensor failure modes are added to the raw sensor data acquired from the CAN bus to simulate sensor failure. The data degradation for different sensors is similar to the cases introduced by [5]. For IMU data degradation, we introduce three different failures in the raw data:

- All the sensor readings are zeroed
- Addition of random noise to the readings to simulate the sensor degradation due to moisture and wear
- Sensor misalignment due to loose setup. This is done by introducing gradually increasing rotation between the sensor reference frame and the vehicle reference frame.

We employ the first two failure modes for the steering and wheel encoder failures. NuScenes provides 1000 scenes, of which, 900 are used for model training while the remaining 100 are for validation and testing. The dataset is divided into N number of samples, each consisting of mini trajectories of length H corresponding to the vehicle motion. Training dataset can be represented by $S = \{(Y_i, X_i)\}_1^N$, where each sample is represented as:

$$\begin{aligned} Y_i &= \left[y_1^i, y_2^i \ldots . y_H^i\right] \\ X_i &= \left[x_1^i, x_2^i \ldots . x_H^i\right] \end{aligned} \tag{17}$$

x_t^i comprises inputs for IMU LSTM, Steering LSTM, and Wheel encoder LSTM for sample i and time instance t. As shown by Eq. (2), each input values require longitudinal velocity, v, and yaw angle, ψ. In this implementation, we provide ground truth values for the velocity and yaw angle at each time instance of the sample. The output for time instance t in the trajectory of sample i, y_t^i is given by Eq. (6). Each LSTM consists of 128 hidden units, while the MLP layer is made of two linear layers, (128–64, 64–5) connected through a ReLU non-linearity.

Loss Function: A weighted mean squared error between the target values $\hat{y}_t$ and the predicted value y_t is computed to optimize the model parameters θ.

$$\begin{aligned} \mathcal{L}(\theta) = \lambda_1 \|\hat{p} - p\|_2 + \lambda_2 \|\hat{\psi} - \psi\|_2 \\ + \lambda_3 \|\hat{v} - v\|_2 + \lambda_4 \|\hat{w} - w\|_2 \end{aligned} \tag{18}$$

where, λ_1, λ_2, λ_3, and λ_4 are scaling values for the different prediction components regarding the position p, yaw ψ, velocity v, and yaw rate w. In our experiments, we assume values of λ_1, λ_2, λ_3, and λ_4 to be 1, 10000, 1 and 100 respectively. The choice of these scaling parameters is based on the dimensionality of the various state components. Notably, we assign a higher value to the yaw scale factor, λ_2, due to its lower value range compared to the other components. In such a way, the values are all scaled to similar ranges, and the network learns different tasks simultaneously. The motion predictor is trained using Back Propagation Through Time (BPTT). Each scene from NuScenes is divided into multiple samples consisting of mini-trajectories of fixed length, $H = 30$. Then, they are shuffled to create the training dataset. Adam, with a constant learning rate of $1e{-}4$ is used to optimize the model parameters. The model is trained for 40 epochs.

4.5.2. Training full RobustStateNet

RobustStateNet is also trained on the modified CAN bus data of the NuScenes Dataset. As illustrated in Fig. 3, the input to the model at any given time instance is the measurements from the IMU, steering encoder, wheel encoders, and the noisy GNSS sensor. For the motion prediction component of the model, we start the training with a pre-trained model. Within RobustStateNet, the motion prediction and state update are jointly trained, hence the model update section is able to consider the errors in the motion prediction while increasing its accuracy. The 1000 scenes from NuScenes are divided into 900 training sets and 100 validation and testing sets. From the training scenes, we create N samples, where each sample consists of a mini-trajectory of the vehicle motion. Similarly to the prediction training, a sample can be represented by Eq. (17). x_1^i in this case, in addition to the IMU, steering, and wheel encoder values, comprises the noisy GNSS data of time instance k and $k-1$. y_1^i is the ground truth state position of the ego vehicle at the first instance of the ith trajectory.

Loss Function A mean square error between the updated state y_u, and predicted state y_p and the target state value $\hat{y}$, given by Eq. (19) is used to optimize the model parameters θ.

$$\mathcal{L}(\theta) = \lambda_1 \|\hat{y} - y_u\|_2 + \lambda_2 \|\hat{y} - y_p\|_2 \tag{19}$$

where, λ_1, and λ_2 are scaling factors for the prediction and update loss. Since the prediction model is based on a pre-trained model, we attribute greater significance to the update model. To reflect this emphasis, we assign the values of 0.8 to λ_1 and 0.2 to λ_2. RobustStateNet is also developed using the Pytorch deep learning framework. Adam optimizer, [22] with a learning rate of 1e−5 is used to train the model.

5. Experiments and results

We perform tests in the NuScenes CAN bus data. Additional noises and sensor degradation are added to the existing sensor measurements

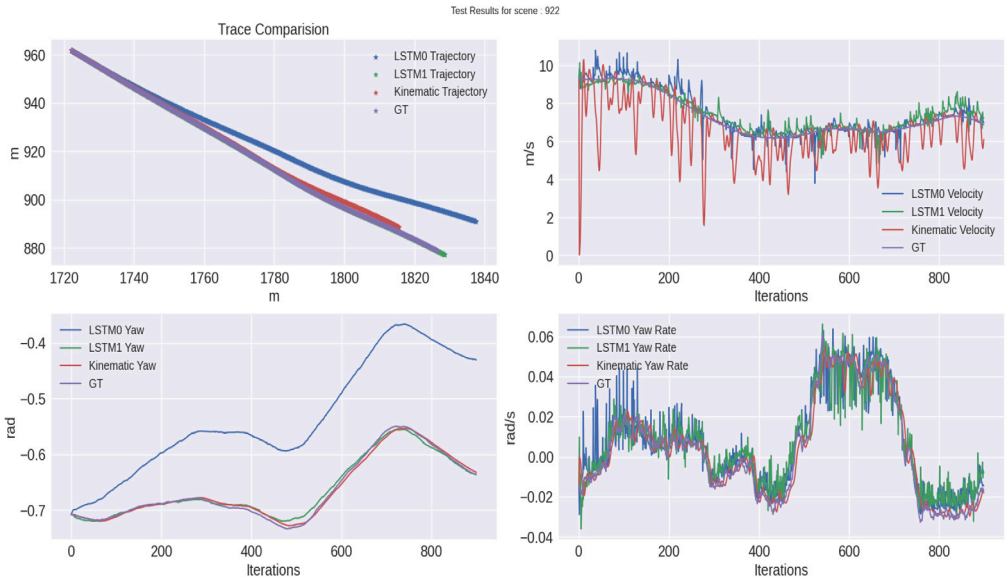


Fig. 4. Comparison of the output of the LSTM-based predictors with the kinematic model for Scene 922 of the Nuscenes Dataset.

to simulate sensor failures for validating the proposed models. Both the LSTM-based motion predictor and the RobustStateNet are developed using the Pytorch [23] deep learning framework. Adam is used to optimize the model parameters. The models are trained in a consumer laptop with Nvidia GeForce RTX 3060 GPU. The models are trained on the first 900 scenes of the NuScenes dataset while the scenes from 900 to 1000 are used to validate and test the models.

To analyze the performance of the proposed algorithms, we use the Mean Squared Error (MSE) metric between the predicted or estimated and the Ground Truth States similar to [7]. The metric can be defined in decibel as :

$$MSE = 10log_{10}(\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2) \tag{20}$$

where, N represents the trajectory length and y_i is the predicted or estimated state at instance i of the trajectory and $\hat{y}_i$ is the ground truth state at the instance.

5.1. Motion prediction

Since all the sensor measurements are available at 50 Hz, the motion prediction algorithm also runs at 50 Hz. The motion prediction is conducted in a recursive manner for both the LSTM-based predictor and the baseline. The ground truth global positioning at the first instance is used to initialize the motion predictor which computes the ego trajectory recursively afterward. The EKF constructed as a baseline is modeled with the actual noise values to obtain the best possible outcome. The qualitative results of the motion prediction for the scene 922 are reported in Fig. 4. The experimental scene consists of 900 iterations, i.e. temporal length amounting to the 18-second period. We can observe that the LSTM-based predictor is able to provide accurate estimates in terms of the vehicle state prediction, the position trajectory, velocity, yaw angles, and yaw rates. The input to the LSTM-based predictor is extremely noisy and consists of multiple failure scenarios generated randomly. However, input to the baseline predictor based on the kinematic model is filtered using an IIR filter to reduce the effects of the introduced noise and failures. This qualitative observation also demonstrates that the LSTM-based predictor outperforms the kinematic model in accuracy and robustness.

Fig. 5 illustrates the Mean Square Errors (MSE) in dB computed using Eq. (20) for the two different LSTM-based predictors and the kinematic model for different scenes in the NuScenes dataset. Table 1 reports the MSE result for the scene 922 computed for 100 Monte Carlo runs for the LSTM0, LSTM1, and the kinematic model. It can be observed that the accuracy of the prediction from LSTM models is higher than the kinematic model. LSTM1 model, with the stochastic hard fusion masking, is demonstrated to be most robust to sensor failures and provides the best result in terms of state prediction.

Table 1
Averaged MSE in dB for 100 MC simulation runs for the data of scene number 922 of the NuScene Dataset (Lower values indicate better performances.).

Mode	LSTM0	LSTM1	Kinematic
Mean MSE (dB)	11.27	**−5.30**	11.61
Std Deviation (dB)	0.43	**1.35**	0.88

5.2. RobustStateNet

In real driving scenarios, assumed cheap GNSSs do not provide the measurements at 50 Hz, hence to simulate a realistic setting, we adopt a multi-prediction, single-update framework. In this training approach, the GNSS data is made available at a frequency of 10 Hz. The motion prediction algorithm performs prediction at 50 Hz while the state update happens only at 10 Hz when the GNSS data is made available. Hence, between the two points of the available GNSS data, only motion prediction is performed and once GNSS data becomes available the predicted states are updated. Hence, the RobustStateNet is designed to operate at 50 Hz. Two variants of RobustStateNet, named $RSN0$ and $RSN1$ are trained with the different motion predictions $LSTM0$ and $LSTM1$ respectively. Both the variants, including the baseline of KalmanNet, are trained for 5 epochs.

Fig. 6 illustrates the RobustStateNet version 0, i.e. RSN0 state estimates for the scene number 918 of the NuScene dataset. We can observe that the input GNSS measurements are extremely noisy, in spite of which, the algorithm is able to provide reasonably accurate state estimation. It also needs to be considered that the input to the motion prediction model, from IMU, steering encoders, and wheel encoders are simulated to add additional noise and failures. Fig. 7 illustrates the mean and standard deviation of the MSE values of the estimated states for the different versions of RobustStateNet, the self-trained KalmanNet, and the Model-Based EKF computed for the 30 Monte Carlo simulations with randomly generated sensor failures. KalmanNet is able to outperform the MB EKF as it is able to learn the measurement

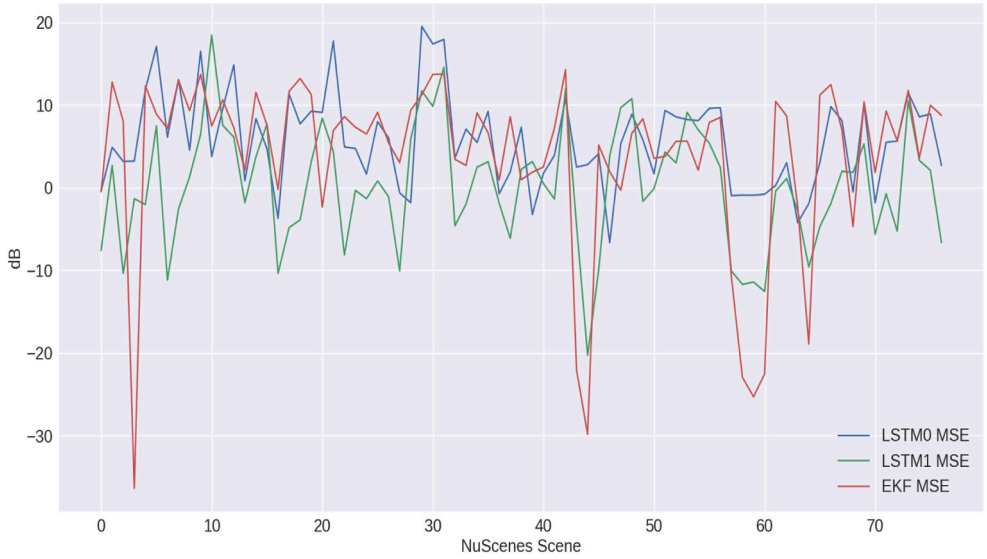


Fig. 5. MSE in dB for predicted state computed against the ground truth for different NuScene scenes from the test set.

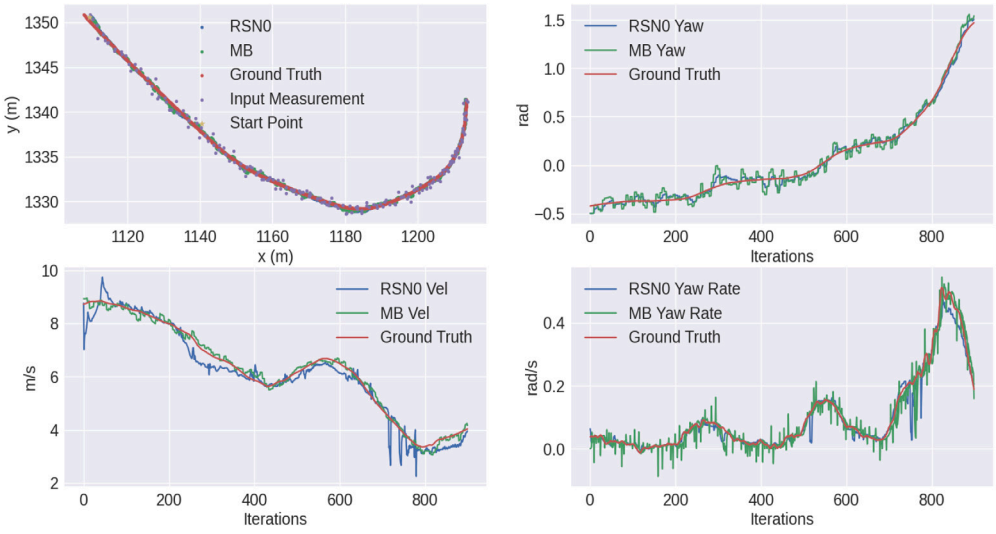


Fig. 6. Comparison of the states estimated using the RobustStateNet with the Model-Based (MB) EKF algorithm for test scene 918 of the Nuscenes dataset.

noise through the data and also to some extent account for the model mismatch and failures introduced by the sensor degradation in the prediction steps. RobustStateNet variants, RSN0 and RSN1 outperform both the baselines and provide the most robust and accurate estimates. In Table 2, the results of 100 Monte Carlo runs for the different RobustStateNet variants, KalmanNeT, and the model-based EKF are reported. In Tables 3 and 4, we report the mean Root Mean Square Error(RMSE) and its standard deviation for all the state components computed for the test scenes, i.e. the scenes 900 to 1000 from the NuScenes dataset. Table 3 reports the results for the cases without the sensor failure. We can observe that both variants of the RobustStateNet outperform the baseline EKF and the KalmanNet. Table 4 reports the results for the test set with the inclusion of the sensor failure and confirms the results of RobustStateNet's superior performance in terms of robustness and accuracy. Furthermore, RobustStateNet is able to perform each iteration of inference at 1 millisecond in a consumer laptop with an i7 Intel processor and Nvidia 3060 RTX GPU, hence ensuring the real-time applicability of the algorithm.

Table 2
Averaged MSE in dB for 100 Monte Carlo runs on the scene number 918 of the NuScene Dataset for two different variants of the RobustStateNet(RSN0 and RSN1), KalmanNet(KN), [7] and Model-Based EKF (MB).

Filter	RSN0	RSN1	KN	MB
Mean MSE (dB)	**−18.27**	−18.06	−11.90	8.62
Standard Dev (dB)	**0.866**	0.7286	0.3135	0.035

Table 3
RMSE Mean/Std for all the testing scenes from 900 to 1000 in the Nuscenes without the sensor failures for the proposed and the baseline algorithms: two different variants of the RobustStateNet(RSN0 and RSN1), KalmanNet(KN), [7] and Model-Based EKF (MB).

Filter	RSN0	RSN1	KN	MB
Position (m)	**0.030/8.85e−5**	0.034/5.26e−5	0.125/8e−3	0.043/8e−5
Yaw (rad)	**0.009/6e−4**	0.010/4e−4	0.011/4e−4	0.069/0.1
Vel (m/s)	0.0003/7.3e−8	**0.0002/1.34e−7**	0.019/1e−4	0.028/1e−4
YawRate (rad/s)	**0.0001/9e−8**	0.0001/1.5e−8	7e−6/8e−11	4e−4/3e−8

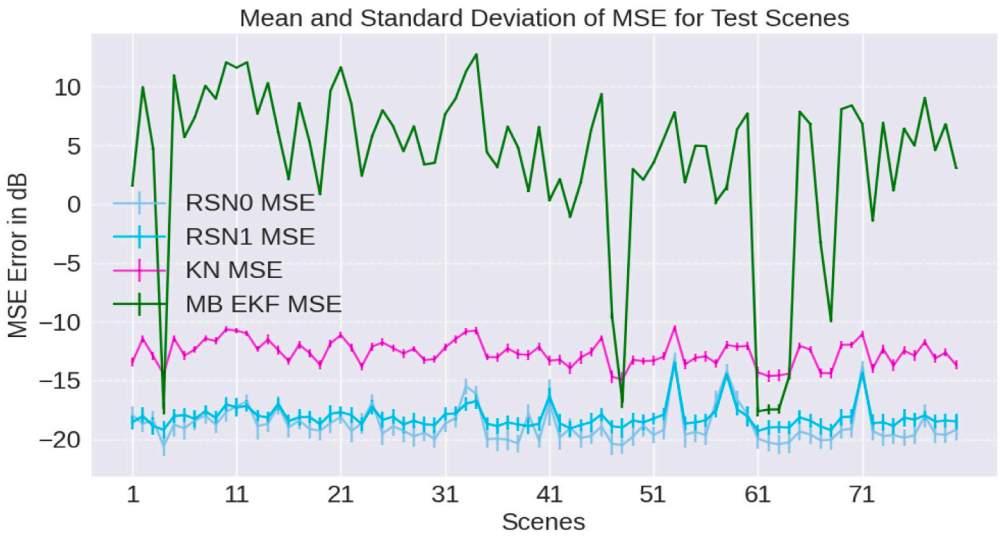


Fig. 7. Mean and Standard deviation of the Mean Square Error (MSE) in terms of dB of the state computed against the ground truth for different NuScene scenes for two different variants of the RobustStateNet (RSN0 and RSN1), KalmanNet (KN), [7] and Model-Based EKF (MB) for 30 Monte Carlo Simulation with artificially induced failures.

Table 4
RMSE Mean/Std for all the testing scenes from 900 to 1000 in the Nuscenes with the sensor failures for the proposed and the baseline algorithms.

Filter	RSN0	RSN1	KN	MB
Position (m)	**0.169/6e−4**	0.185/4e−4	0.35/2e−3	0.20/4e−4
Yaw (rad)	**0.070/6e−4**	0.087/4e−4	0.094/4e−4	0.116/0.1
Vel (m/s)	**0.0213/7.3e−8**	0.0242/1.34e−7	0.132/1e−4	0.164/1e−4
YawRate (rad/s)	**0.014/9e−8**	0.014/1.5e−8	0.013/8e−11	0.024/3e−8

6. Discussion

MB Kalman filters, which are used for state estimation, rely on a solid understanding of the statistical properties of the measurements. Specifically, they assume that the measurements follow a Gaussian distribution and are subject to additive noise In addition to the statistical requirements, implementing MB Kalman filters also necessitates well-defined motion and measurement models. These models describe how the state of the system evolves over time and how it is related to the measurements obtained. It is crucial to have accurate and reliable definitions of these models to ensure the Kalman filter performs optimally. If the parameters defining the models are erroneous or poorly defined, it can lead to significant degradation in the estimation accuracy of the Kalman filter. Recently proposed KalmanNet [7] performs well within the domain of the partial knowledge of the system dynamics. It does not require the definition of noise statistics however, does require an explicit definition of the motion and measurement models. The authors have demonstrated its superiority to the Model-Based filters and vanilla RNN in their work. This superiority can also be seen in the results portrayed by Fig. 7 and Table 2 when it was trained in our dataset. Our proposed RobustStateNet architecture maintains the structure of the MB filter with a clear definition of prediction and update steps. Like KalmanNet, we do not require any definition of noise statistics, which are learned implicitly by the models while performing filtering recursions. We take inspiration from KalmanNet to develop the update step of the filtering system with some modifications introduced to fit our application. However, the physics-based motion model used in the KalmanNet is replaced entirely by an LSTM-based robust motion predictor. The proposed two different variants employ different feature selection techniques, deterministic and stochastic feature selection. This feature selection allows for the introduction of robustness in the prediction model which in turn is exploited by the whole filtering recursion. RobustStateNet computes a state updating matrix, which can be seen as Kalman Gain (KG) and as demonstrated by [24], does provide insight into the confidence of the estimated state. Integration of the LSTM-based robust predictor enables the filtering recursion to perform better than KalmanNet, which is demonstrated by the results in Fig. 7 and Table 2.

7. Conclusions and future work

In this paper, we propose a novel LSTM-based motion predictor for ego vehicle state prediction and RobustStateNet, a robust ego vehicle state estimation algorithm. We simulate sensor failures into the NuScenes dataset to train the proposed models. The proposed models are compared against the baseline, the kinematic model for motion prediction, and MB EKF and self-trained KalmanNet [7] for the RobustStateNet. We demonstrated that the LSTM-based predictor's accuracy with the stochastic feature selector performs better in terms of accuracy and robustness than the kinematic model. When the prediction model is integrated into the RobustStateNet estimation framework, the overall model performs better than the model-based EKF and KalmanNet in all the state component predictions. These results hold true for scenarios both with and without the presence of sensor failures. Additionally, for future research, RobustStateNet can be extended to accommodate a learned mapping between the state and measurements, i.e. the measurement model in the MB KF filter setting. Furthermore, a state representation of ego motion can be studied.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The research employed publicly available data from the NuScenes CAN bus dataset.

Acknowledgments

This paper was supported by "Sustainable Mobility Center (Centro Nazionale per la Mobilità Sostenibile – CNMS)" project funded by the European Union NextGenerationEU program within the PNRR, Mission 4 Component 2 Investment 1.4.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2023.104585.

References

[1] T. Shan, B. Englot, LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2018, pp. 4758–4765, http://dx.doi.org/10.1109/IROS.2018.8594299.

[2] C. Chen, S. Rosa, C.X. Lu, B. Wang, N. Trigoni, A. Markham, Learning selective sensor fusion for state estimation, IEEE Trans. Neural Netw. Learn. Syst. (2022) 1–15, http://dx.doi.org/10.1109/TNNLS.2022.3176677.

[3] M. Bersani, M. Vignati, S. Mentasti, S. Arrigoni, F. Cheli, Vehicle state estimation based on Kalman filters, in: 2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive, AEIT AUTOMOTIVE, 2019, pp. 1–6, http://dx.doi.org/10.23919/EETA.2019.8804527.

[4] L. Han, Y. Lin, G. Du, S. Lian, DeepVIO: Self-supervised deep learning of monocular visual inertial odometry using 3D geometric constraints, 2019, http://dx.doi.org/10.48550/ARXIV.1906.11435, arXiv URL https://arxiv.org/abs/1906.11435.

[5] C. Chen, S. Rosa, Y. Miao, C.X. Lu, W. Wu, A. Markham, N. Trigoni, Selective sensor fusion for neural visual-inertial odometry, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2019, pp. 10534–10543, http://dx.doi.org/10.1109/CVPR.2019.01079.

[6] P. Dahal, S. Mentasti, L. Paparusso, S. Arrigoni, F. Braghin, Fault resistant odometry estimation using message passing neural network, in: 2023 IEEE Intelligent Vehicles Symposium, IV, 2023, pp. 1–8, http://dx.doi.org/10.1109/IV55152.2023.10186649.

[7] G. Revach, N. Shlezinger, X. Ni, A.L. Escoriza, R.J.G. van Sloun, Y.C. Eldar, KalmanNet: Neural network aided Kalman filtering for partially known dynamics, IEEE Trans. Signal Process. 70 (2022) 1532–1547, http://dx.doi.org/10.1109/tsp.2022.3158588.

[8] S. Wang, R. Clark, H. Wen, N. Trigoni, DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks, in: 2017 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2017, http://dx.doi.org/10.1109/icra.2017.7989236.

[9] S. Wang, R. Clark, H. Wen, A. Trigoni, End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks, Int. J. Robot. Res. 37 (2018) 513–542.

[10] Y. Almalioglu, M. Turan, M.R.U. Saputra, P.P. de Gusmão, A. Markham, N. Trigoni, SelfVIO: Self-supervised deep monocular Visual–Inertial Odometry and depth estimation, Neural Netw. 150 (2022) 119–136, http://dx.doi.org/10.1016/j.neunet.2022.03.005.

[11] M. Bersani, S. Mentasti, P. Dahal, S. Arrigoni, M. Vignati, F. Cheli, M. Matteucci, An integrated algorithm for ego-vehicle and obstacles state estimation for autonomous driving, Robot. Auton. Syst. 139 (2021) 103662, http://dx.doi.org/10.1016/j.robot.2020.103662.

[12] D. Chindamo, M. Gadola, Estimation of vehicle side-slip angle using an artificial neural network, MATEC Web Conf. 166 (2018) 02001, http://dx.doi.org/10.1051/matecconf/201816602001.

[13] D. Kong, W. Wen, R. Zhao, Z. Lv, K. Liu, Y. Liu, Z. Gao, Vehicle lateral velocity estimation based on long short-term memory network, World Electr. Veh. J. 13 (2021) 1, http://dx.doi.org/10.3390/wevj13010001.

[14] A. Bonfitto, S. Feraco, A. Tonoli, N. Amati, Combined regression and classification artificial neural networks for sideslip angle estimation and road condition identification, Veh. Syst. Dyn. 58 (2019) 1–22, http://dx.doi.org/10.1080/00423114.2019.1645860.

[15] N. Spielberg, M. Brown, N. Kapania, J. Kegelman, J. Gerdes, Neural network vehicle models for high-performance automated driving, Science Robotics 4 (2019) eaaw1975, http://dx.doi.org/10.1126/scirobotics.aaw1975.

[16] M. Rokonuzzaman, N. Mohajer, S. Nahavandi, S. Mohamed, Model predictive control with learned vehicle dynamics for autonomous vehicle path tracking, IEEE Access 9 (2021) 128233–128249, http://dx.doi.org/10.1109/ACCESS.2021.3112560.

[17] L. Hermansdorfer, R. Trauth, J. Betz, M. Lienkamp, End-to-end neural network for vehicle dynamics modeling, in: 2020 6th IEEE Congress on Information Science and Technology, CiSt, 2020, pp. 407–412, http://dx.doi.org/10.1109/CiSt49399.2021.9357196.

[18] M. Frosi, M. Matteucci, ART-SLAM: Accurate real-time 6DoF LiDAR SLAM, 2021, http://dx.doi.org/10.48550/ARXIV.2109.05483, arXiv. URL https://arxiv.org/abs/2109.05483.

[19] H. Coskun, F. Achilles, R. DiPietro, N. Navab, F. Tombari, Long short-term memory Kalman filters:Recurrent neural estimators for pose regularization, 2017, http://dx.doi.org/10.48550/ARXIV.1708.01885, arXiv URL https://arxiv.org/abs/1708.01885.

[20] Z. Shi, Incorporating transformer and LSTM to Kalman filter with EM algorithm for state estimation, 2021, http://dx.doi.org/10.48550/ARXIV.2105.00250, arXiv URL https://arxiv.org/abs/2105.00250.

[21] H. Caesar, V. Bankiti, A.H. Lang, S. Vora, V.E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, O. Beijbom, nuScenes: A multimodal dataset for autonomous driving, 2019, arXiv preprint arXiv:1903.11027.

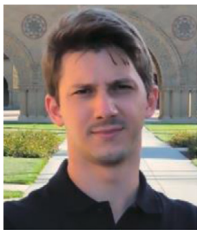
[22] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, http://dx.doi.org/10.48550/ARXIV.1412.6980, arXiv URL https://arxiv.org/abs/1412.6980.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[24] I. Klein, G. Revach, N. Shlezinger, J.E. Mehr, R.J.G. van Sloun, Y.C. Eldar, Uncertainty in data-driven Kalman filtering for partially known state-space models, 2021, http://dx.doi.org/10.48550/ARXIV.2110.04738, arXiv URL https://arxiv.org/abs/2110.04738.

Pragyan Dahal is a Ph.D candidate at Mechanical Engineering Department of Politecnico Di Milano, Italy. He received M.Sc in Mechanical Engineering from Politecnico Di Milano, Italy in 2020 with specialization on Mechatronics and Robotics. His main research interests include Environment Perception, Multi Object Tracking (MOT) algorithms, Control and Path Planning etc of Autonomous Vehicle.

Simone Mentasti is a researcher at Dipartimento di Elettronica Informazione e Bioingegneria of Politecnico di Milano, Italy. He obtained the M.S. degree in computer science from Università Statale di Milano, Italy, in 2017 and a Ph.D. in Computer Engineering from Politecnico di Milano in 2022. His interest focuses on robotics, perception, sensor fusion, sensor calibration and deep learning for autonomous driving cars. His research concerns the development of a sensor fusion framework for autonomous vehicles able to retrieve a uniform representation of the environment surrounding the car.



Luca Paparusso received the M.Sc. degree in mechanical engineering from Politecnico di Milano, Italy, in 2018, where he is pursuing the Ph.D. degree. He was research fellow at Istituto Italiano di Tecnologia (IIT), Italy, in 2019, and Visiting Ph.D. at Stanford University, Autonomous Systems Laboratory (ASL), CA, USA, from 2021 to 2022. His research is focused on trajectory forecasting and motion control for efficient and safe autonomous navigation in multi-agent environments

Stefano Arrigoni is currently a postdoctoral researcher at the institute of mechanical engineering at the Politecnico di Milano. He received the M.S. degree in mechanical engineering and the Ph.D. degree in applied mechanics both from Politecnico di Milano, Milano, Italy, in 2013 and 2017, respectively. His research interests lie in the area of autonomous vehicles with a focus on motion planning techniques and V2V communication.

Francesco Braghin (Member, IEEE) received the M.S. degree in mechanical engineering and the Ph.D. degree in applied mechanics both from Politecnico di Milano, Milano, Italy, in 1997 and 2001, respectively. In 2001, he became an Assistant Professor and in 2011, an Associate Professor with the Department of Mechanical Engineering, Politecnico di Milano. Since 2015, he has been a Full Professor in applied mechanics. Research interests range from autonomous vehicles to mechatronic systems in general.