End-to-End Velocity Estimation For Autonomous Racing

Sirish Srinivasan[1], Inkyu Sa[2], Alex Zyner[1], Victor Reijgwart[1], Miguel I. Valls[3] and Roland Siegwart[1]

***Abstract*— Velocity estimation plays a central role in driverless vehicles, but standard and affordable methods struggle to cope with extreme scenarios like aggressive maneuvers due to the presence of high sideslip. To solve this, autonomous race cars are usually equipped with expensive external velocity sensors. In this paper, we present an end-to-end recurrent neural network that takes available raw sensors as input (IMU, wheel odometry, and motor currents) and outputs velocity estimates. The results are compared to two state-of-the-art Kalman filters, which respectively include and exclude expensive velocity sensors. All methods have been extensively tested on a formula student driverless race car with very high sideslip (10°at the rear axle) and slip ratio ($\approx 20\%$), operating close to the limits of handling. The proposed network is able to estimate lateral velocity up to 15x better than the Kalman filter with the equivalent sensor input and matches (0.06 m/s RMSE) the Kalman filter with the expensive velocity sensor setup.**

I. Introduction

Self-driving cars have become popular in recent years because they promise to transform cities, provide universal access to mobility, and increase transport efficiency [1]. However, to reach full (level 4) autonomy as defined by the Society of Automation Engineers (SAE) [2], no driver attention must be required even in extreme scenarios or unusual conditions like adverse weather. Most research focuses on maneuvers that can be modeled kinematically since these cover most common cases. Dynamic maneuvers that imply high sideslip can arise in emergency avoidance maneuvers, in which the autonomous pipeline must also remain functional. Therefore, autonomous racing presents an interesting scenario to test all algorithms at the limit of handling.

Velocity estimation has a central and pivotal role in the entire autonomous system. The velocity estimates are for example used by the perception pipeline to perform motion undistortion on the LiDAR pointclouds, by the SLAM pipeline as the motion model, and by the vehicle motion control system for the critical task of providing state feedback. Velocity estimation must therefore provide high-rate, high-quality data. The overall architecture of similar autonomous race cars has been described in detail by Kabzan *et al.* [3] and a simplified diagram is shown in Figure 1.

The standard velocity estimation systems can be classified into two categories, both based on Kalman filters. The first type is often used in Electronic Stability Program (ESP) modules of road cars, like [4], [5], and does not have external velocity sensors. Such systems are designed to work in standard driving conditions but fail in extreme ones. The second type of velocity estimation systems relies on expensive sensors and is typically used on race cars. These systems prevail in extreme conditions, but cannot cost-effectively be deployed on commercial road cars [6], [7], [8].

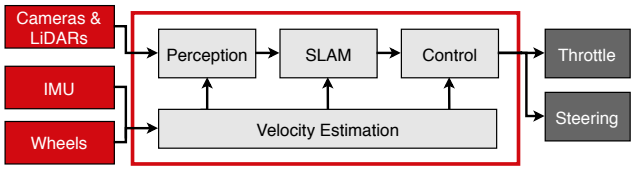

Fig. 1: Top: *pilatus* driverless, the formula student race car used for testing. ©FSG - Zenker. Bottom: Simplified software architecture of *pilatus* driverless race car showing the central role of velocity estimation

In this paper, we propose a learning-based method to perform velocity estimation in extreme scenarios like racing (second category) without external velocity sensors. This technique achieves equivalent performance to Kalman Filters that include external velocity sensors. The method is demonstrated on a full scale autonomous racecar (Figure 1) which is able to accelerate from $0-100km/h$ in $2.1s$ and reaches lateral accelerations of $1.7g$.

The contributions of this paper are as follows:

- Presenting a novel recurrent neural network application to estimate the velocity of a car using only raw, inexpensive sensor measurements (e.g., IMUs, wheel odometry and motor currents).
- Intensive real-world performance evaluation of the proposed approach and comparison with Kalman Filter approaches.
- Showing how the proposed end-to-end estimation approach outperforms state-of-the-art Kalman Filters with equivalent sensor setups by a large margin, and matches the approaches with external velocity sensors.

Section II presents related work. The proposed network, and Kalman filter are presented in Section (III), followed by real-world experimental results in Section IV. The limitations and conclusions are presented in Sections V and VI.

[1]Authors are with the Autonomous Systems Lab, ETH Zürich, Zürich
[2]Author is with Robotics and Autonomous Systems Group, CSIRO, Pullenvale, Australia
[3]Author is affiliated with Sevensense Robotics A.G, Zürich

II. Related work

In this section, we start by reviewing conventional state estimation approaches (e.g., Stochastic Filtering) and then move on to deep learning-based approaches from which we drew inspiration.

Xue and Schwartz [9] compare the state of the art approaches for state estimation in wheeled mobile robot applications. The main differences between these approaches and race cars are the complex dynamics that occur when driving at the limits of handling ($18m/s$ and $1.5g$ lateral acceleration), which are very difficult to model completely. Wheel speeds are a biased measurement of the velocity, due to the high lateral and longitudinal slips (up to 20%) that race cars require to reach optimal accelerations as described by Pacejka [10].

An Extended Kalman Filter (EKF) is used by Valls *et al.* [6] to fuse data from multiple sensors and perform velocity estimation for an autonomous race car. Outlier detection and observability analysis have been studied as a part of this work to find the minimal sensor setup. Gosala *et al.* [7] extend this work by using a complex slip ratio-based model to perform reliable velocity estimation even during sensor failure. The only drawback of this approach is the dependence on dedicated velocity sensors to provide a reliable estimate. The model mismatch at high speeds and during aggressive maneuvers causes the estimate not to be as accurate. Wischnewski *et al.* [8] describe an EKF to fuse data from a velocity sensor and lidar data for velocity estimation in a high-speed roborace car. Even they discuss a reduction in accuracy of the estimate during velocity sensor failure.

The rise of deep learning not only has flourished object detection and classification from an image using convolutional neural networks but has accelerated learning complex dynamics or underlying kinematics by making use of sequential and temporal neural networks.

A deep Rectified Linear Unit (ReLU) based neural network has been used by Punjani and Abbeel [11] to model the complex dynamics of a helicopter during aggressive maneuvers. They show that this model outperforms state-of-the-art methods by more than 50%. Spielberg *et al.* [12] use a deep neural network with delayed inputs over multiple time steps to model the dynamics of a high-performance car and have shown this works over a variety of operating conditions. These approaches motivate the usage of neural networks to model complex dynamics that would not be possible with traditional approaches.

Karpathy [13] demonstrates the effectiveness of Recurrent Neural Networks (RNNs) in time series prediction. It has been shown to learn the time dependency of the underlying dynamics without explicitly defining the number of input time steps. Only one time step is used as input every time step and the hidden state is propagated over time. This reduces the need to optimise for the number of input time steps to the network during deployment.

One drawback of RNNs is the vanishing gradient problem during back propagation which has been discussed in detail by Pascanu *et al.* [14]. To counter this, Hochreiter *et al.* [15] use Long-Short Term Memory (LSTM) cells to build the RNN. This allows the network to propagate gradients over time and learn long-term dependencies. Jozefowicz *et al.* [16] compares the performance of Gated Recurrent Units (GRU) and LSTM cells. They show that LSTMs are harder to train and are dependent on the chosen network architecture. Also, an LSTM network has more weights to optimise when compared to a GRU, requiring more data to train.

Drews *et al.* [17] use a mono camera processed using CNNs combined with LSTMs to obtain a representation of the track. This is then combined with IMU, wheelspeeds and RTK GNSS using a particle filter to estimate the states. This motivates the usage of RNNs for state estimation, especially in very aggressive driving scenarios. A drawback of this approach is that RTK GNSS is not always available.

Overall, modelling the dynamics of a vehicle involves the identification of many parameters, most of which have a very minor effect on the actual dynamics. Also, extracting meaningful information from noisy data requires extensive filtering techniques. We combine both these steps and use a machine learning-based approach to perform end-to-end velocity estimation from the sensor measurements to state estimates. To the best of the authors' knowledge, this approach hasn't been done before in the literature.

This is then tested on real sensor data obtained on a formula student driverless car, and the performance is compared to the Kalman Filter reference.

III. Methodology

In this section, we present two approaches to estimate the car's velocity: a classic filtering approach (Mixed Kalman Filter), and a learning-based approach (End-to-end Recurrent Neural Network).

A. Sensors

The sensor setup, shown in Figure 2, includes two IMUs measuring accelerations and rotational rates, four motor encoders measuring the wheel speeds, steering angle sensor and motor torques of each wheel (derived from the motor current sensors). There are two velocity sensors (an optical flow-based velocity sensor and a GNSS velocity sensor) which are used only for validation and target generation.

B. Mixed kalman filter

As a baseline and the first approach to estimate velocity, we develop a Kalman Filter to fuse all the sensor measurements. The presented propagation model (Eq. 2) is mildly non-linear, and all measurement models are linear except for the combined velocity measurement (Eq. 3), which is highly non-linear. For this reason, and to render the filter as computationally light as possible, a mix of the Linear Kalman filter (LKF), Extended Kalman filter (EKF) and Unscented Kalman Filter (UKF) is used. The Mixed Kalman Filter (MKF) propagates the state and covariance with an EKF step, the rotation and acceleration measurement with LKF updates, and only the combined velocity measurement

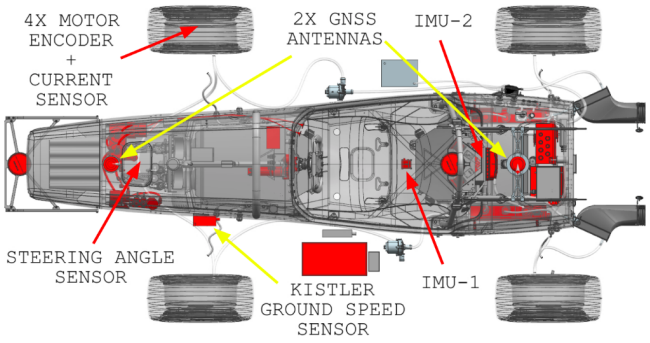

Fig. 2: Sensor setup: two IMUs, 4x motor encoders + current sensors, steering angle sensor and two dedicated velocity sensors that are used for validation and target generation

with a UKF update. This proved to increase the accuracy of the filter with reduced computational load, beating state-of-the-art filters such as [7].

1) States and propagation model: The system's state, $\mathbf{x}$, is defined as:

$$\begin{aligned} \mathbf{x} &= [\mathbf{v}^T,\, \dot{\psi},\, \mathbf{a}^T]^T \\ \mathbf{v} &= [v_x,\, v_y]^T,\, \mathbf{a} = [a_x,\, a_y]^T \end{aligned} \tag{1}$$

where $\mathbf{v}$ is the velocity, $\dot{\psi}$ the rotational rate along the z-axis (i.e., yaw) and $\mathbf{a}$ the acceleration. The system evolution can described by:

$$\begin{aligned} \dot{\mathbf{v}} &= \mathbf{a} + \left[v_y\dot{\psi},\, -v_x\dot{\psi}\right]^T + \mathbf{n_v} \\ \ddot{\psi} &= n_r,\, \dot{\mathbf{a}} = \mathbf{n_a} \end{aligned} \tag{2}$$

where $n_{\{\cdot\}}$ are white i.i.d noises.

2) Measurement model: The measurement models of rotation rates, and accelerations are straight forward since they belong to the state. For the other sensors, they are combined into one single velocity measurement as follows:

$$\mathbf{z_v} = h_v(\mathbf{x}) = (\mathbf{v}_{\text{min}} + [-\dot{\psi}\, p_y,\, \dot{\psi}\, p_x]^T) + \mathbf{n_{z_{vx}}} \tag{3}$$

$$\mathbf{v_i} = [cos(\delta_i),\, sin(\delta_i)]^T \cdot \omega_i \cdot R_i/(SR(T_i)+1) + n_{z_v} \tag{4}$$

$\mathbf{z_v}$ is the combined velocity measurement, $\mathbf{v_i}$ is the velocity of the *ith* wheel in car frame, ω_i is the rotational velocity of the *ith* wheel, δ is the angle of each wheel w.r.t car frame, which is the steering angle for the front and 0 for the rear wheels. $SR(\cdot)$ is a function that maps the torque applied on a wheel (T_i) to its slip ratio under low slip conditions (Pacejka magic tire model [10]). R_i is the radius of the *ith* wheel. p_x and p_y is the wheel position in car frame.
Slip ratio calculation is fairly accurate at low slips, but uncertain at high slips, this is why in practice, the velocity update only takes the single wheel with smallest absolute SR. The linear velocity of this wheel is denoted $\mathbf{v}_{\text{min}}$

C. End-to-End Recurrent Neural Network

The core of this work is a recurrent end-to-end learning approach. The temporal nature of the data makes recurrent neural networks particularly suitable. Two networks are presented and compared, each being the best performing hyperparameter combination with 1 and 2 GRU layers, denoted RNN-1 and RNN-2, respectively.

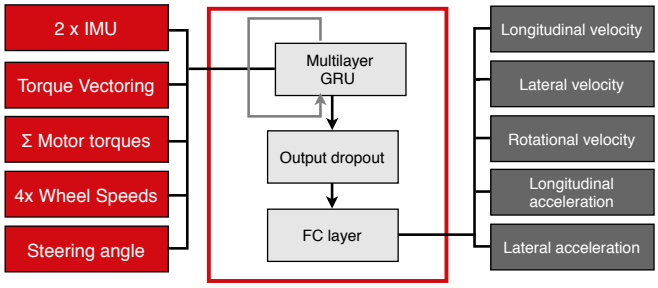

Fig. 3: Basic architecture of the proposed recurrent neural network consisting of multiple hidden GRU layers that propagate over time, followed by an output dropout and a fully connected dense layer to extract the outputs.

1) Target calculation: To generate the target for the RNN, an MKF was used (See Section III-B) in combination with the sensors described in Section III-A including the external velocity sensors. This filter was extensively tested and proved to be successful in multiple Formula Student competitions around Europe in 2019. The target of the end-to-end approach should thus obtain similar results but do so without relying on expensive velocity sensors. The output from the MKF is post-processed using a non-causal Gaussian moving average filter to obtain a smoothed non-delayed target referred to as the reference.

2) Network architecture: The inputs to the network are the sensors described in Sec. III-A, and the output is the state estimate, described in Sec. III-B.1. Different architectures were explored, and two were selected, named RNN-1 and RNN-2 and shown in Figure 3. RNN-1 has a single hidden layer of 64 GRU neurons while RNN-2 has two hidden layers of 32 neurons each in the multilayer GRU. The rest of the architecture remains the same for both the networks, and includes a dropout layer and a fully connected dense layer, to transform the inner network state to the outputs.

3) Activation function: Pascanu *et al.* [14] discuss the problem of vanishing gradient for RNNs, especially when the activation function used saturates for high inputs - like sigmoid and tanh. To overcome this, Xu *et al.* [18] suggest using a leaky-ReLU which does not saturate for high inputs and has a small negative output for negative inputs. This ensures that cells do not "die" and that they participate in the training even after being wrongly modified by the optimiser.

4) Optimiser: The main trade-off for an optimiser lies between training time and stability. Two common optimisers that implement adaptive learning rate are adadelta [19] and adam [20]. Training with adadelta consistently converged irrespective of network architecture, probably due to its lack of tuning parameters. On the other hand, Adam's learning rate needs to be tuned to find this trade-off. However, once a good learning rate has been identified, adam slightly outperformed adadelta. Thus, adam with a learning rate of 5×10^{-4} was chosen.

5) Output dropout: Dropout is added to only the output layers, and not the recurrent layers as per Zaremba *et al.* [21]. Output dropout is applied only during training and not during testing and validation to provide the most accurate prediction from the network. A dropout fraction of 7.5% was

Dataset Type	# Datasets	Duration
Training	11	22 min 28.440 s
Testing	3	5 min 8.485 s
Validation	4	4 min 40.990 s
Total	17	32 min 17.915 s

TABLE I: Dataset split between training, testing and validation for the recurrent neural network

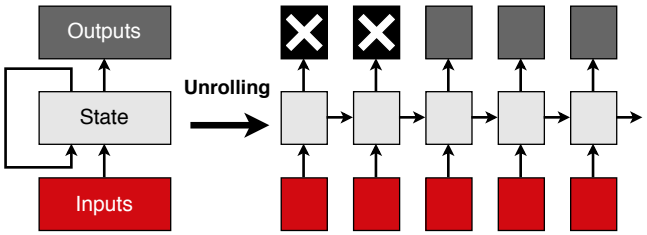

Fig. 4: Pictorial representation of a training sample with the inputs and outputs in each time step. The hidden state propagates over time providing the possibility to learn dependencies in the time-series data. The output for the initial time steps (shown in black, crossed out) are ignored while calculating losses.

found to have the similar train, test and validation losses and the smallest validation loss.

IV. Experiments and Analysis

In this section we refer to the MKF with velocity sensors as reference, and the MKF without velocity sensors as baseline. We refer to RNN-1 and RNN-2 as described in Sec. III-C

A. Data collection

Datasets were created from real data obtained from noisy sensors over both testing and competition runs. The hardware time-synced measurements are sampled at a constant rate of 200 Hz (zero-order hold) and these raw sensor measurements are used as input to the network. The target is generated as described in Sec. III-C.1. The datasets include data from different road surfaces (flat, bumpy, gravel), temperatures (20-60 °C) and grip conditions. This data has been split into training, testing and validation as shown in Table I.

B. Implementation details

The RNN has internal hidden states necessary to learn dependencies between the time series data, shown graphically in Figure 4. The RNN predictions for the first few time steps are ignored and the prediction loss computed as the root mean square error (RMSE) after this initial setup period. This period is chosen as 200 samples or $1s$, which provides a nice trade off between output quality and initial wait time.

The training is performed in parallel using a batch size of 500 steps, and including standard methods like input-output normalisation, gradient clipping and early stopping. The network was implemented in Tensorflow using the dynamic_rnn package for unrolling over time. Multiple GRU layers are stacked together using the Multi RNN Cell package.

The prediction time of the network while running on a Nvidia GeForce RTX 1080 Ti GPU for 500 timesteps was found to be 0.20 ms.

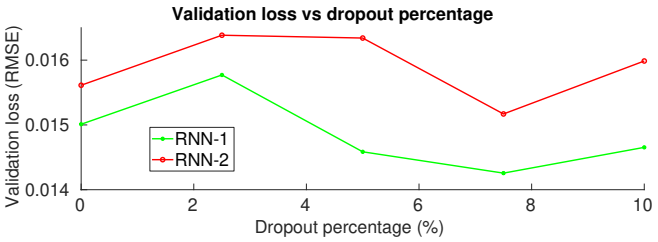

Fig. 5: Validation loss as a function of output dropout percentage. The dropout percentage with minimum validation loss is 7.5% for both networks

Hyper-parameter	Searched values
Number of GRU layers	**1**, 2, 3
Neurons per layer	16, ...**64** ...256
Input steps for training sample	20...**300**...500
Output steps for training sample	100...**200**...1000
Activation function	relu, **leaky-relu**, elu
Optimiser	adadelta, **adam**
Learning rate	0.0001...**0.0005**...0.01
Gradient Clipping - global norm	No, **Yes**
Dropout fraction	0...**0.075**...0.25
Training epochs	1,000...**10,000**

TABLE II: Hyper-parameters and the ranges that were iterated over with the final values in bold

C. Hyperparameters and Ablation study

The different hyper-parameters that were searched are listed in Table II and final values are shown in bold. An ablation study is performed for the least standard hyperparameter - output dropout percentage. Results are shown in Figure 5 and it can be seen 7.5% has lowest validation loss.

D. Accuracy

The main quantitative analysed metrics are the RMSE of the prediction (from RNN or MKF) vs the reference as well as the normalized error %. The results can be seen in Table III. It can be noted that lateral velocity has the highest prediction error, and this is due to it being the hardest to estimate given the input sensors. In fact the baseline MKF has an error 15 times larger than both RNNs.

In addition to the RMSE, we analyse the quality of the estimate by visual inspection. Figure 6 shows the lateral velocity error at different positions along the track for one Formula Student Germany (FSG) run. The regions with the highest error correspond to the ones where the vehicle experiences the highest lateral slip. Figure 7 shows the performance of the proposed recurrent neural networks as compared to the reference and the MKF baseline over the entire lap. The networks clearly outperform the baseline and perform similar to the reference even without using velocity sensor measurements. This highlights the dependence of the Kalman filter on a direct velocity sensor.

State	Baseline		RNN-1		RNN-2	
	RMSE	%Error	RMSE	%Error	RMSE	%Error
v_x	0.779	5.95	0.141	**0.94**	0.159	1.06
v_y	0.898	59.78	0.059	**3.90**	0.061	4.09
$\dot{\psi}$	0.018	**1.21**	0.029	1.91	0.030	2.00
a_y	0.665	4.43	0.411	**2.74**	0.450	3.00

TABLE III: Quantitative Performance metrics comparing the performance of the proposed recurrent neural networks with the Kalman filter reference and baseline

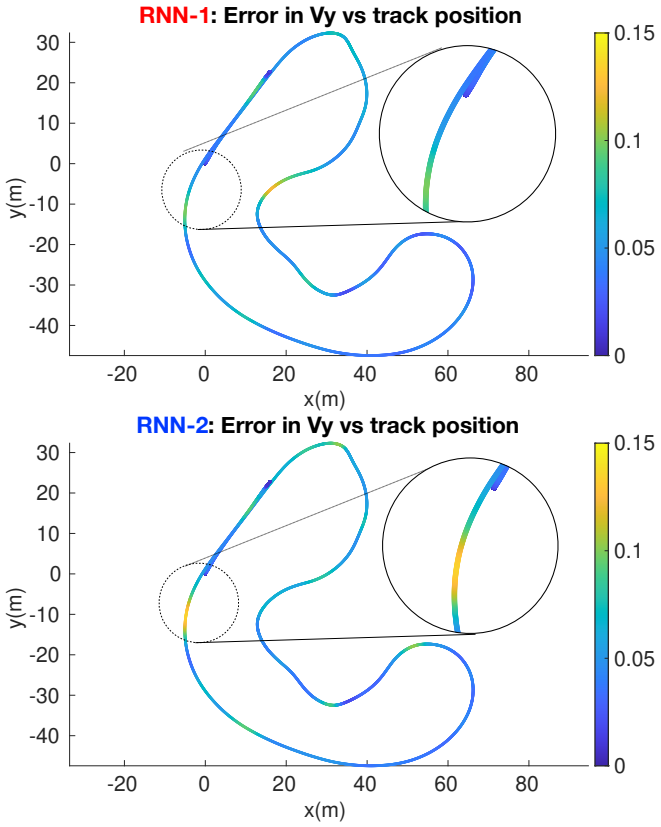

Fig. 6: Lateral velocity error along the track for the single layer RNN (RNN-1 in red) and double layer RNN (RNN-2 in blue). The highest error can be observed in the sharp transition from right to left for RNN-1 and in the high speed corner for RNN-2.

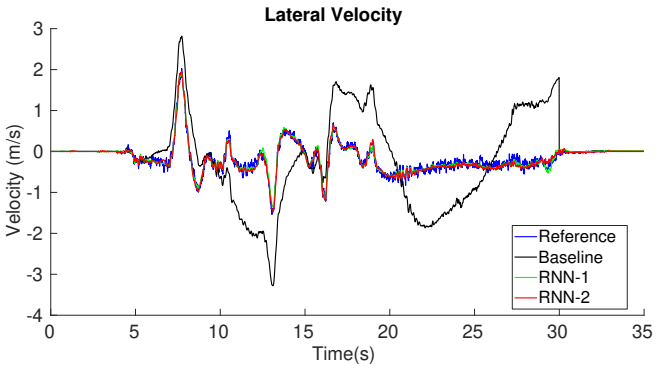

Fig. 7: Comparison of the lateral velocity estimates shows the superior performance of the proposed networks w.r.t the Kalman filter baseline. The predictions are comparable to the Kalman filter reference even for the hardest state to estimate.

E. Case studies

We discuss some case studies below that highlight the performance of the proposed network in some difficult scenarios for velocity estimation. Since both the one layer and two layer architectures have similar performance, the single layer RNN is preferred due to a lower prediction time ($\approx$ 30%) and we would focus on this in the rest of the paper.

1) Bias calibration: For the MKF, the biases of the various sensors are calibrated explicitly either during startup or while running. This is necessary due to biased sensor measurements. This has been learned implicitly by the

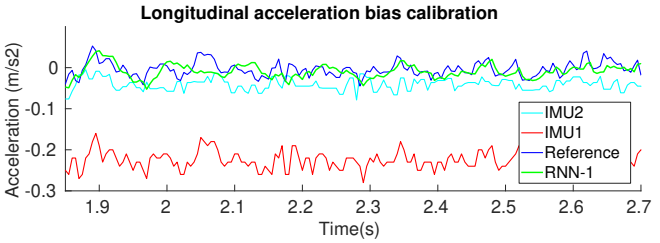

Fig. 8: The proposed network is able to calibrate for the bias in raw sensor measurements and provide an unbiased estimate of longitudinal acceleration

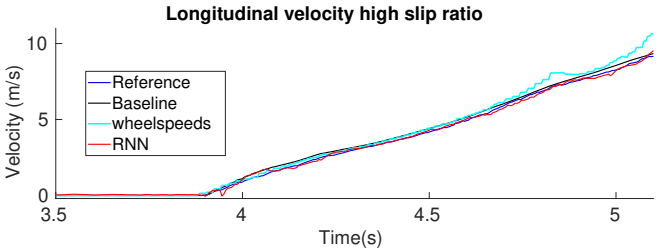

Fig. 9: The performance of the proposed network for the longitudinal velocity estimate is comparable to the Kalman filter reference even during vehicle launch - a very high slip ratio condition ($\approx$ 20%)

proposed network as can be seen in Figure 8. Two sensor measurements, one from each IMU, with different biases are shown while the vehicle is stationary. As expected, the output from the network is an unbiased estimate of the acceleration centered at zero.

2) Vehicle launch: During launch, the car accelerates from stand still using traction control to maximise its acceleration. This requires an accurate estimate of the vehicle's speed which is hard without velocity sensors since the wheels are slipping. Figure 9 shows the performance of the proposed network in this case. Initially the velocity is slightly over-estimated due to wheel slip, but then the network is able to identify this after a few steps and corrects it resulting in a performance similar to the reference.

3) High lateral slip: To minimise the lap-time, carrying speed through corners is essential. To maximise the cornering force, the car needs to maintain an optimal lateral slip at the tires. This is a difficult condition to control and relies on velocity feedback from velocity estimation to ensure stability. Figure 10 shows the lateral velocity estimate of the proposed solutions during very high lateral slip ($\approx$10°at the rear axle). The proposed network has learned the complex dynamics for the hardest state in a very aggressive maneuver and the estimate is very close to the reference and significantly outperforms the baseline while using the same sensor setup.

4) Outlier rejection: The importance of rejecting outlier measurements has been discussed in detail by Valls *et al.* [6]. In Figure 11, it can be seen that the measurements of one IMU (IMU-2) are not updated after t $\approx$ 149s. In this case, the Kalman filter has explicit outlier detection that rejects the bad sensor measurement to provide a good estimate of the acceleration. The proposed network is capable of detecting this based on the learnt dynamics and the measurement of the other IMU (IMU-1) resulting in the outlier measurement being implicitly rejected to match the reference.

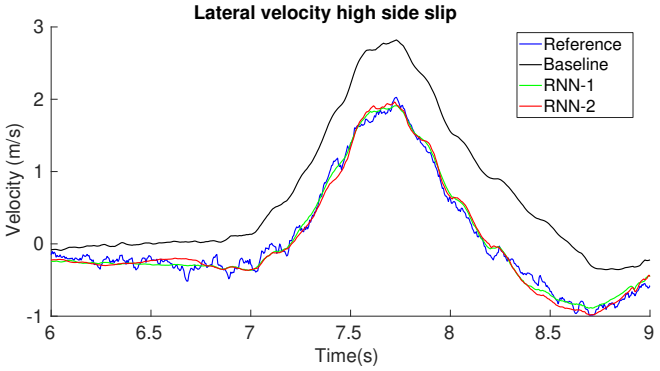

Fig. 10: Comparison of the lateral velocity estimate during high lateral slip condition ($\approx 10°$at the rear axle) shows the proposed recurrent neural network completely outperform the Kalman filter baseline and produce a result comparable to the reference

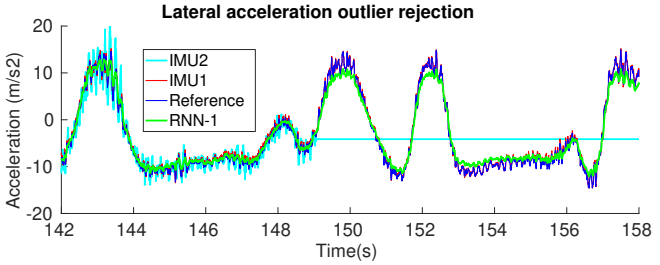

Fig. 11: The proposed network is able to reject bad measurements and provide a very good estimate even under very aggressive driving conditions, validated in the lateral acceleration estimate during failure of one IMU (shown in yellow)

V. Discussion and Limitations

An advantage of Kalman filters, as opposed to the RNNs, is the notion of uncertainty. Having an estimate of the uncertainty could be useful for downstream modules, but is not possible with the current architecture. Gal and Ghahramani [22] suggest using dropout to obtain a Monte Carlo distribution. Mixture density functions described by Bishop [23] could also be used.

Another limitation would be the inability to incorporate minor, known changes. Re-training with new data is currently the only option.

VI. Conclusion

We present a novel end-to-end recurrent neural network application that takes affordable and available raw sensors as input (IMU, wheel odometry, and motor currents) and estimates the output velocity. This network is extensively tested on real-world data generated while racing autonomously, evaluated at very high sideslip (10° at the rear axle), close to the limits of handling. The network outperforms the state-of-the-art Kalman filter with equivalent input data and matches the Kalman filter with expensive external velocity sensors.

Acknowledgment

The authors thank the AMZ Driverless team for their sustained hard-work and passion, as well as the sponsors for their financial and technical support. We also express our gratitude to everyone at the Autonomous Systems Lab of ETH Zürich for their support throughout the project.

References

[1] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2020.

[2] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles (j3016_201609)," https://www.sae.org/standards/content/j3016_201609/, accessed: 2020-02-24.

[3] J. Kabzan, M. I. Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, *et al.*, "AMZ driverless: The full autonomous racing system," *arXiv preprint arXiv:1905.05150*, 2019.

[4] H. Guo, D. Cao, H. Chen, C. Lv, H. Wang, and S. Yang, "Vehicle dynamic state estimation: state of the art schemes and perspectives," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 2, pp. 418–431, 2018.

[5] A. Y. Ungoren, H. Peng, and H. Tseng, "A study on lateral speed estimation methods," *International Journal of Vehicle Autonomous Systems*, vol. 2, no. 1-2, pp. 126–144, 2004.

[6] M. I. Valls, H. F. Hendrikx, V. J. Reijgwart, F. V. Meier, I. Sa, R. Dubé, A. Gawel, M. Bürki, and R. Siegwart, "Design of an autonomous racecar: Perception, state estimation and system integration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2048–2055.

[7] N. Gosala, A. Bühler, M. Prajapat, C. Ehmke, M. Gupta, R. Sivanesan, A. Gawel, M. Pfeiffer, M. Bürki, I. Sa, *et al.*, "Redundant perception and state estimation for reliable autonomous racing," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6561–6567.

[8] A. Wischnewski, T. Stahl, J. Betz, and B. Lohmann, "Vehicle dynamics state estimation and localization for high performance race cars," in *IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019*. Elesevier, Aug 2019.

[9] Z. Xue and H. Schwartz, "A comparison of several nonlinear filters for mobile robot pose estimation," in *2013 IEEE International Conference on Mechatronics and Automation*, 2013.

[10] H. B. Pacejka, *Tire and Vehicle Dynamics*. Butterworth-Heinemann, 2012.

[11] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3223–3230.

[12] N. A. Spielberg, M. Brown, N. R. Kapania, J. C. Kegelman, and J. C. Gerdes, "Neural network vehicle models for high-performance automated driving," *Science Robotics*, vol. 4, no. 28, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/28/eaaw1975

[13] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks."

[14] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013, pp. 1310–1318.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[17] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Vision-based high-speed driving with a deep dynamic observer," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1564–1571, April 2019.

[18] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[19] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[20] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[21] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[22] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48. PMLR, 20–22 Jun 2016.

[23] C. M. Bishop, "Mixture density networks," Tech. Rep., 1994.