Transformer Approach to Kalman Filter

Idan Daniel
308088624
Tel-Aviv University
idand242@gmail.com

Mor Keshet
ID
Tel-Aviv University
EMAIL

Abstract

Kalman Filters are one of the most influential models of time-varying phenomena. They admit an intuitive probabilistic interpretation, have a simple functional form, and enjoy widespread adoption in a variety of disciplines. Motivated by recent Hybrid-Model method, that uses deep learning RNNs and classic Kalman Filter algorithms for having better predictions, we introduce a transformers hybrid algorithm that replace the RNN approach to efficiently learn a broad spectrum of Kalman filters. We investigate the accuracy of such model for predictions based on sensors measurements, and to that end we used synthetic dataset created by the data generator program that was used in earlier article "KalmanNet" and include long-term structure, noise and bias that applied to ground truth trajectories. We show the efficacy of our method for this dataset against Extended Kalman Filter and KalmanNet.

1. Introduction

1.1. Kalman Filter (KF)

Kalman Filter (KF) [Kalman, 1960] is a widely-used method for linear filtering and prediction, with applications in many fields [Paul Zarchan, 2000] including object tracking [Kirubarajan, 2002]. A major challenge in the implementation of a KF is to determine the parameters Q, R representing the covariance matrices of the noise in the motion model and in the observation model, respectively. Many works have addressed this issue over the years [Abbeel et al., 2005, Odelson et al., 2006, Zanni et al., 2017, Park et al., 2019], aiming to allow estimation of the noise covariance under various conditions (e.g., non-stationary processes, or unknown system's state in the training data). The true covariance matrices of the noise indeed yield optimal tracking under KF assumptions – that is, known and linear motion and observation models, with i.i.d and normally-distributed noise. These assumptions are quite restrictive and are often not applied to real-world problems. Under different sets of assumptions, the covariance of the noise does not necessarily correspond to optimal tracking. While the original KF assumes linear SS models, many problems encountered in practice are governed by non-linear dynamical equations. Therefore, shortly after the introduction of the original KF, non-linear variations of it were proposed, such as the extended Kalman filter (EKF) and the Parts of this work focusing on linear Gaussian state space models were presented at the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). unscented Kalman filter (UKF). Methods based on sequential Monte-Carlo (MC) sampling, such as the family of particle filters (PFs) were introduced for state estimation in non-linear, non-Gaussian SS models. The limitations of MB Kalman filtering and DD state estimation motivate a hybrid approach that exploits the best of both worlds; i.e., the soundness and low complexity of the classic KF, and the model-agnostic nature of DNNs.

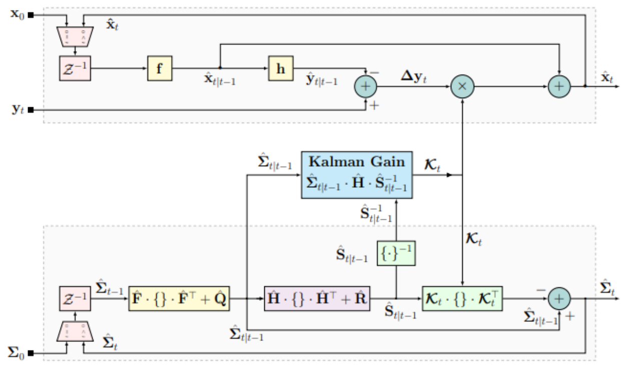
$\mathbf{x}_0$
$\hat{\mathbf{x}}_t$
$\mathcal{Z}^{-1}$
$\mathbf{f}$
$\hat{\mathbf{x}}_{t|t-1}$
$\mathbf{h}$
$\hat{\mathbf{y}}_{t|t-1}$
$\Delta\mathbf{y}_t$
$\mathbf{y}_t$
Kalman Gain
$\hat{\Sigma}_{t|t-1} \cdot \hat{\mathbf{H}} \cdot \hat{\mathbf{S}}_{t|t-1}^{-1}$
$\mathcal{K}_t$
$\hat{\mathbf{S}}_{t|t-1}$
$\{\cdot\}^{-1}$
$\hat{\Sigma}_{t-1}$
$\mathbf{F} \cdot \{\} \cdot \mathbf{F}^\top + \mathbf{Q}$
$\mathbf{H} \cdot \{\} \cdot \mathbf{H}^\top + \mathbf{R}$
$\mathcal{K}_t \cdot \{\} \cdot \mathcal{K}_t^\top$
$\hat{\Sigma}_t$
Σ_0

Figure 1: EKF block diagram. Here, Z −1 is the unit delay

1.2. Recurrent Neural Networks (RNNs) - GRU

Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. The gated recurrent unit is a special type of optimized LSTM-based recurrent neural network. The GRU internal unit is similar to the LSTM internal unit, except that the GRU combines the incoming port and the forgetting port in LSTM into a single update port. In, a new system called the multi-GRU prediction system was developed based on GRU models for the planning and operation of electricity generation. The GRU was introduced by Cho et al. Although it was inspired by the LSTM unit, it is considered simpler to calculate and implement. It retains the LSTM immunity to the vanishing gradient problem. Its internal structure is simpler and, therefore, it is also easier to train, as less calculation is required to upgrade the internal states. The update port controls the extent to which the state information from the previous moment is retained in the current state, while the reset port determines whether the current state should be combined with the previous information.

1.3. Transformers

Transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence. Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, to detect subtle ways even distant data elements in a series influence and depend on each other. First described in a 2017 paper from Google, transformers are among the newest and one of the most powerful classes of models invented to date. They're driving a wave of advances in machine learning some have dubbed transformer AI. Stanford researchers called transformers "foundation models" in an August 2021 paper because they see them driving a paradigm shift in AI. In addition, in comparison to RNN units as GRU, LSTM transformers can use parallelism better, it built more over metrics multification. As we want to have good performance for using Kalman Filter on real time, the parallelism quality is advantage for this use.

2. Related Work

In this section we will discuss three approaches to improve Kalman Filter using deep neural-network and how if reflect our attitude to improve Kalman Filter.

2.1. Optimized Kalman Filter

Under the assumption that the noise metrics Q, R are not known or not optimal, this work uses the metrics as weights and optimize over training data. Since the noise metrics are optimized and load to classic Kalman Filter, this approach can be referred as parameter tuning to classic Kalman filter. Kalman Filter tries to maximize the probability for its prediction regarding Q, R the noise metrics for the next state uncertainty and measurement noise relatively. By using this tuning, the prediction state error is minimized as we can see in figure 2. Since the optimized metrics are constants they are not depends over the preview's measurements or predictions. Moreover, the optimization

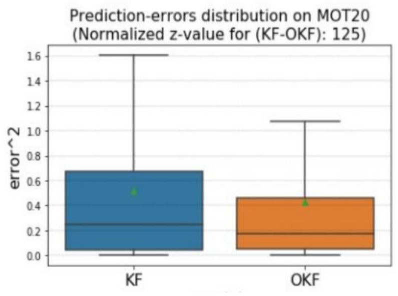

Figure 2: Summary of the prediction errors of KF and OKF on 1208 targets of the test data of MOT20. The errors of OKF are clearly smaller than those of KF

is all over the training data although every measurement might have different noise (Measurements noise can be influenced by the distance from the sensor).
The work has conclusions point out that the noise metrics might change for every measurement and even that in many cases not using the Q metric at all gave better result.

2.2. Kalman Net - #1 Architecture

The method tries to implement deep-learning into the classic Kalman Filter by replacing the Kalman-Gain calculations with GRU and FC layers. It tries to use parts from the classic model (ex. motion model and structure) and improve it. The advantage of Kalman filter is the simplicity because many uses are real time calculation. The first architecture of KalmanNet asks for big number of parameters, according the article utilizing the first architecture requires the order of 5 · 105 trainable parameters. This work will use the same concept of replacing the Kalman Gain calculations with deep neural network but uses part of the original features in the classic Kalman Filter. Since GRUs are RNN as they contain hidden state from previews time stamps, this break the Markov assumption of Kalman Filter and give better results, then we will use number of previews measurements and states.

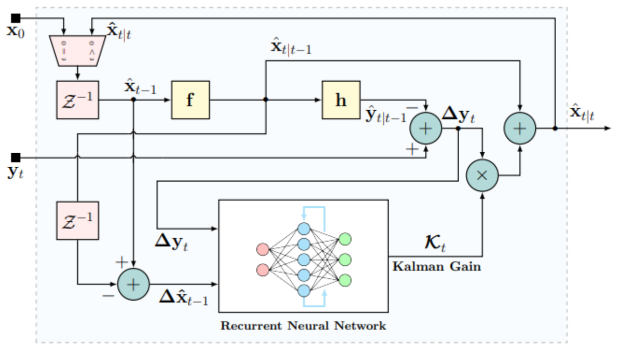

Figure 3: Kalman Net block diagram.

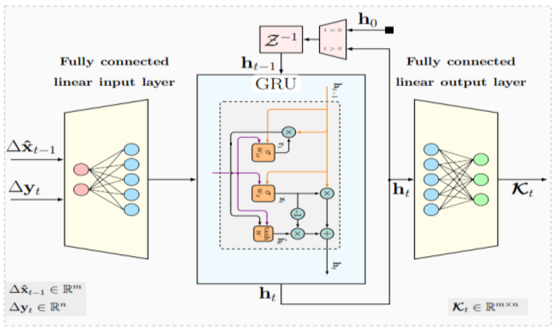
Fully connected
linear input layer
GRU
Fully connected
linear output layer
$\mathbf{h}_0$
$\mathbf{h}_{t-1}$
$\mathbf{h}_t$
$\Delta\hat{\mathbf{x}}_{t-1}$
$\Delta\mathbf{y}_t$
$\mathcal{K}_t$
$\Delta\hat{\mathbf{x}}_{t-1} \in \mathbb{R}^m$
$\Delta\mathbf{y}_t \in \mathbb{R}^n$
$\mathcal{K}_t \in \mathbb{R}^{m\times n}$

Figure 4: KalmanNet RNN block diagram (architecture #1). The architecture comprises a fully connected input layer, followed by a GRU layer

2.3. Kalman Net - #2 Architecture

The second architecture tries to mimic the Kalman filter algorithm structure more the previews KalmanNet, it uses separate GRU blocks for each of the tracked metrics (second-order statistical moments). The division of the architecture into separate GRU cells and FC layers and their interconnection is illustrated in figure 5, the network composes three GRU layers, connected in a cascade with dedicated input and output FC layers. The first GRU layer tracks the unknown state noise covariance Q. Similarly, the second and third GRUs track the predicted metrics Σ, R. The GRUs are interconnected such that the learned Q is used to compute Σ which in turn is used to obtain S, while both metrics involved to get the Kalman Gain. This architecture is more directly tailored towards the formulation of the classic Kalman Filter workflow and the

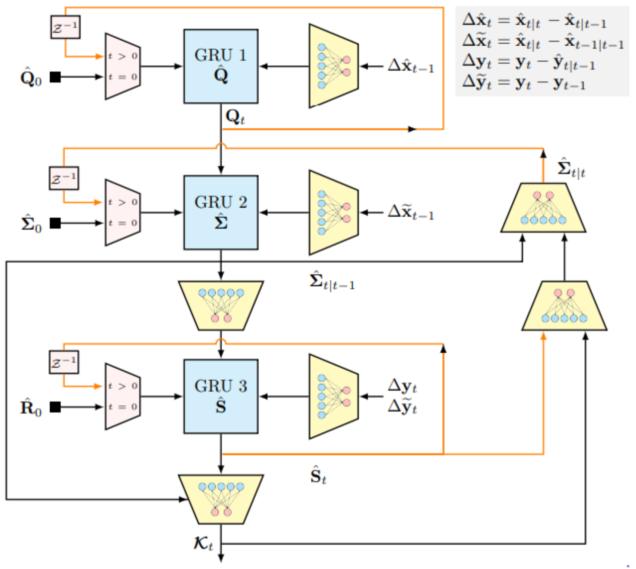
$\Delta\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t|t-1}$
$\Delta\tilde{\mathbf{x}}_t = \hat{\mathbf{x}}_{t|t} - \hat{\mathbf{x}}_{t-1|t-1}$
$\Delta\mathbf{y}_t = \mathbf{y}_t - \hat{\mathbf{y}}_{t|t-1}$
$\Delta\tilde{\mathbf{y}}_t = \mathbf{y}_t - \mathbf{y}_{t-1}$
GRU 1 $\hat{\mathbf{Q}}$
GRU 2 $\hat{\mathbf{\Sigma}}$
GRU 3 $\hat{\mathbf{S}}$
$\hat{\mathbf{Q}}_0$
$\hat{\mathbf{\Sigma}}_0$
$\hat{\mathbf{R}}_0$
$\mathbf{Q}_t$
$\hat{\mathbf{\Sigma}}_{t|t}$
$\hat{\mathbf{\Sigma}}_{t|t-1}$
$\hat{\mathbf{S}}_t$
$\mathcal{K}_t$
$\Delta\hat{\mathbf{x}}_{t-1}$
$\Delta\tilde{\mathbf{x}}_{t-1}$
$\Delta\mathbf{y}_t$
$\Delta\tilde{\mathbf{y}}_t$

Figure 5: KalmanNet block diagram (architecture #2). The input features are used to update three GRUs with dedicated FC layers

operation of the approach compared with the simpler first architecture. As a result, it requiring less trainable parameters the second architecture utilizes merely $2.5 \cdot 104$ parameters for same result as the first architecture. The second architecture improve that there is an advantage for cascaded implementation, this is why we suggest using transformer which uses blocs of attention inside and not attention alone. Moreover, the noise metrics are rebuilt for every step according to the last measurement and hidden states, in the transformer decoder we use as input the last few states output so the self-attention will be able to use it in order to model the noise from previews outputs.

3. Data

training and examine the model was made by using the data model generator from KalmanNet origin work. The data is synthetic and generated by creating the Lorenz-Attractor trajectory sample locations in the trajectory (decimated) and adding noise to the results. The noisy data considered as the measurements while the decimated data is the ground true trajectory our model needs to fit. In order to see the differences and compere this work to related works we will use the same data and model that was used at KalmanNet article and shared over their git account.

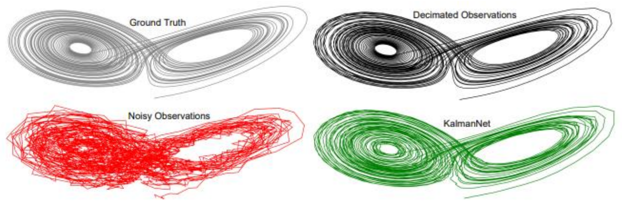

Figure 6: Lorenz attractor with sampling mismatch (decimation), T = 3000.

4. Methods

Our transformer approach to Improve Kalman Filter is using the conclusions from primer related work, first we will use a Hybrid-model that retain the classic Kalman Filter structure and use the prior knowledge as motion model. The work will not implement static noise metrics and as so the noise metrics will be built by the preview's measurements and states. Optimize the running time is implemented by the parallel quality of the transformer structure. The cascade blocs are also implemented by separating the previews measurement and outputs. Related to the classic Kalman Filter, Q is generated from the previews outputs as it's related to it and R which is the measurements noise, generated from the last number of measurements, Σ is calculated by both, measurements and algorithm final state outputs, then the Kalman Gain is the transformer fully connected layer output. As the attention mechanism work we assume that by looking at previews measurements and output states that was created by more steps backward in time, the model can calculate the noise. The translation can be compared to the same way that human will solve the Kalman Filter problem. Measurements first, when human have map of measurements and ask to point over the real state of the ground true he calculates the noise of the measurements and point at reasonable state somewhere in the middle of the measurements, this approach explained in figure 7. Output states, the model noise Q, can be calculated from looking at the last states, same process as calculating the noise from measurements we keep track the noise estimation and its standard deviation so if the last states are smoother we will consider it more trustable (lower model noise) and versatile states will be considered noisier.

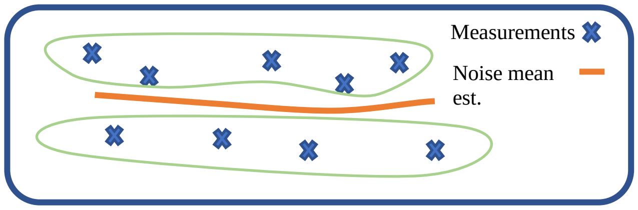

Figure 7: Human mean noise estimation by using measurements over time. Part of the process is grouping the measurements for upper and lower groups as expected from Self-Attention mechanism.

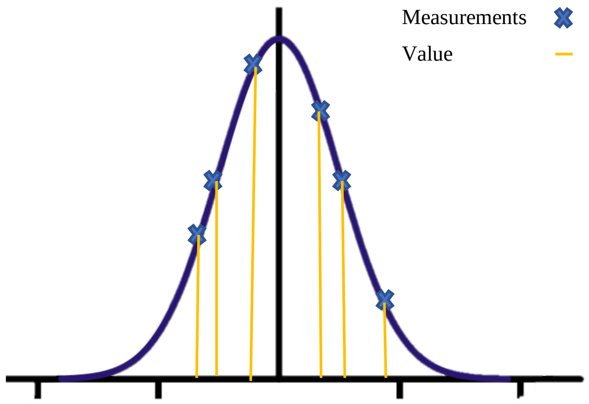

Figure 8: Excepted transformer noise detection. Using the self-Attention will separate the measurement into upper and lower groups, we except the values to be the estimated distance from the ground true state.

Position encoding and inputs embedding, in order to have low number of parameters we didn't use embedding for the transformer inputs. Moreover, we didn't find justification to do so according to the method we want the model to calculate the noises. But, the use of positional embedding was made so the model will get the sequence of the inputs. While in the original use of the position embedding at first

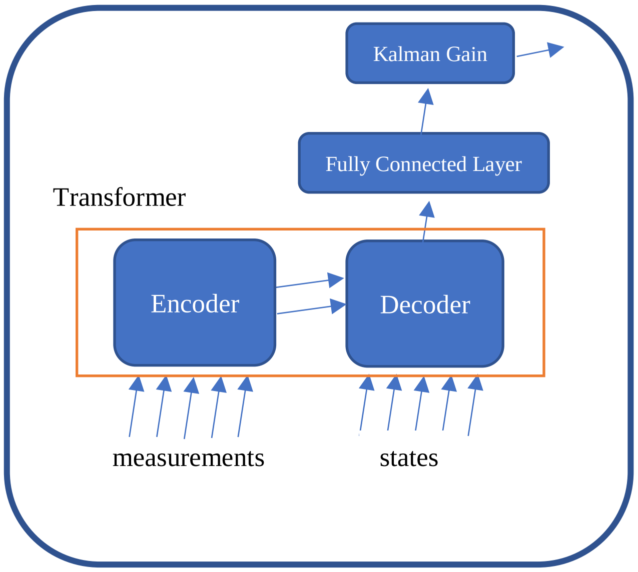

Figure 9: Model architecture

the values were added to the embedded vectors. Because we have low dimension at each state vector (size of three values) we decide to concatenate the position embedding vector of size four to the state vector, it helped the model to extract the position for each state and increase the parameters in the transformer model so it have more calculation power and can extract more data from the inputs regarding the fact that there was no use in embedding layer to the model inputs.
Markov assumptions, was implemented in this work by looking at the same number of previews state which we consider "One Last Step" that consider of the last 5 states. Moreover, to model that the history is infinite we consider the target as no movements before measurements, at the first One Last Step we zeroed the states and measurements differences.
Model inputs specifies to six-time steps to the past since we noticed that it's the minimum measurement we needed to understand the trend of ground true trajectory, in order to minimize the number of learning parameters we choose no more than five steps although we assume that by increasing the inputs number the results will get better. By using GRUs as in the related work we might get information from late in history measurements the classic Kalman Filter suggest that the last output will be depends only over the last measurements regarding the history. Considered that we want the model to figure out the measurements and states noise we choose the last six-time steps.
Weights initialization was applied by using the Xavier uniform distribution according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a uniform distribution.

Exploding gradient, to deal with it we used phased training. followed the implementing of the gradient clip by norm-2 The model was trained in multiple phases. Each phase trained over different Hyper-parameters includes: learning rate, gradient clip value, TBTT. While the start of the training we got the exploding gradient phenomenon for every try or change in the learning rate and gradient clip value, then we decide to try a version of TBTT (truncated backpropagation through time) method by computing only the first number of states in each trajectory. Every phase of the training had a graduate number of states to consider, learning rate and gradient clip value. Regularization was used by dropout and weight decay.

5. Experiments

In order to check the model performance, it was trained over the training data in 20 batches. During the training TBTT was implemented. The model was trained for 100 epochs for each step with different number of outputs to backpropagate loss starts at 3 up until full trajectory length. Learning rate starts at 0.001 and If in the first 10 epochs in every step had no improvement in validation loss, the learning rate was multiplied by 0.1. Gradient normalized value was changed twice at each step from 0.4 to 0.2. Loss function used MSE (Mean Square Error).

Comparison of the method was made to classical Kalman filter and KalmanNet 2# architecture. Both algorithms were in the KalmanNet Github and was taken from there, we were trying to train the KalmanNet model but it didn't have improvements to the published already trained model.

Test data includes 360 parts off the Lorrentze-Attractor and each part is a trajectory of 100 measurements. Same as training, the data was taken from the KalmanNet published code, and was made by the DataGenerator script. Lorrenze-Attractor parameter and the movement-model (matrices F, H) were the same as in the KalmanNet publish and were used to build the network, same as building the KalmanNet network.

Final results over the test data can be seen in the figures below:

Model Name	Number of Learnable Parameters	MSE Score [dB]
EKF (classic)		−4.322
KalmanNet	23928	−5.308
TransformerNet	4665	−5.624

Figure 10: Models Summary results. EFK, KalmanNet, TransformerNet comparison for MSE score and number of trainable parameters.

In figure 10 we can notice that the transformer approach we implemented, TransformerNet bet the original RNN approach KalmanNet and the enormous difference in the count of learnable parameters, it affects the speed of algorithm calculation which is considered important in the use of Kalman Filter for real time uses.

At the visualizations below, we show the model outputs over a sample of trajectories. Red dots symbol the noisy measurements that input to the model every time step. Green dots Symbol the model outputs, Blue line is the ground-truth trajectory that the model needs to estimate.

Straight line trajectory has the best estimation as expected because it's the easiest pattern, moreover by the visualization we can notice that we have bias for areas with measurement that are not isotropic from all the directions around the ground truth line.

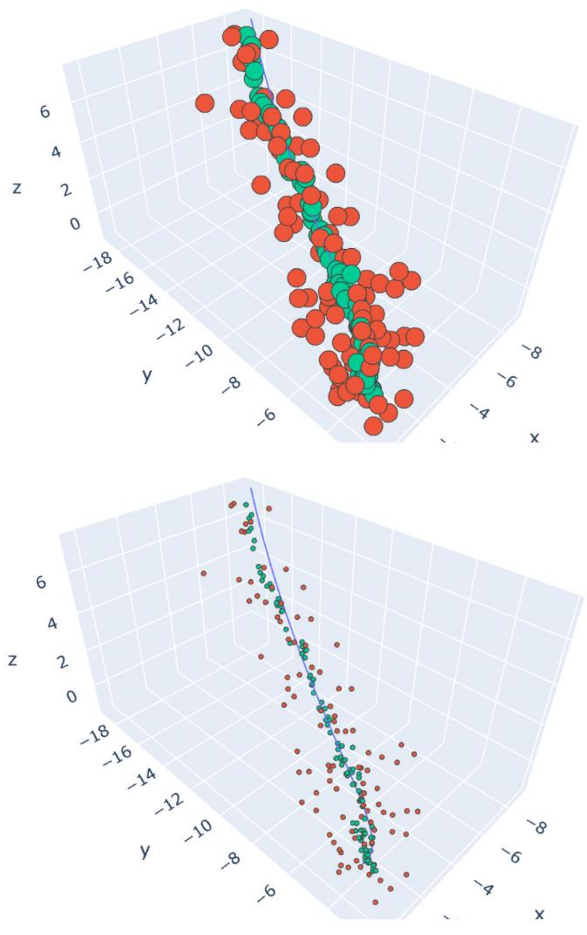

Figure 11: Straight line trajectory results.

Soft-Turn, as figure 12 shows while estimating soft-turn the model preform decrease but it is clear that the outputs coverage to the ground truth and the outputs have smoothed the measurements in opposite to figure 13 where the is a Sharp-Turn. While having the sharp turn we notice that the model underperforms and wasn't able to smooths the measurements well. We suggest that one of the reasons is that the model has the same standard deviation of noise over all the axis and by turning sharply the measurements can match the sharp turn measurements but also continuation in the same direction.

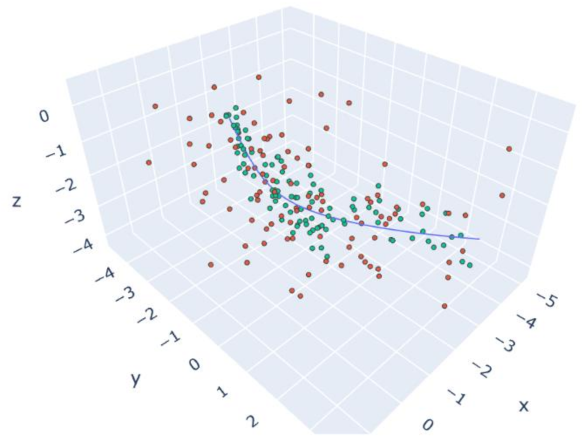

Figure 12: Soft-Turn trajectory results.

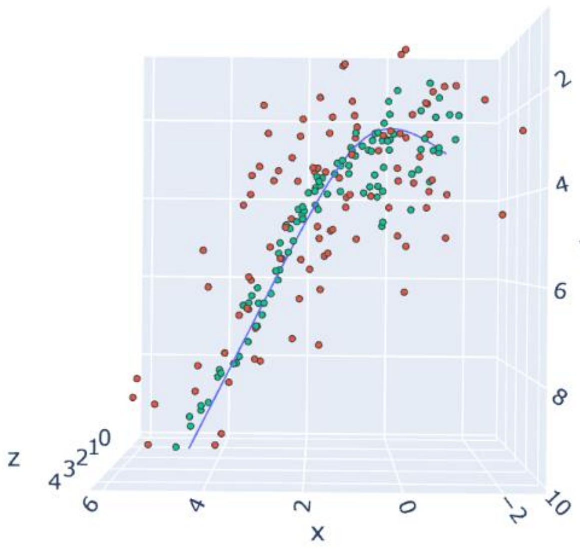

Figure 13: Sharp-Turn trajectory results.

6. Conclusions

During this article we have seen that using transformers for improving Kalman Filter can get the best results over previous methods, moreover the transformers reduce the learning parameters significantly and are optimized for parallelized, in a result we created optimized better performance algorithm. For farther extensions we would examine the results of changing the memory size during high outputs covariance that can point low accuracy at situation as sharp-turns. For this extension we will increase the runtime of the model but we suggest that the results will get better.

References

[1] FirstName Alpher, Frobnication. *IEEE TPAMI*, 12(1):234–778, 2002.

[2] FirstName Alpher and FirstName Fotheringham-Smythe. Frobnication revisited. *Journal of Foo*, 13(1):234–778, 2003.

[3] FirstName Alpher, FirstName Fotheringham-Smythe, and FirstName Gamow. Can a machine frobnicate? *Journal of Foo*, 14(1):234–778, 2004.

[4] FirstName Alpher and FirstName Gamow. Can a computer frobnicate? In *CVPR*, pages 234–778, 2005.

[5] FirstName LastName. The frobnicatable foo filter, 2014. Face and Gesture submission ID 324. Supplied as supplemental material `fg324.pdf`.

[6] FirstName LastName. Frobnication tutorial, 2014. Supplied as supplemental material `tr.pdf`.

[7] https://arxiv.org/pdf/2104.02372v1.pdf

[8] https://arxiv.org/pdf/1511.05121.pdf

[9] https://arxiv.org/pdf/2107.10043.pdf

[10] https://reader.elsevier.com/reader/sd/pii/S0169207021000637?token=6738295B71A32738E3011113CBEB540C800CD1339DE07EE704E33C119117D1D78A0D555B3A67EB922910201BC1B892E5&originRegion=eu-west-1&originCreation=20220719130752 - Temporal Fusion Transformers for interpretable multi-horizon time series forecasting

[11] https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf - Attention Is All You Need