Sensor Fusion - MO869A

Final Term Project Report

Smartphone-user's Position Estimation using Data-driven Inertial Navigation Approach

Medhavi Mishra

20255462

2025-12-19

Table of Contents

1 Introduction

Data driven inertial navigation domain has gained significant relevance with increasing use of smartphones, smartwatches, hearables, and other everyday devices. Such devices contain sensors, such as Inertial Measurement Units (IMUs), that measure acceleration and angular velocity.

Traditional Inertial Navigation Systems (INS) suffer from unbounded integration drift, where small errors in the sensor measurements are compounded over time, leading to significant trajectory estimation errors. The research motivation is to overcome this drift and produce globally consistent, accurate, and real-time localization without relying on external signals like GPS, which are often unavailable or degraded in indoor environments or urban canyons. Consequently, a core objective has been the development of robust machine learning models that can directly map raw IMU sensor data to an accurate, low-drift velocity or position estimate.

The resulting research is foundational for numerous use cases, including seamless indoor/outdoor navigation, where the system transitions smoothly when GPS is lost; context-aware applications in smart homes and retail, where a device's precise, power-efficient location is needed; first responder tracking in hazardous, GPS-denied environments; and ubiquitous personal health monitoring, where precise movement and activity tracking are required regardless of device placement.

2 Problem Statement and objective

The central problem to be addressed is the unbounded accumulation of errors (drift) in displacement and velocity estimates generated by purely data-driven, single-sensor Inertial Measurement Unit (IMU) systems, particularly those that use deep learning to regress velocity (like RoNIN). While Deep Learning models (Temporal Convolutional Networks, Transformer) excel at pattern recognition and compensating for non-linear IMU errors, they lack the state-space modeling and optimal recursive estimation capabilities needed to maintain long-term stability and optimally fuse the learned velocity with a dynamic motion model.

Hence, the overall objective of this work is to test different deep learning–Kalman filtering framework for inertial velocity regression that reduces drift and improves trajectory accuracy. To this end, the main goals of this project are as follows:

1. To integrate a Kalman Filter with the temporal convolution network and fuse network predictions with a motion dynamics model, aiming for smoother trajectories and reduced transient errors.

2. To investigate transformer architecture for inertial velocity regression and compare performance gains when fusing a Kalman filter based dynamics model.

3. To compare the hybrid model (feature fusion of TCN and Transformer) and study the performance gains when integrated with Kalman filter.

3 Methodology

3.1 Dataset

The RoNIN Dataset [1] is a large-scale benchmark specifically designed for "in-the-wild" inertial navigation. It comprises over 40 hours of raw IMU data (accelerometer and gyroscope) collected from 100 diverse human subjects performing natural, unconstrained daily activities, critically allowing for arbitrary smartphone placement (e.g., in a pocket, held in hand, or in a bag). The IMU data of the accelerometer and gyroscope (a_x, a_y, a_z, g_x, g_y, g_z) obtained at 200 Hz, is used as input to predict velocity vectors (v_x and v_y) by the models. Ground-truth 3D trajectories of the body were obtained from a body-mounted 3D-tracking phone (Asus Zenfone AR using Tango/visual–inertial SLAM). The ground-truth position data is then differentiated over time to yield the corresponding ground-truth velocity vector.

The test sets are of two types: seen subjects (human subjects' samples included during training) and unseen subjects, which are used for performance evaluation.

However, a subset of the RoNIN dataset is used for this work as only 50% of this dataset is released to the public due to security concerns.

3.2 Model Architecture

1. Temporal Convolution Network (TCN) is a 1D convolutional network designed for sequences / time series instead of images. Here, TCN is used with receptive field size

[1]https://www.frdr-dfdr.ca/repo/dataset/816d1e8c-1fc3-47ff-b8ea-a36ff51d682a

is 253 (kernel size=3, num_channels=6), input window size of 400 and channels as [32, 64, 128, 256, 72, 36], which are design choices. It leverages causal convolution which means that at time step t, the output can only depend on time steps $\leq t$ (no "peeking into the future").

2. TCN+KF: Learnable Kalman filter parameters are introduced in the baseline model. The goal is to learn temporal features using 1D convolutions that are causal (only use past and current time steps, not future ones). Here, TCN predicted velocities are treated as noisy measurements by the kalman filter which refines velocity and then integrates it to obtain position (dt = 0.005s).

3. Transformer (TFM): A six-layer transformer encoder with 4 heads alongwith causal masking (no "peeking into the future") is used to construct the TransformerSeqNetwork. In this network, maximum length for positional encoding is 512 and the input window size is 400, with a history focus of 200.

4. TFM+KF: Kalman filter parameters (process noise and measurement noise covariances) are learned while the transformer model is trained.

5. Hybrid: In this model, features obtained from both TCN and TFM (dim 36 channels from each) are fused during training. The fusion method used is concatenation, then a linear layer is applied which outputs the predicted velocities (v_x and v_y).

6. Hybrid+KF: Kalman filter parameters (process noise and measurement noise covariances) are learned while the hybrid model is trained.

3.3 Kalman Filter Equations

State-Space Model

State vector:

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix} \tag{1}$$

System Matrices

State transition matrix (constant-velocity model):

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Measurement matrix:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Process noise covariance:

$$\mathbf{Q} = \begin{bmatrix} q_{pos} & 0 & 0 & 0 \\ 0 & q_{pos} & 0 & 0 \\ 0 & 0 & q_{vel} & 0 \\ 0 & 0 & 0 & q_{vel} \end{bmatrix} \tag{4}$$

Measurement noise covariance:

$$\mathbf{R} = \begin{bmatrix} r_{vel} & 0 \\ 0 & r_{vel} \end{bmatrix} \tag{5}$$

Prediction Step

Predicted state estimate:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} \tag{6}$$

Predicted error covariance:

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q} \tag{7}$$

Update Step

Measurement residual (innovation):

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \tag{8}$$

Innovation covariance:

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R} \tag{9}$$

Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1} \tag{10}$$

Updated state estimate:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{y}_k \tag{11}$$

Updated error covariance:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1} \tag{12}$$

Measurement Model

The measurement vector contains the model-predicted velocities:

$$\mathbf{z}_k = \begin{bmatrix} v_x^{model} \\ v_y^{model} \end{bmatrix}_k \tag{13}$$

3.4 Loss Function

For training vanilla models, GlobalPosLoss is used, as implemented for RoNIN TCN[2]. It applies mean squared error loss between target position, and position derived from cumulating predicted velocity over a time window. (*refer to GlobalPosLoss in main.py*)

For training vanilla+KF models, KFFilteredPosLoss is introduced. This loss function combines a differentiable Kalman Filter with a windowed position error to train the velocity prediction model end-to-end. It takes the predicted velocities and runs them through a Kalman filter, the KF smooths noisy velocity predictions into cleaner position estimates. It applies a mean-squared error loss between ground truth position, and position derived by integrating predicted velocity over 0.005s (200 Hz). (*refer to KFFilteredPosLoss in main.py*).

Notably, the Kalman gain K updates the mean state depends on parts of P^{-1} that are independent of q_{pos} (because H only looks at the velocity block). Therefore, the updated

[2]Herath, Sachini, Hang Yan, and Yasutaka Furukawa. "Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods." 2020 IEEE international conference on robotics and automation (ICRA). IEEE, 2020

means p_k, v_k do not depend on q_{pos} at all (no position measurements). This also implies that the loss function is independent of q_{pos}, hence it remains constant, while q_{vel} and r_{vel} are optimized.

3.5 Experimental details

Since pretrained weights of the baseline model RoNIN TCN were available, the TCN+KF model was trained for 100 epochs, while other models were trained for 200 epochs using NVIDIA RTX 5070 Ti 16 GB. Training time per epoch of the models are mentioned in the Table 1.

Table 1: Comparison of training time per epoch.

Model	**TCN**	**TCN+KF**	**TFM**	**TFM+KF**	**Hybrid**	**Hybrid+KF**
Time (s)	11.60	94.45	43.8	128.8	56.2	140.4

Inference

During inference, the learned KF parameters are extracted from the trained model and applied to refine the prediction and smoothen the trajectory reconstruction ($kf\_smooth\_velocity$ function in the $kf\_util.py$ script is called by main.py script). [3]

3.6 Performance Metrics

For position evaluations, two standard metrics are used (*refer to metric.py*):

1. Absolute Trajectory Error (ATE)(m) is calculated as the average Root Mean Squared Error (RMSE) between the estimated and ground-truth trajectories as a whole.

2. Relative Trajectory Error (RTE) (m) is defined as the average RMSE over a fixed time interval i.e 1 minute here.

[3]For reference, all required python scripts are provided in SF_FinalTermProject_20255462_Medhavi.zip

4 Results and analysis

4.1 Quantitative Results

The values of ATE and RTE across different experiments are reported in Tables 2-4.

As shown in Table 2, TCN+KF achieved lower ATE on both Seen Subject and Unseen Subject test sets, indicating a slight enhancement in overall trajectory estimation accuracy. On the other hand, a slight increase in RTE with the addition of KF suggests that the KF integration might slightly worsen the model's consistency and local accuracy over short time intervals. A similar trend is observed in the case of TFM+KF, as shown in Table 3.

The Hybrid model, which combines aspects of TCN and TFM, shows the least pronounced changes with KF integration, particularly in RTE. As shown in Table 4, the Hybrid+KF model achieved lower ATE on both the Seen Subject (4.74 vs. 4.84) and Unseen Subject (5.82 vs. 6.02) test sets, maintaining the trend of KF improving overall trajectory accuracy. Howerver, the change in RTE was negligible or slight. On Seen Subject, RTE slightly decreased, indicating a marginal improvement in local consistency.

Table 2: Performance of TCN and TCN+KF

Test Set	Metric	TCN	TCN+KF
Seen Subject	ATE	4.78	4.57
	RTE	2.91	2.98
Unseen Subject	ATE	5.88	5.75
	RTE	4.095	4.28

Table 3: Performance of TFM and TFM+KF

Test Set	Metric	TFM	TFM+KF
Seen Subject	ATE	5.78	5.19
	RTE	2.96	3.22
Unseen Subject	ATE	7.84	7.59
	RTE	4.05	4.25

The integration of the Kalman Filter (KF) consistently leads to a reduction in ATE across all three model architectures (TCN, TFM, and Hybrid) and both test sets. This suggests that the KF is effective in smoothing and correcting long-term global drift in the estimated trajectories.

Table 4: Performance of Hybrid and Hybrid+KF

Test Set	Metric	Hybrid	Hybrid+KF
Seen Subject	ATE	4.84	4.74
	RTE	2.75	2.73
Unseen Subject	ATE	6.02	5.82
	RTE	3.87	4.01

Furthermore, for the TCN and TFM models, the KF integration results in a consistent increase in RTE. This trade-off suggests that the KF's smoothing effect, while beneficial for overall position accuracy, may slightly degrade the model's ability to maintain high local consistency and short-term accuracy. The Hybrid model shows the most balanced result, with marginal RTE changes alongside ATE improvement.

Overall, the TFM model received the largest ATE benefit from the KF, while the Hybrid model demonstrated the most stable performance across both metrics.

4.2 Qualitative Results

A lower ATE means the estimated path is closer to the true path while a lower RTE means the estimated motion is locally more accurate and consistent over short periods. Figures 1 to 12 illustrate ATE and RTE values alongwith errors in v_x and v_y estimation during inference of sample s across the test sets using different models.

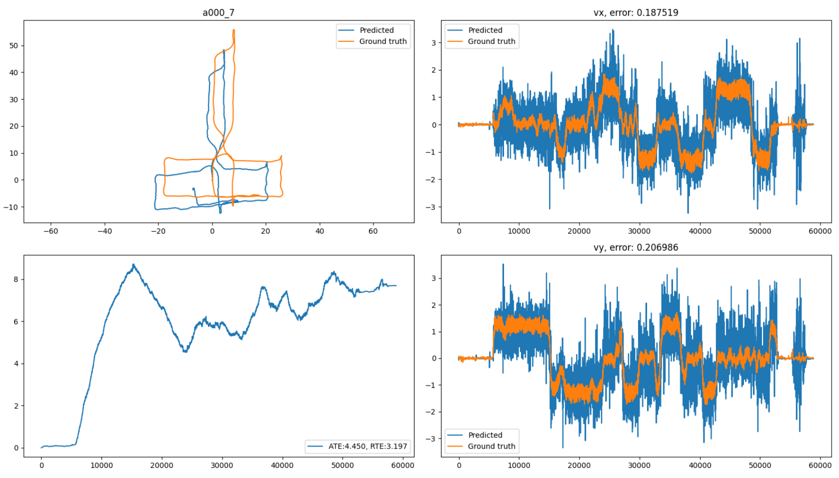

Figure 1: TCN without KF on seen subject sample a000_7

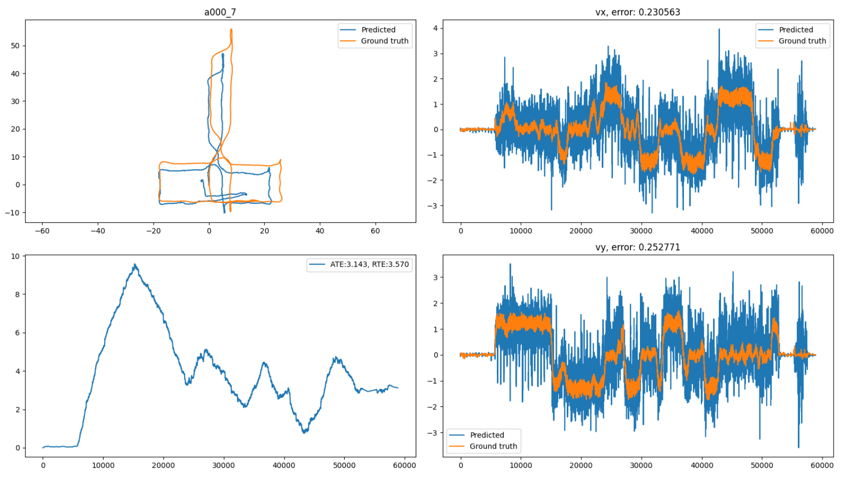

Figure 2: TCN with KF on seen subject sample a000_7

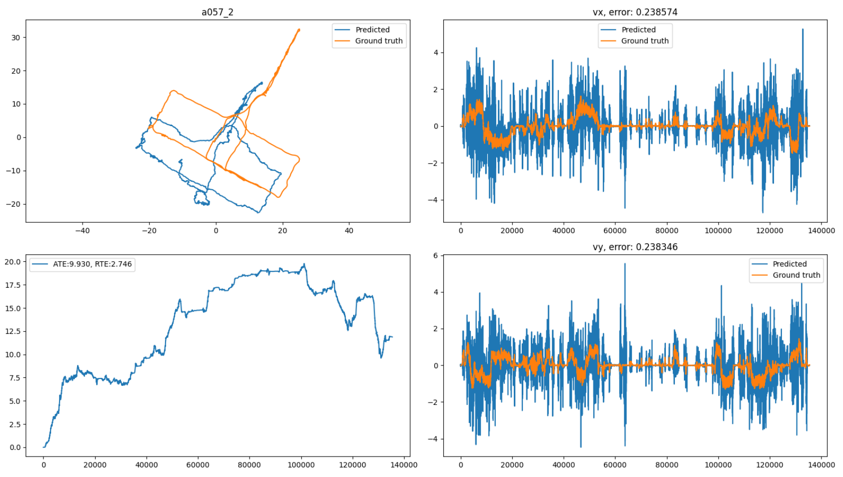

Figure 3: TCN without KF on unseen subject sample a057_2

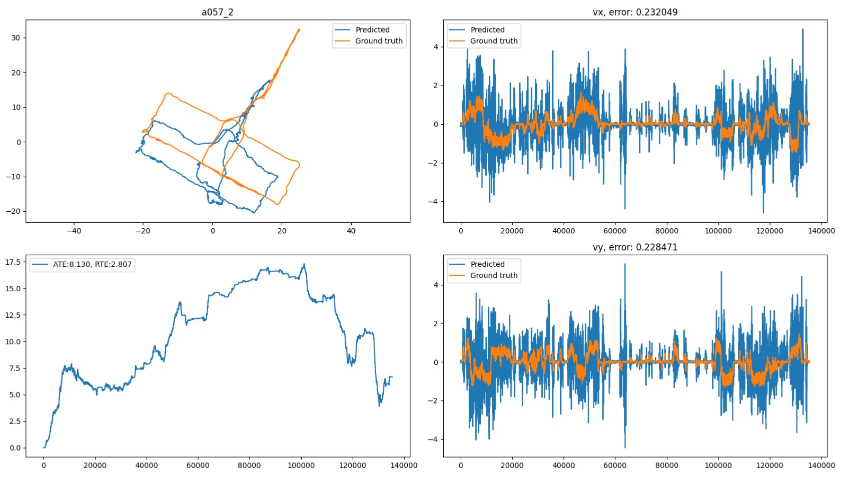

Figure 4: TCN with KF on unseen subject sample a057_2

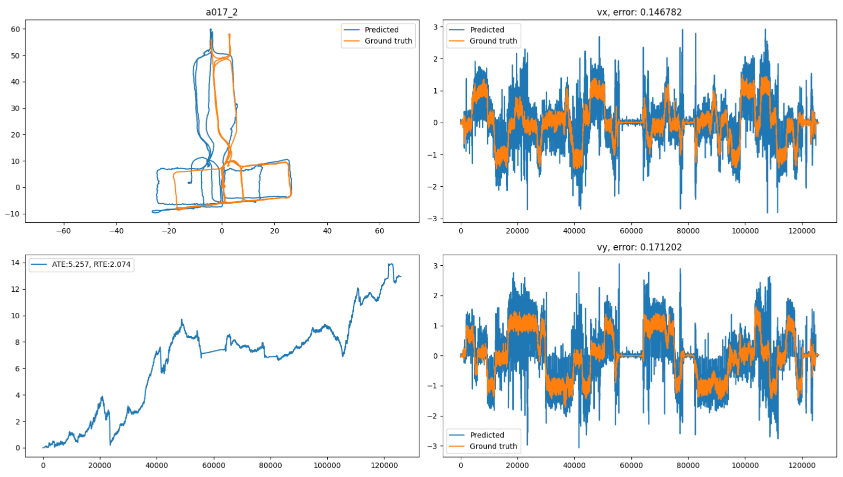

Figure 5: TFM without KF on seen subject sample a017_2

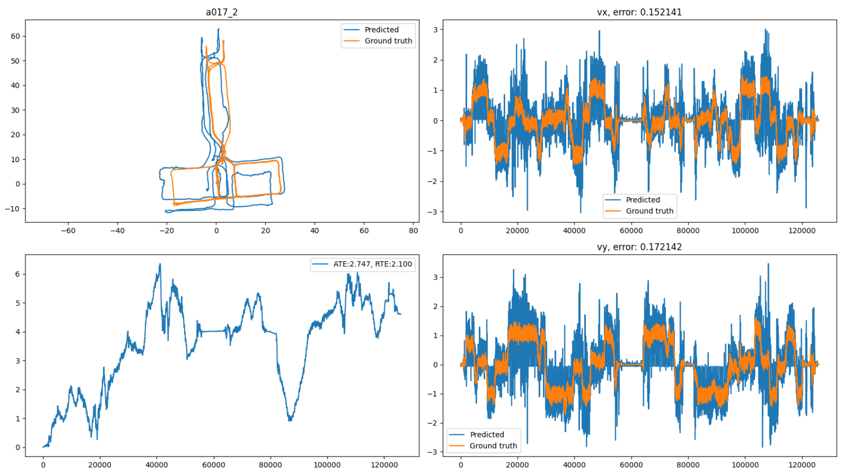

Figure 6: TFM with KF on seen subject sample a017_2

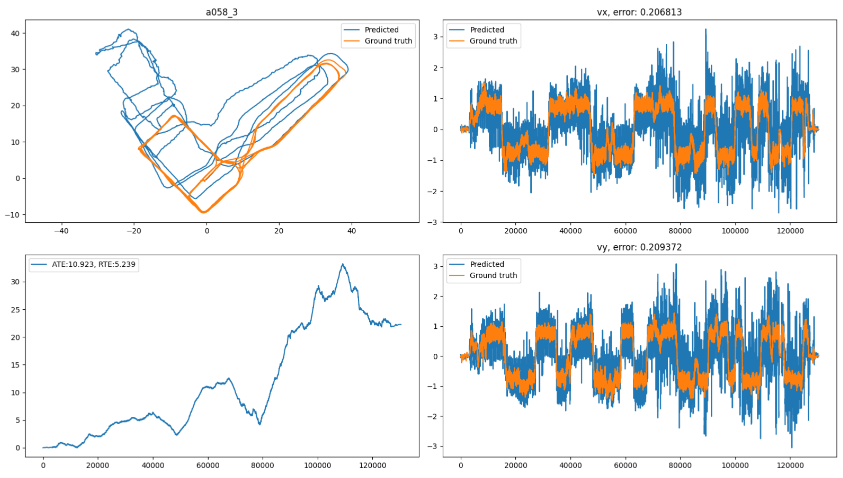

Figure 7: TFM without KF on unseen subject sample a058_3

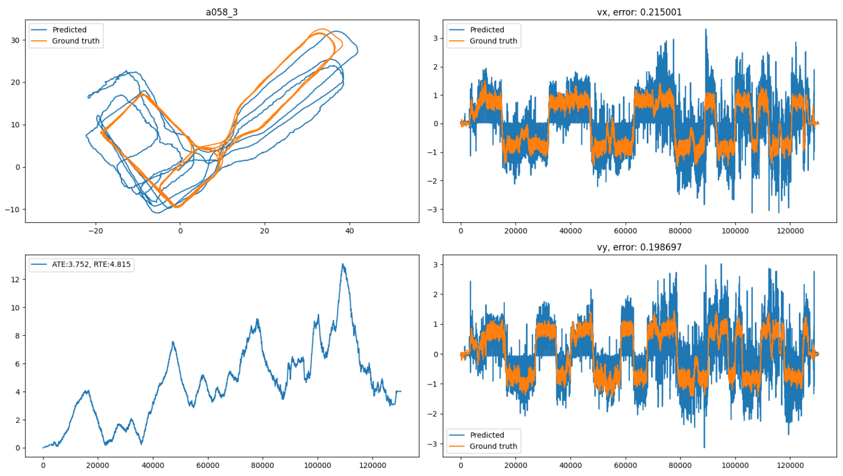

Figure 8: TFM with KF on unseen subject sample a058_3

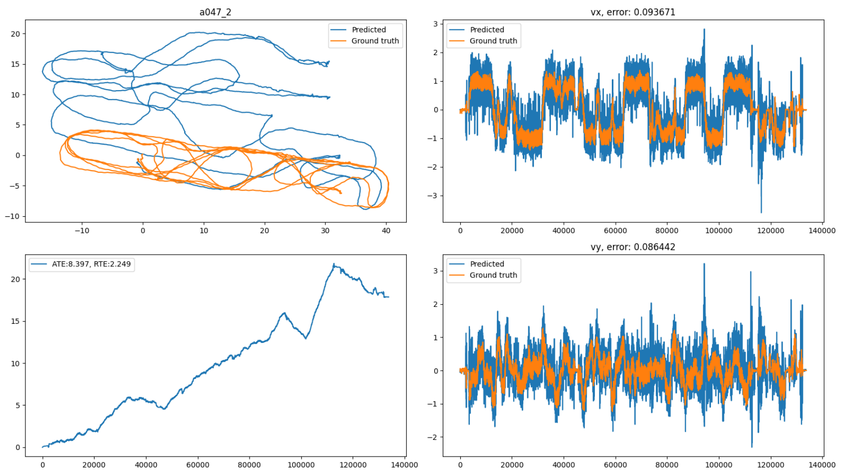

Figure 9: Hybrid without KF on seen subject sample a047_2

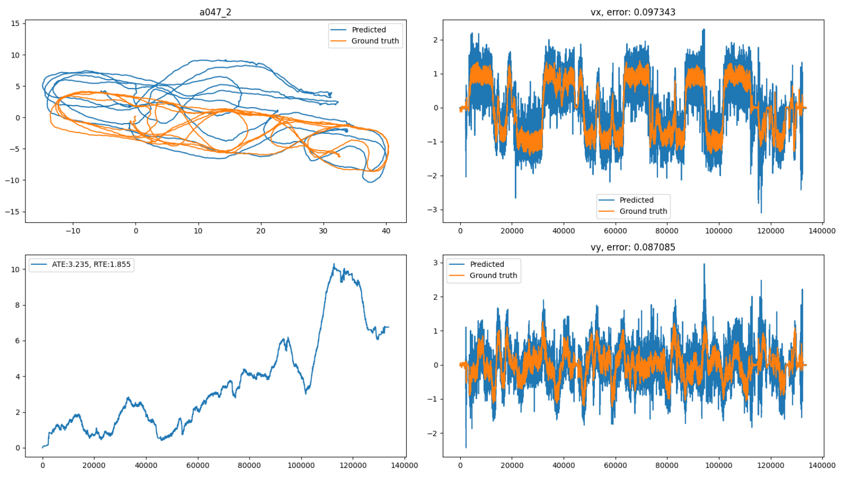

Figure 10: Hybrid with KF on seen subject sample a047_2

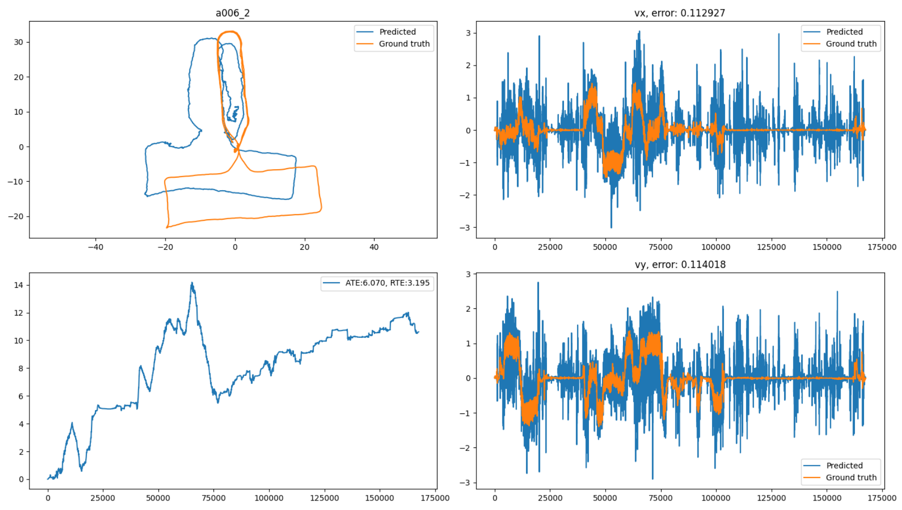

Figure 11: Hybrid without KF on unseen subject sample a006_2

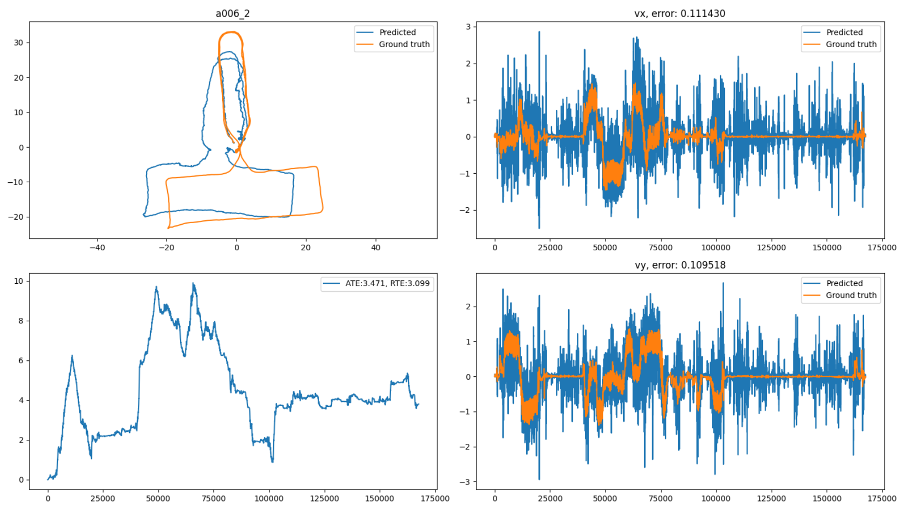

Figure 12: Hybrid with KF on unseen subject sample a006_2

5 Conclusion

The Hybrid model demonstrated the most robust baseline performance, achieving the lowest RTE values (2.75 for seen, 3.87 for unseen subjects), and with KF integration (2.73 for seen, and 4.01 for unseen). After KF integration, TCN maintained competitive ATE performance on seen subjects (4.57) and unseen subjects (5.75), while Hybrid+KF achieved the best overall balance between ATE and RTE metrics.

During inference, the KF acts as a post-processing smoother/corrector applied to the neural network's output. It leverages a dynamic model of motion to filter out high-frequency noise and long-term drift from the network's prediction. Kalman Filter integration presents a trade-off between absolute and relative trajectory accuracy. As demonstrated by results, better overall global accuracy (lower ATE) comes at the expense of a slight degradation in local consistency (higher RTE) due to the smoothing effect. The appropriateness of KF integration depends on application-specific requirements, prioritizing either global trajectory accuracy or local motion consistency.