Network-based Inertial Navigation

S. G. Andersen, M. Bach, A. G. Golles, A. A. Sousa-Poza & K. Tavanxhi

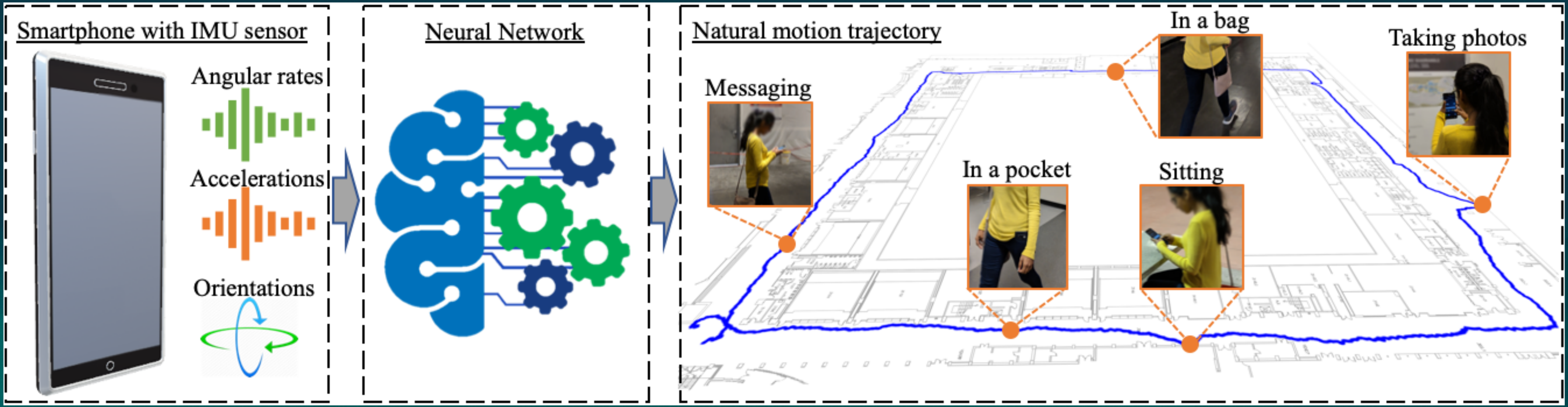
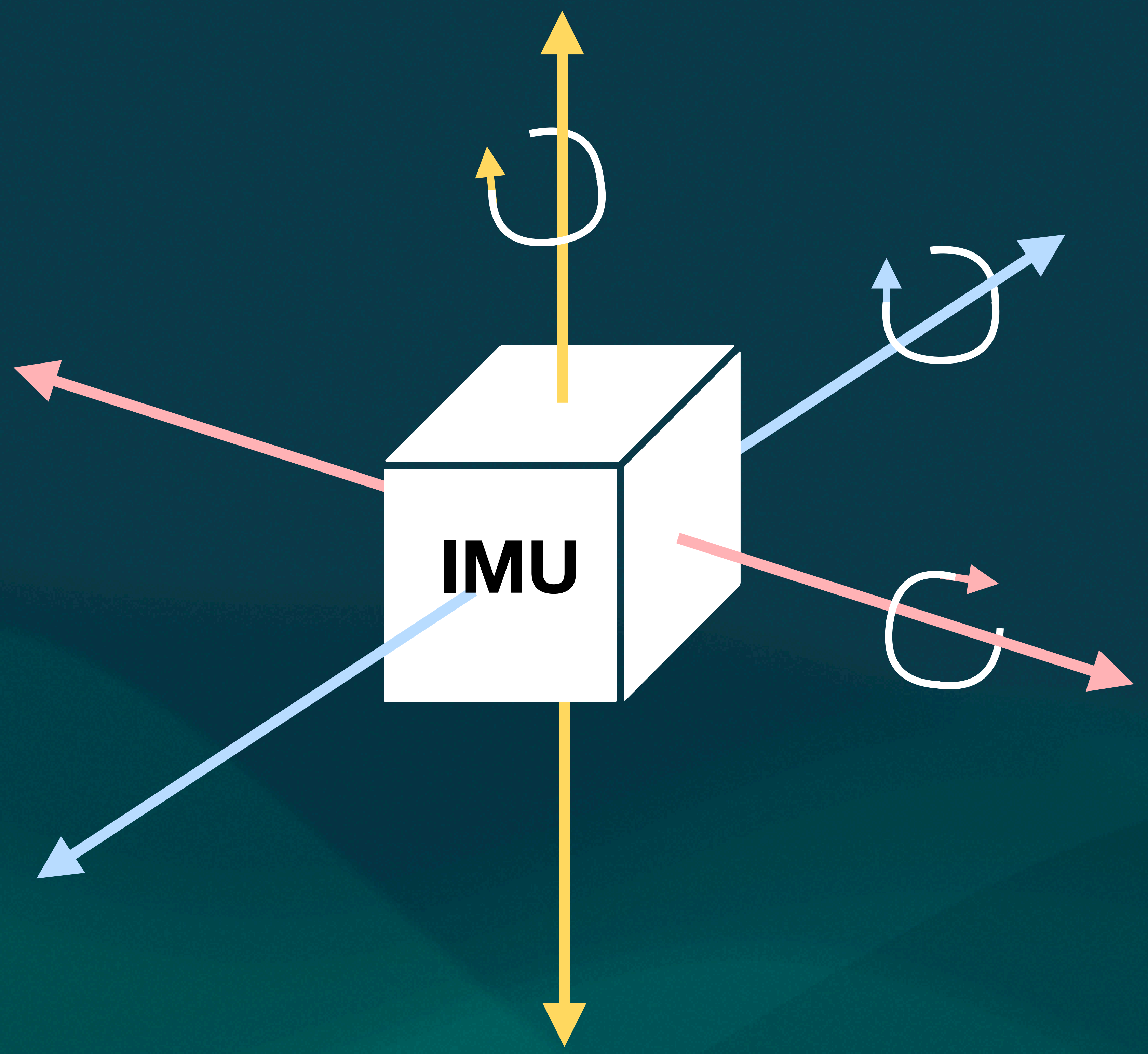


Illustration from https://ronin.cs.sfu.ca/

Agenda

- Data
- Typical IMU processing
- End-to-end architecture
- Feature extension (clustering)
- Gyroscope integration
  - Dynamic filter combination
- End-to-end network
- Neural accelerometer integration
- Conclusion



The Data

Raw → Synced

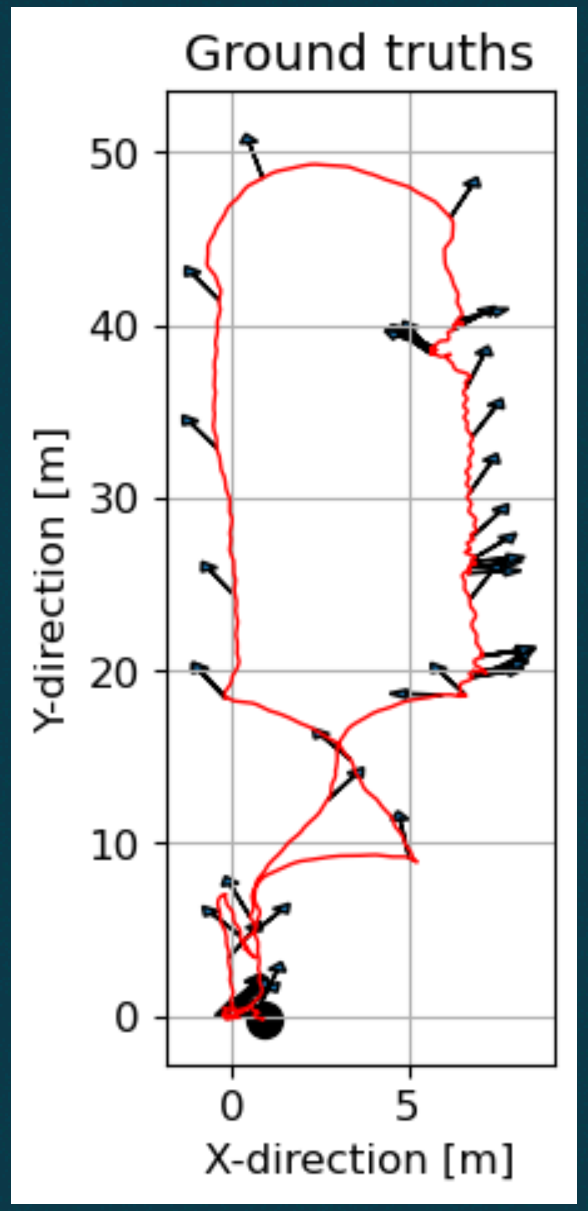
12 Features

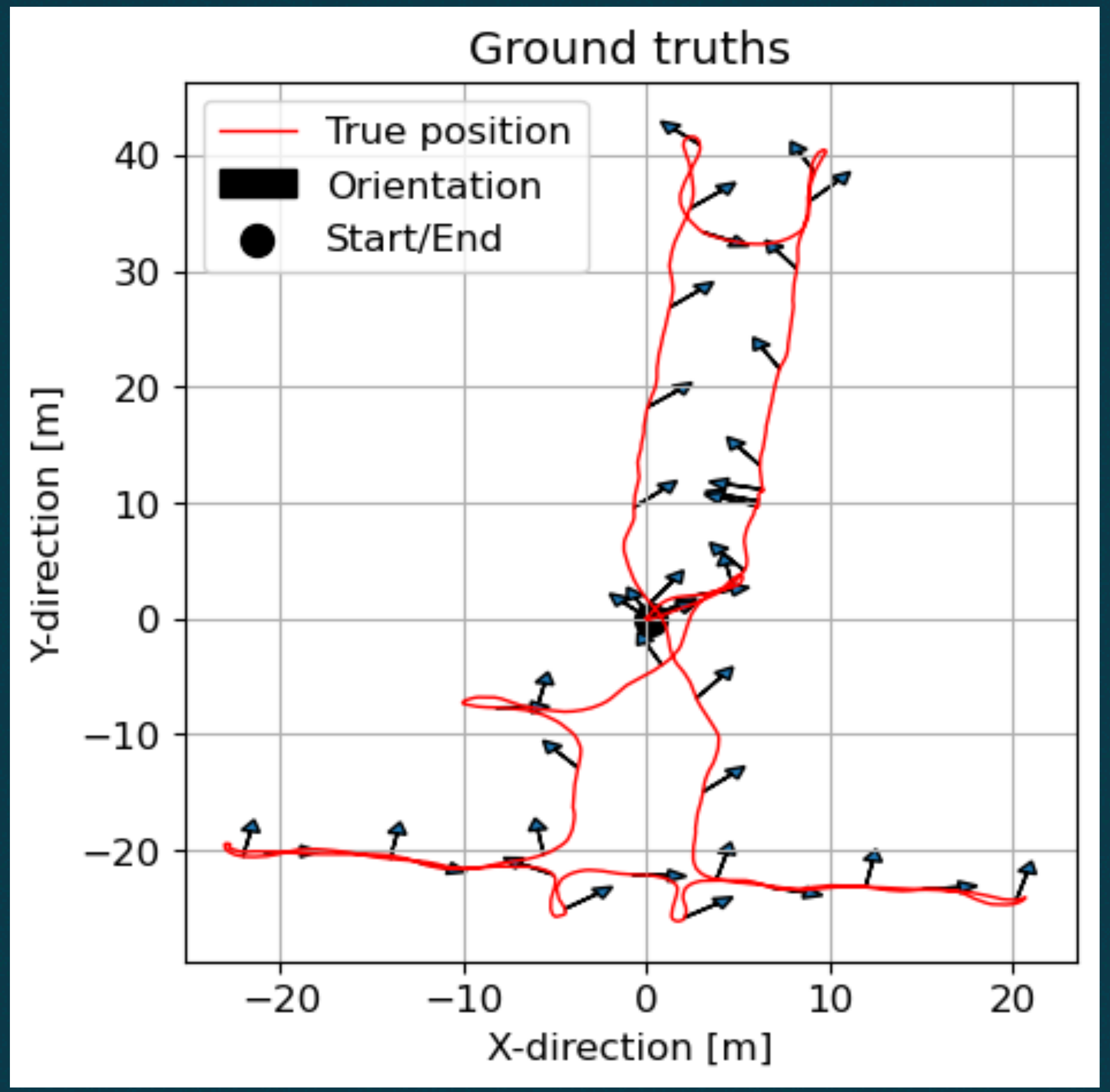
- Accelerometer
- Magnetometer
- Gyroscope
- RV

Pose

2 Ground truths

- True position
- True orientation





100 recorded and synced runs, with 50.000 - 250.000 data points at 200Hz.
Plot displays true position and orientation for sample runs.

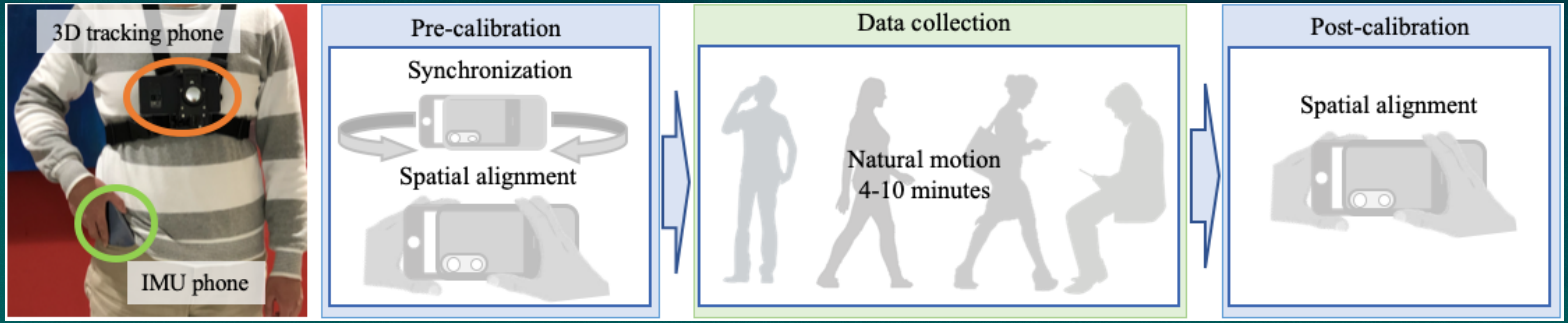
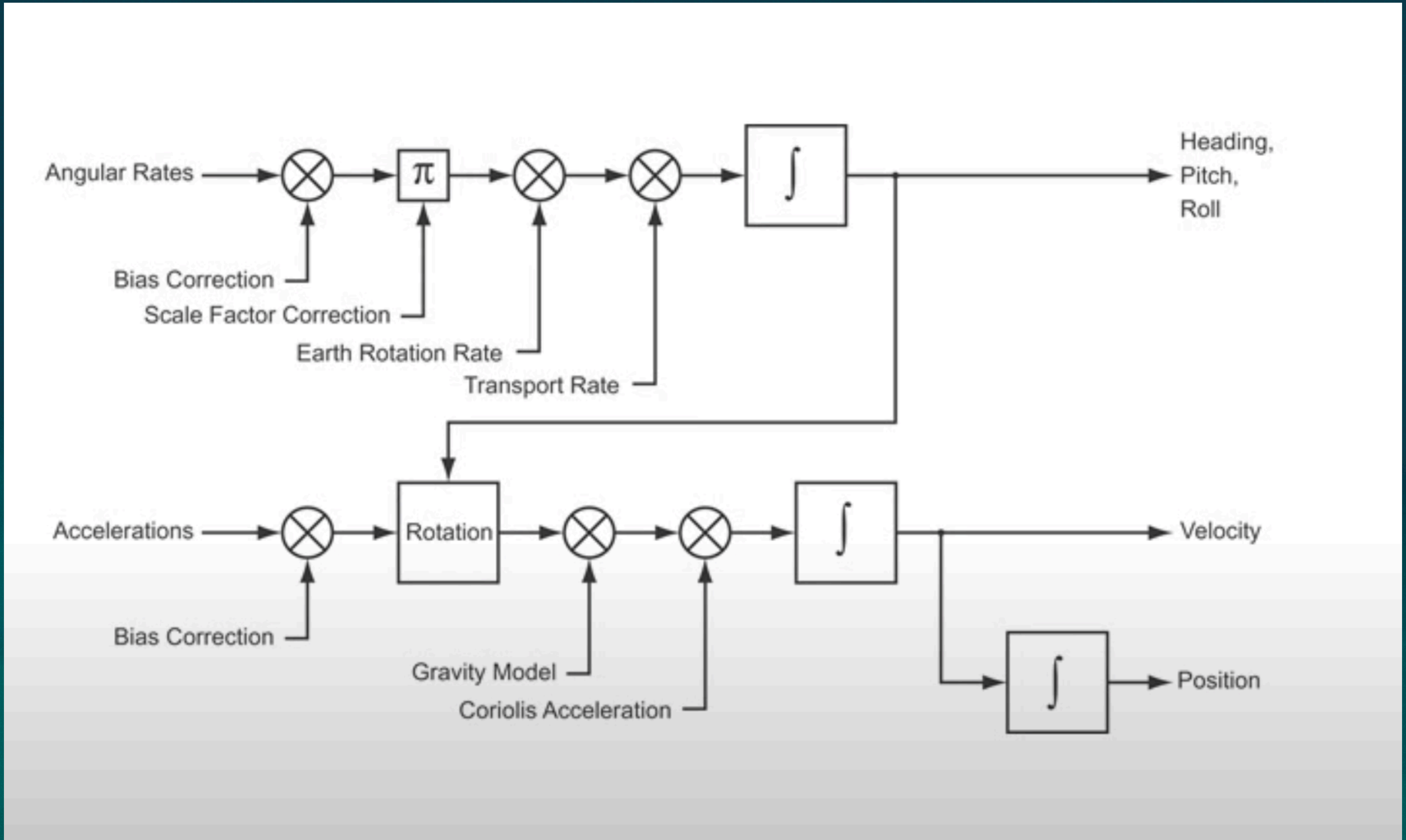
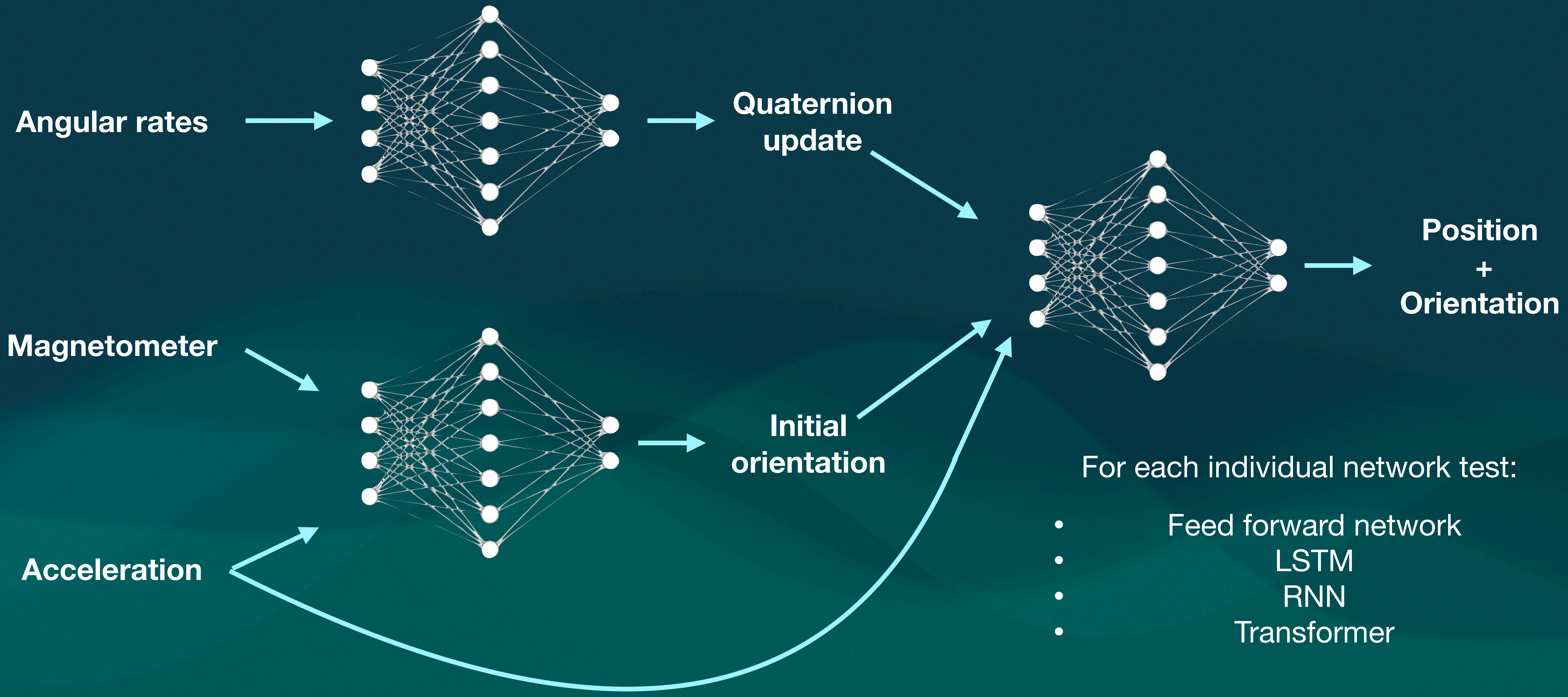


Illustration from https://ronin.cs.sfu.ca/

Naïve IMU processing (double integration)



Network following analytical path



For each individual network test:

- Feed forward network
- LSTM
- RNN
- Transformer

Inertial Navigation Neural Network

Pros and cons

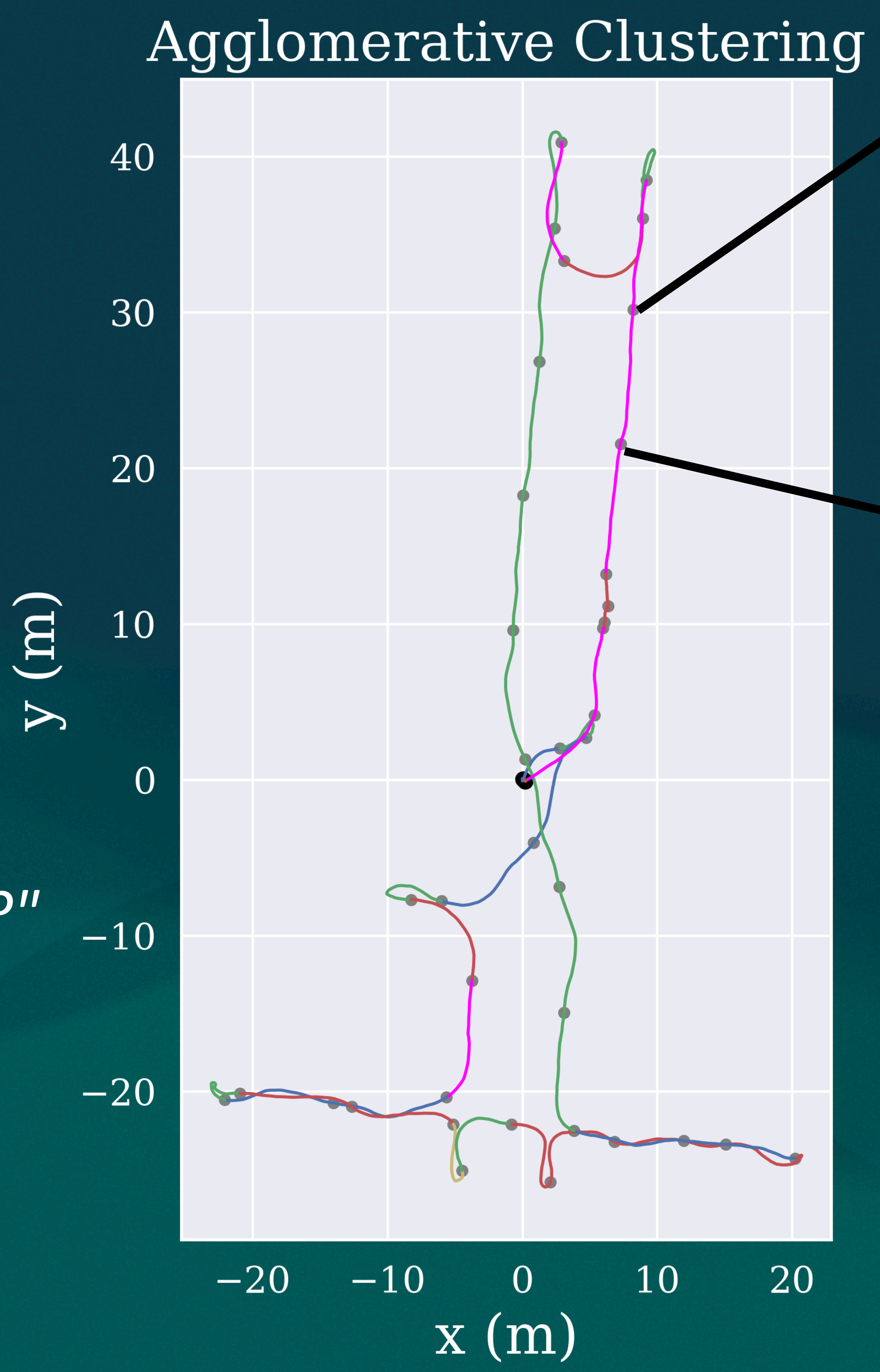
	Analytical approach	Neural Network
Speed	Fast	Maybe slow
Testing requirements	Reliable without much testing	Need vigorous testing to guarantee reliability
Drift	Quadratic positonal drift caused by linear velocity	Increased stability. Since we can identify stand-still!
Perfomance given noisy data	Poor	Potentially stable

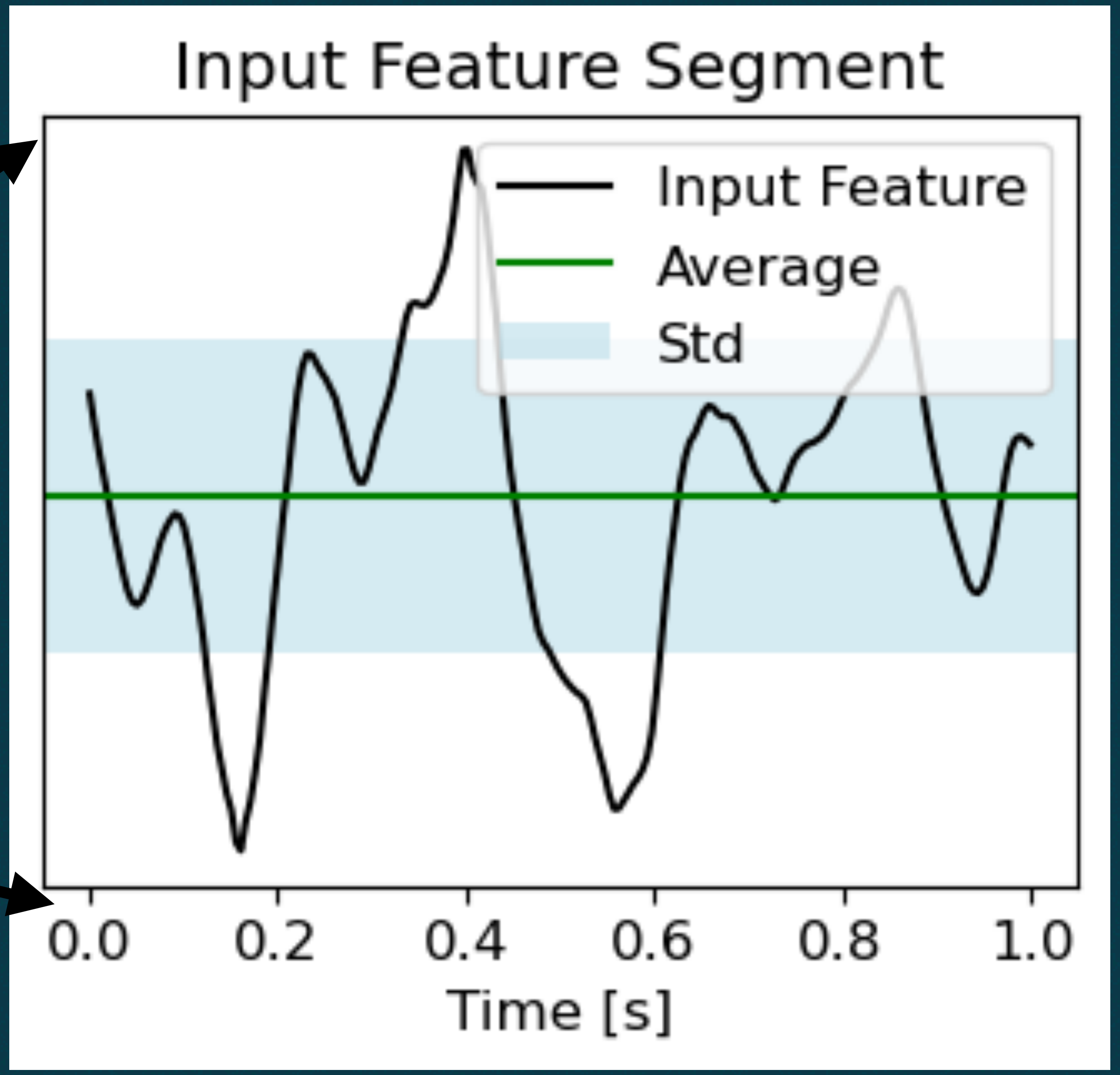
Feature Extension
Clustering

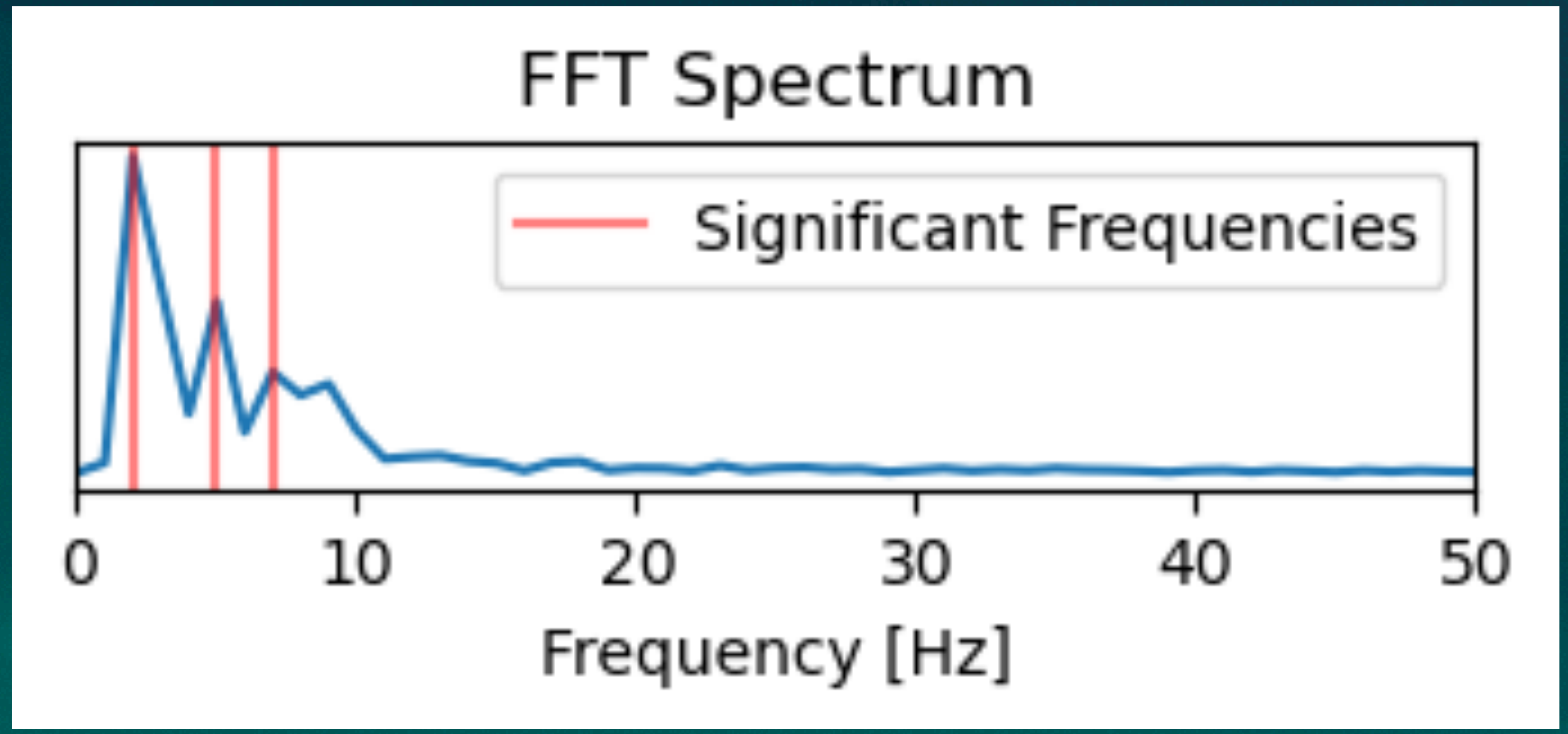
Idea: Extend features with clustering labels applied to processed segments of data.

What segments have similar characteristics or equivalently;

"Where are the patterns in the data?"



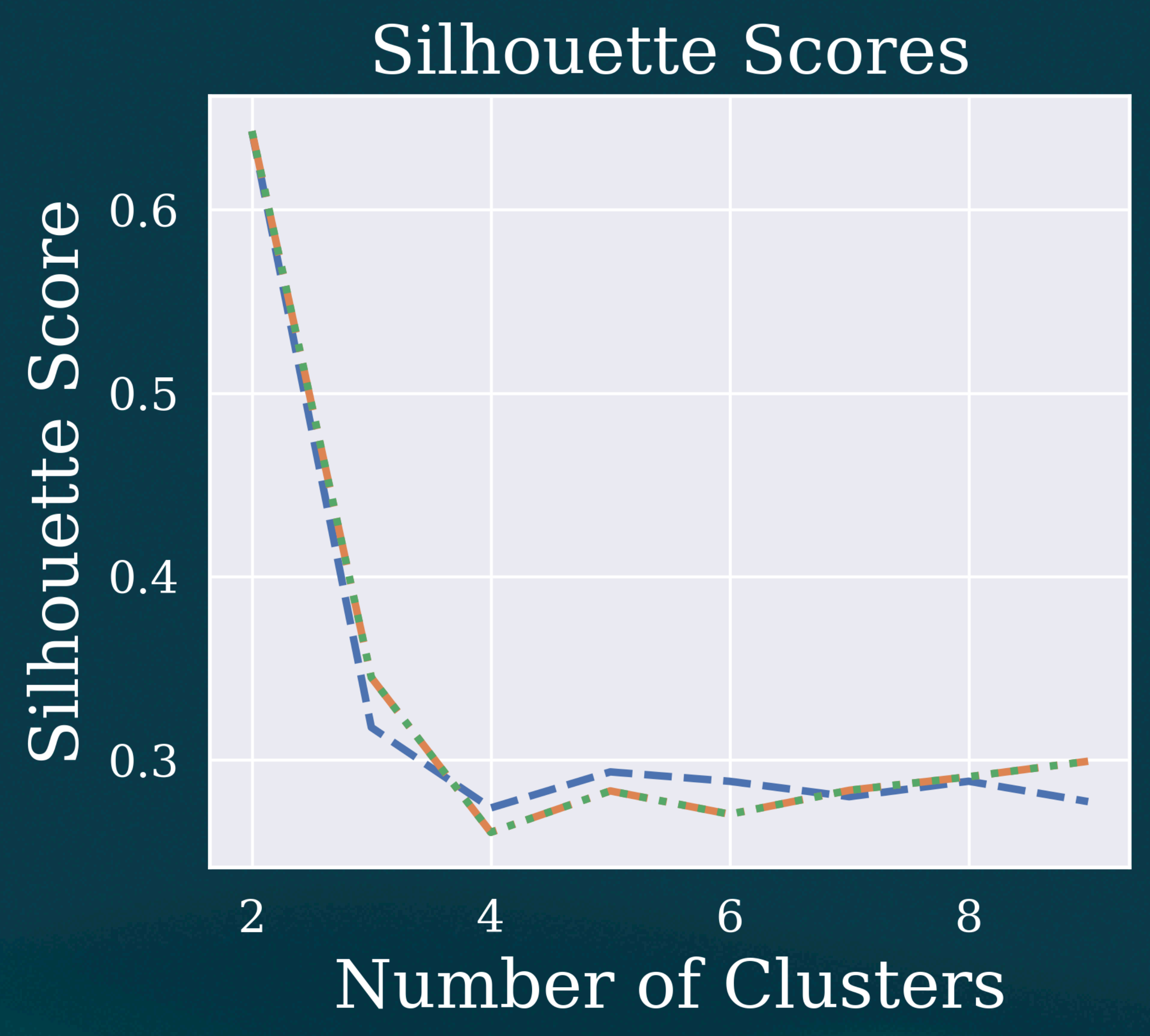


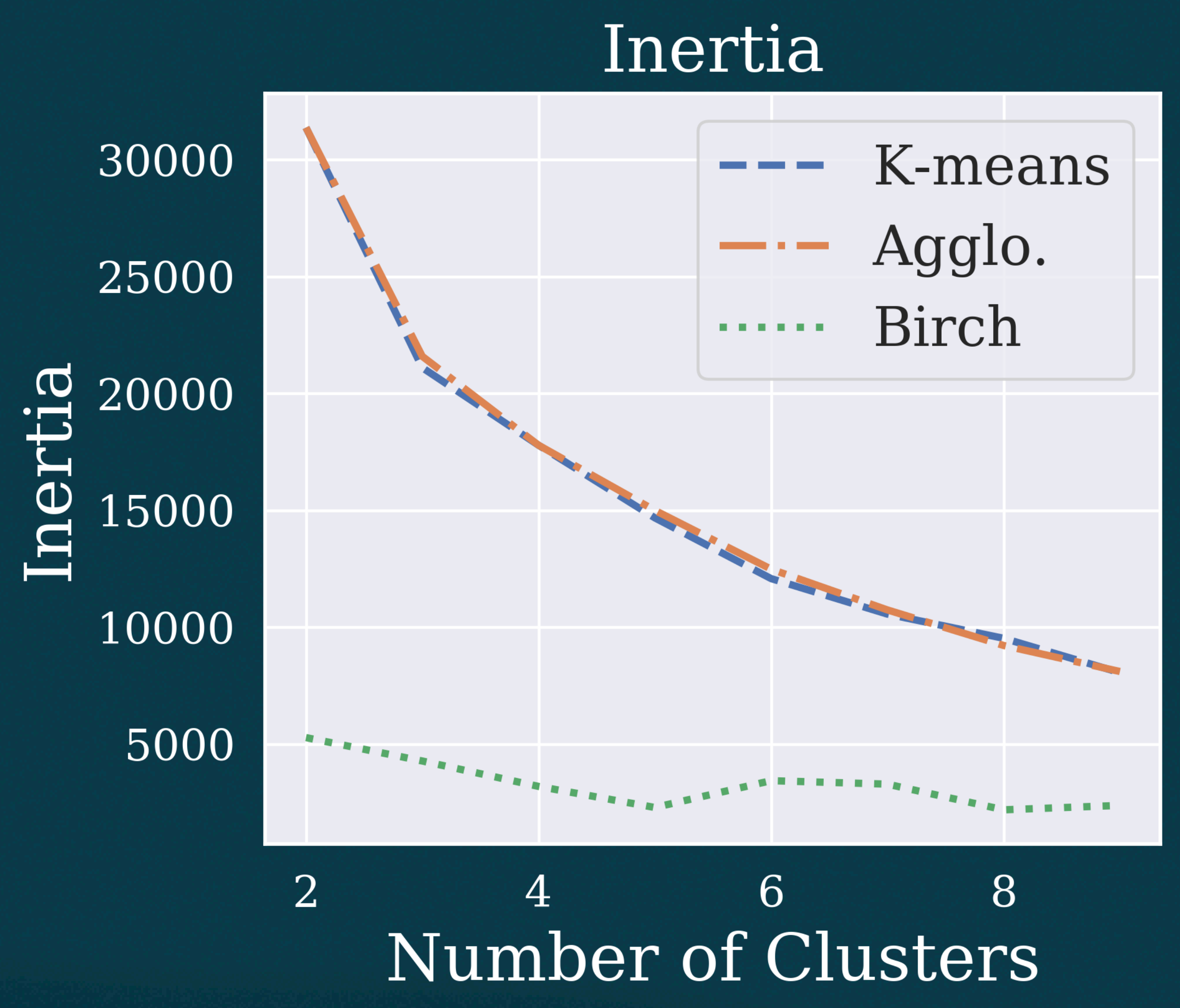


Feature Combinations

Clustering

- Number of clusters determined with silhouette score and set to 9
- Significant peaks set to 3





Input Features	Characteristics	Clustering methods
• Accelerations • Orientations • Angular rates	• Average • Standard Deviation • Significant Frequencies	• Kmeans labels • Agglomerative labels • Birch labels • Cross run clustering

Extending each feature component with factor 11

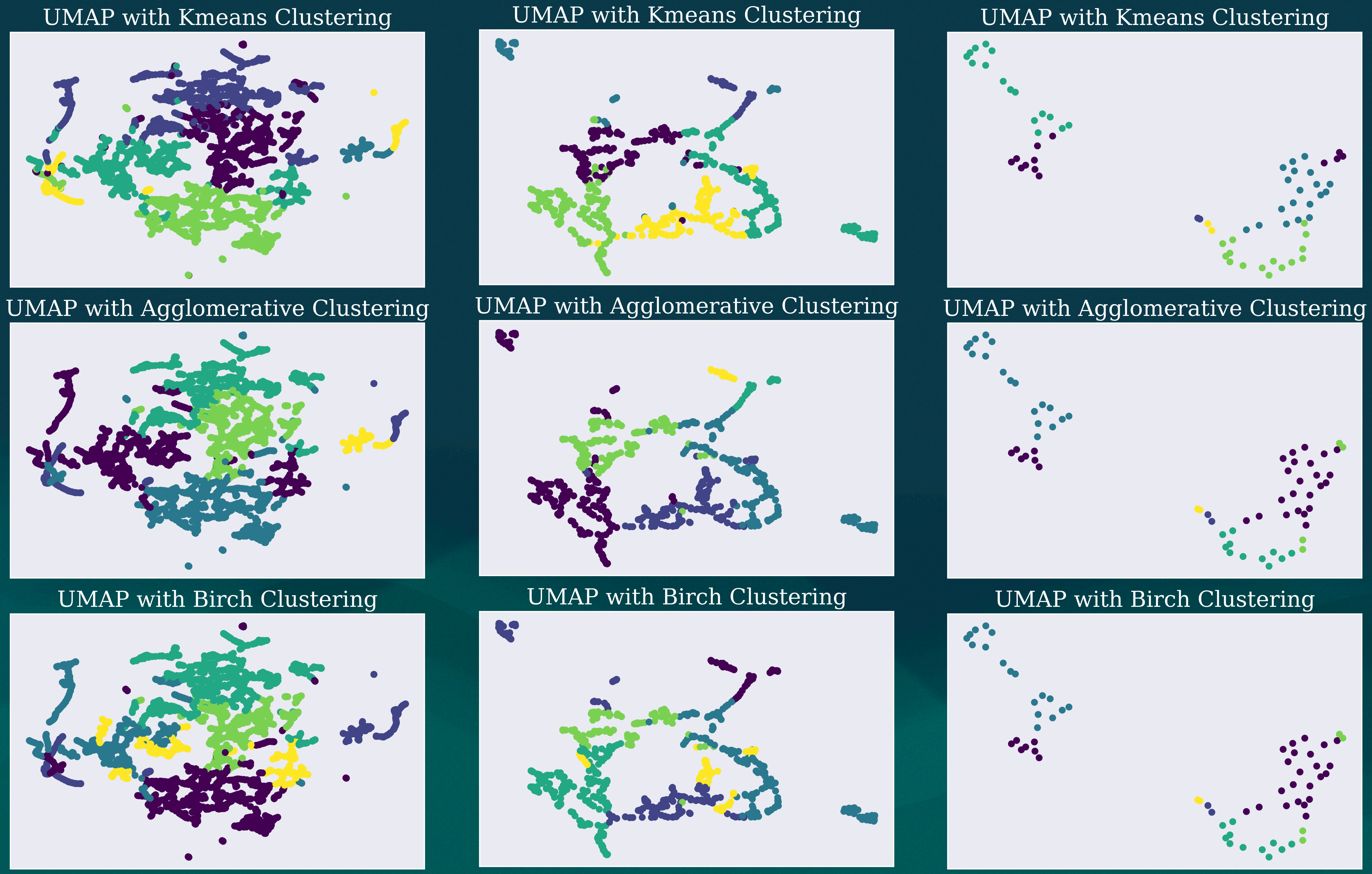
Feature Combination Clustering

Segmentation lengths reveal different behaviour

Tiny 0.1s: To capture micromovements like single-steps

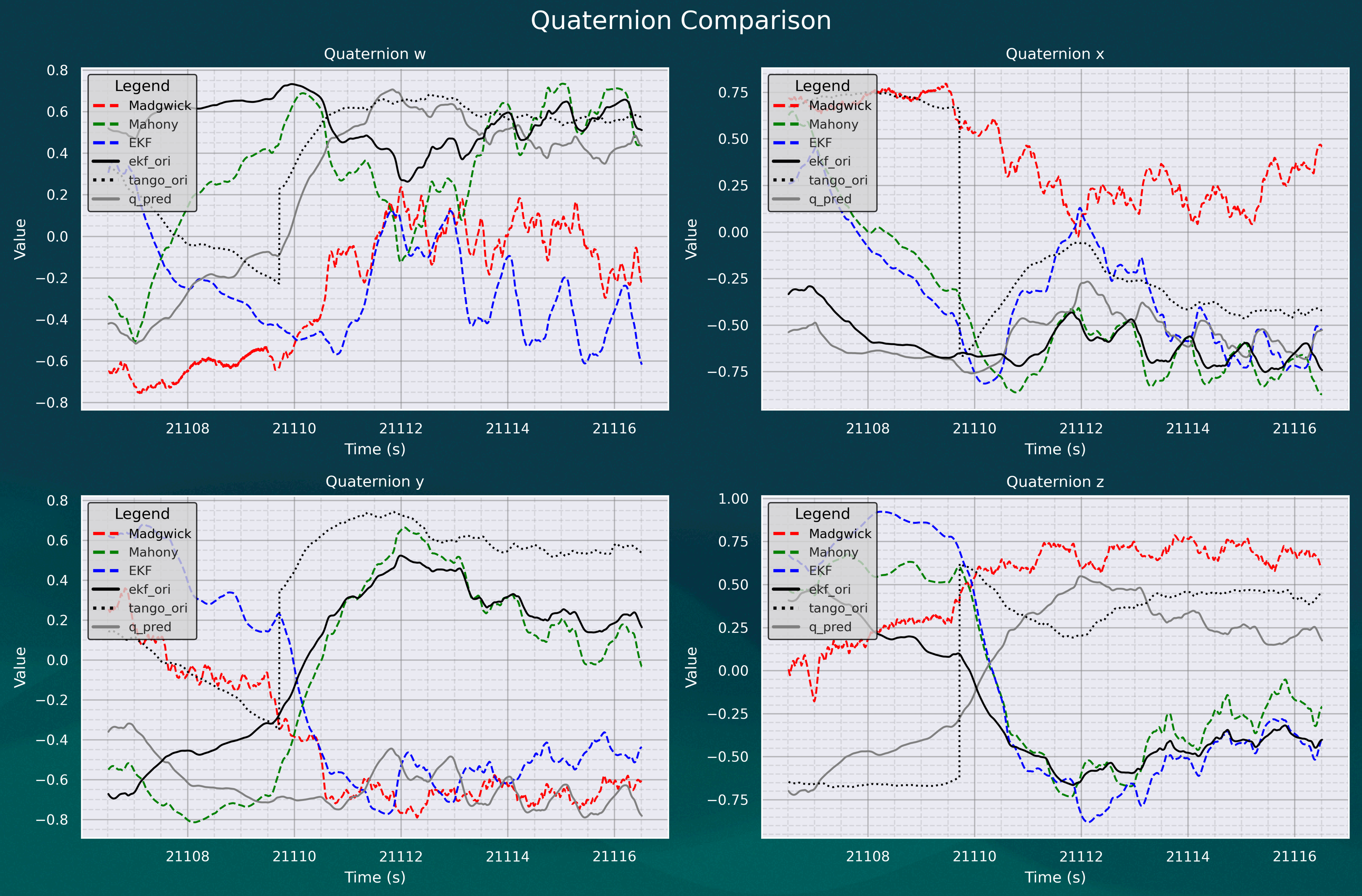
Short 0.5s: To capture movements like steps sequences and turns

Long 5s: To capture macromovement like running or talking on the phone



Filtering

- Problems with integration
- A multitude of analytical filters we applied to the gyroscopic data
- None yielded a clean fit with the ground truth



Loss Function for Quaternions

- Quaternions have symmetric properties, i.e. we have to change our loss function
- Divergence and Performance should be considered

$$\mathbf{q} = -\mathbf{q}$$

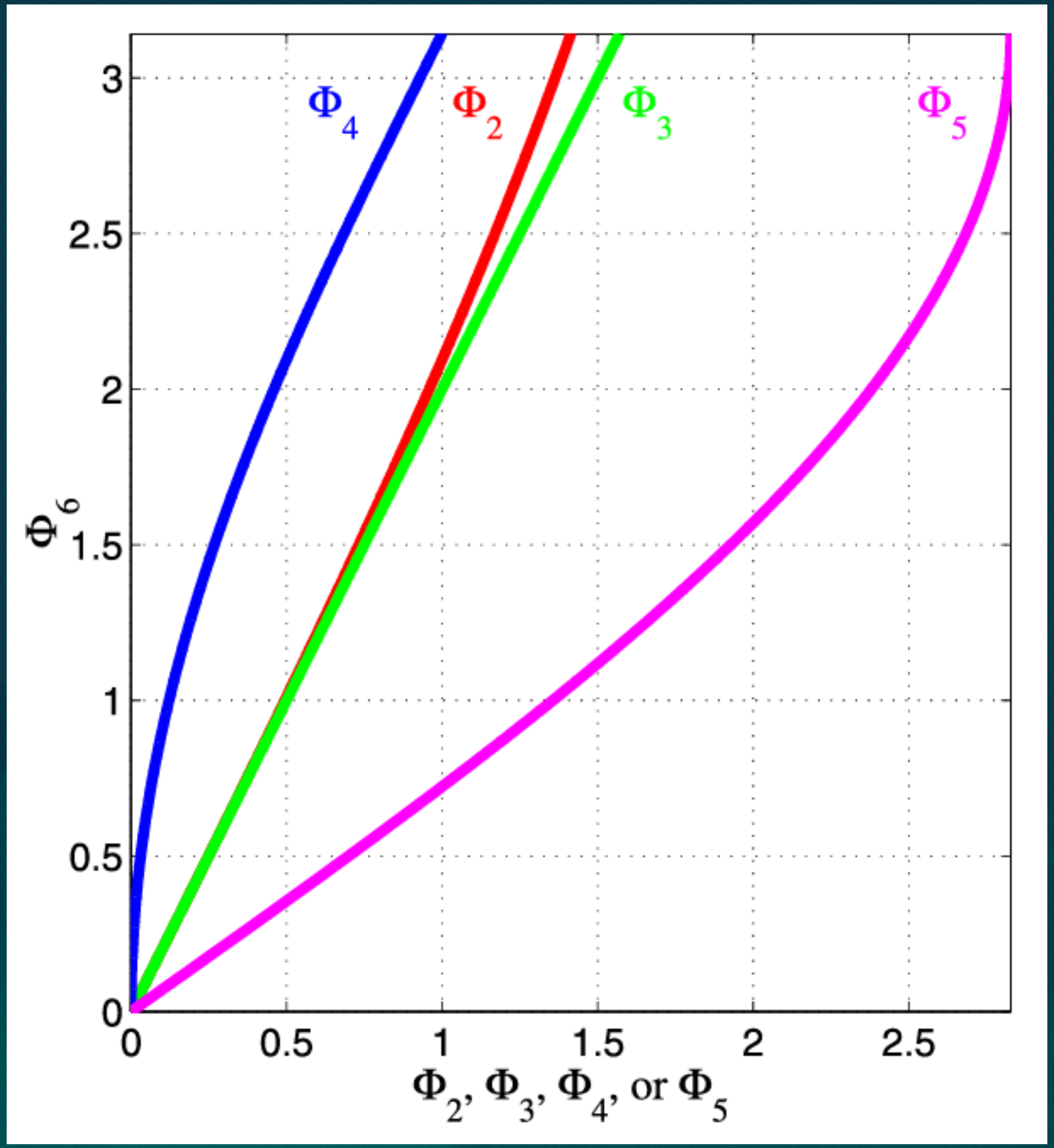
$$\phi_2(\mathbf{q}_1, \mathbf{q}_2) = \min(||\mathbf{q}_1 - \mathbf{q}_2||, ||\mathbf{q}_1 + \mathbf{q}_2||)$$

$$\phi_3(\mathbf{q}_1, \mathbf{q}_2) = \arccos(|\mathbf{q}_1 \cdot \mathbf{q}_2|)$$

$$\phi_4(\mathbf{q}_1, \mathbf{q}_2) = 1 - |\mathbf{q}_1 \cdot \mathbf{q}_2|$$

$$\phi_5(\mathbf{R}_1, \mathbf{R}_2) = ||\mathbf{I} - \mathbf{R}_1\mathbf{R}_2^T)||_F$$

$$\phi_6(\mathbf{R}_1, \mathbf{R}_2) = ||\log(\mathbf{R}_1\mathbf{R}_2^T)||$$

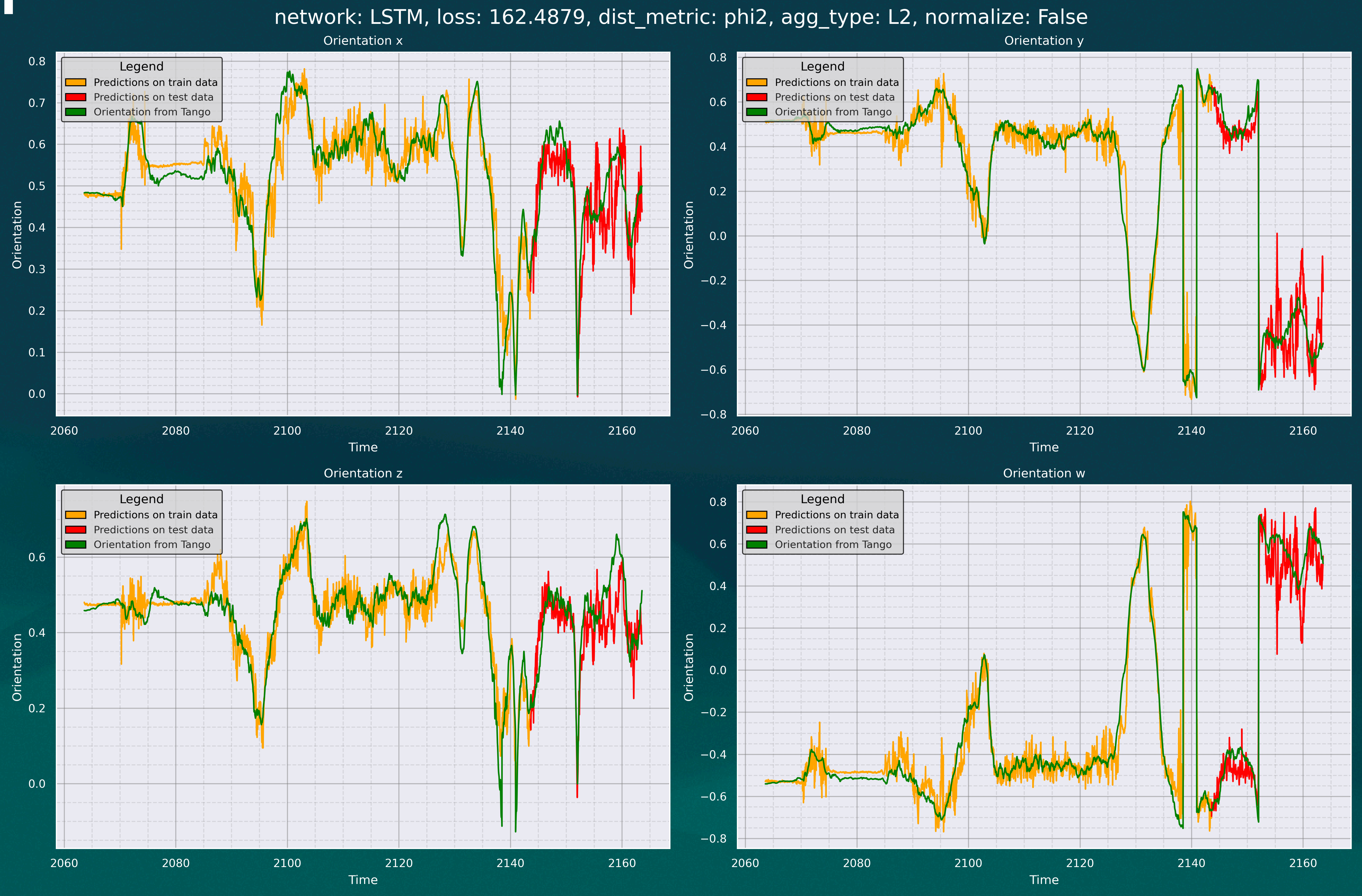


Convergence of different loss metrics
(source: 10.1007/s10851-009-0161-2)

Dynamic Filter Combination (LSTM)

Quaternion estimation

- Poor performance of individual filters led us to implement dynamic filter-weighting
- Filters are combined with an LSTM model
- Hyperparameter tuning is essential for a well-performing LSTM



Dynamic Filter Combination (comp.)

Quaternion estimation

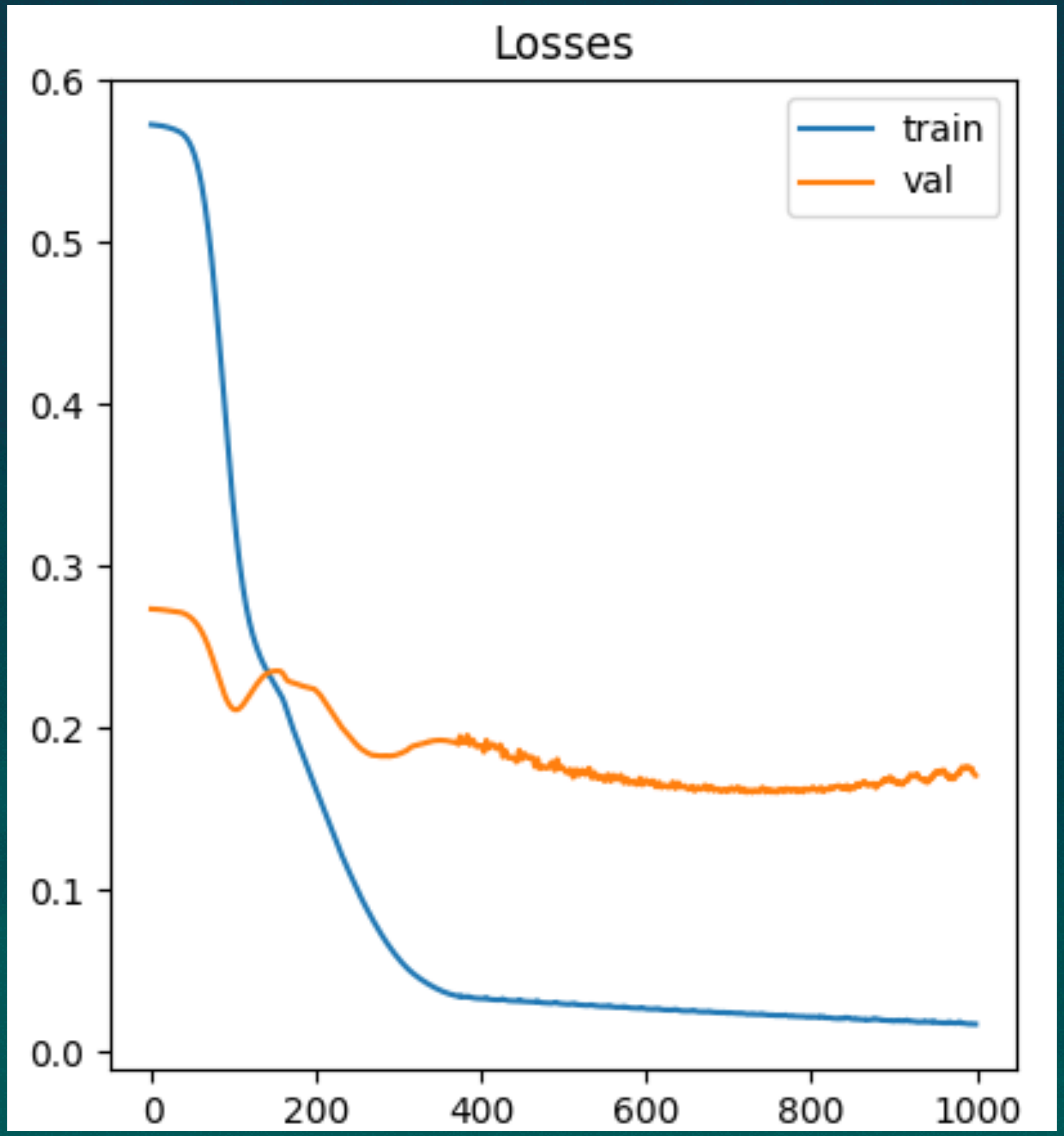
	NN-based models			Analytical approaches			
Model	LSTM	GRU	RNN	Integration	EKF	Mahoney	Madwick
Validation Loss	190 ± 40	250 ± 10	310 ± 40	9174	7772	7161	7904
Runtime (seconds)	150 ± 50	370 ± 90	74 ± 8	5.2	7.0	3.1	8.5

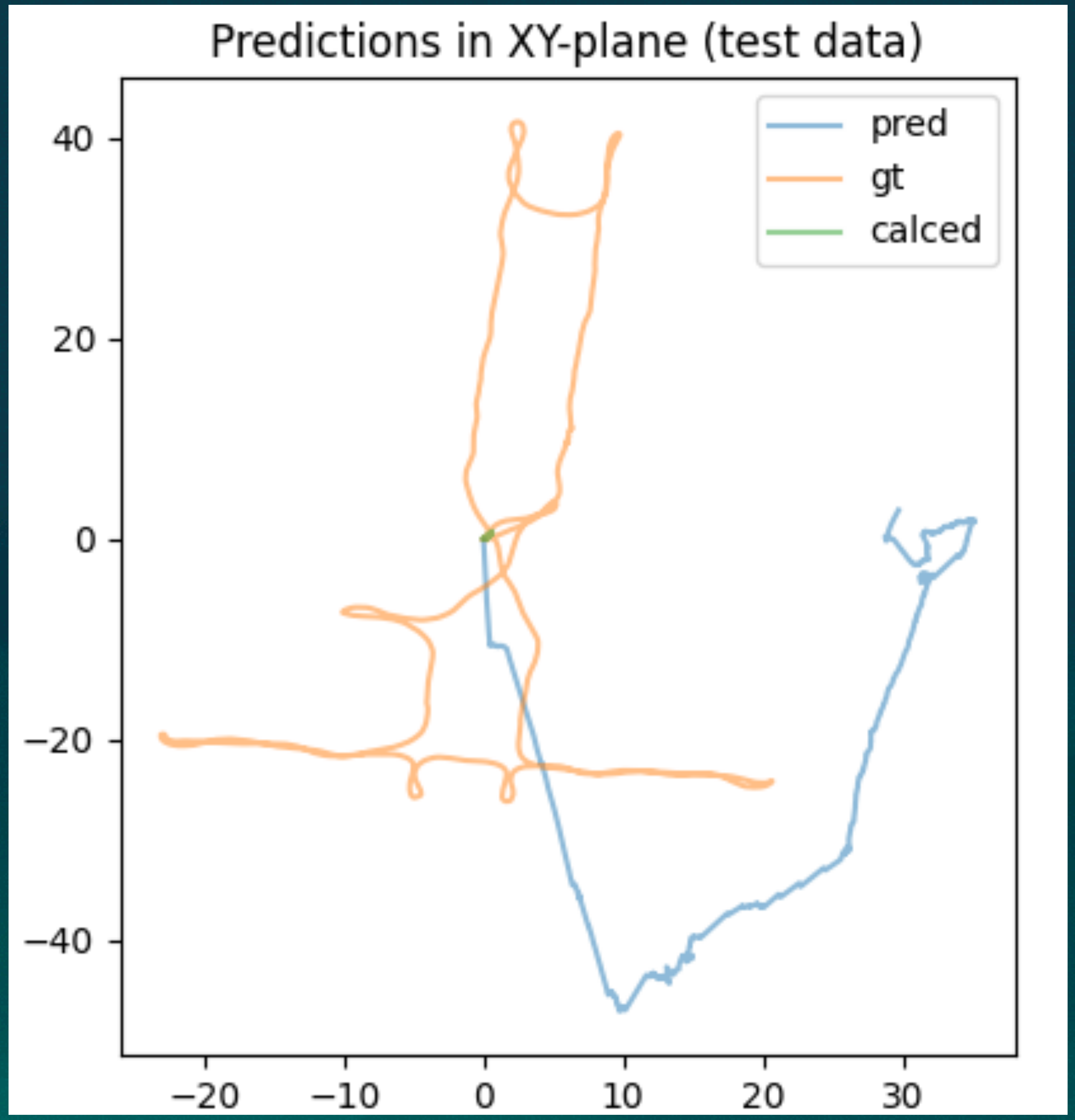
- Poor performance of individual filters led us to implement dynamic filter-weighting.
- Filters are combined with an LSTM, GRU and RNN model

Position estimation using just IMU data

EKF for quaternion estimation with a Linear network used for position estimation

Despite loss-convergence trace error remains great.

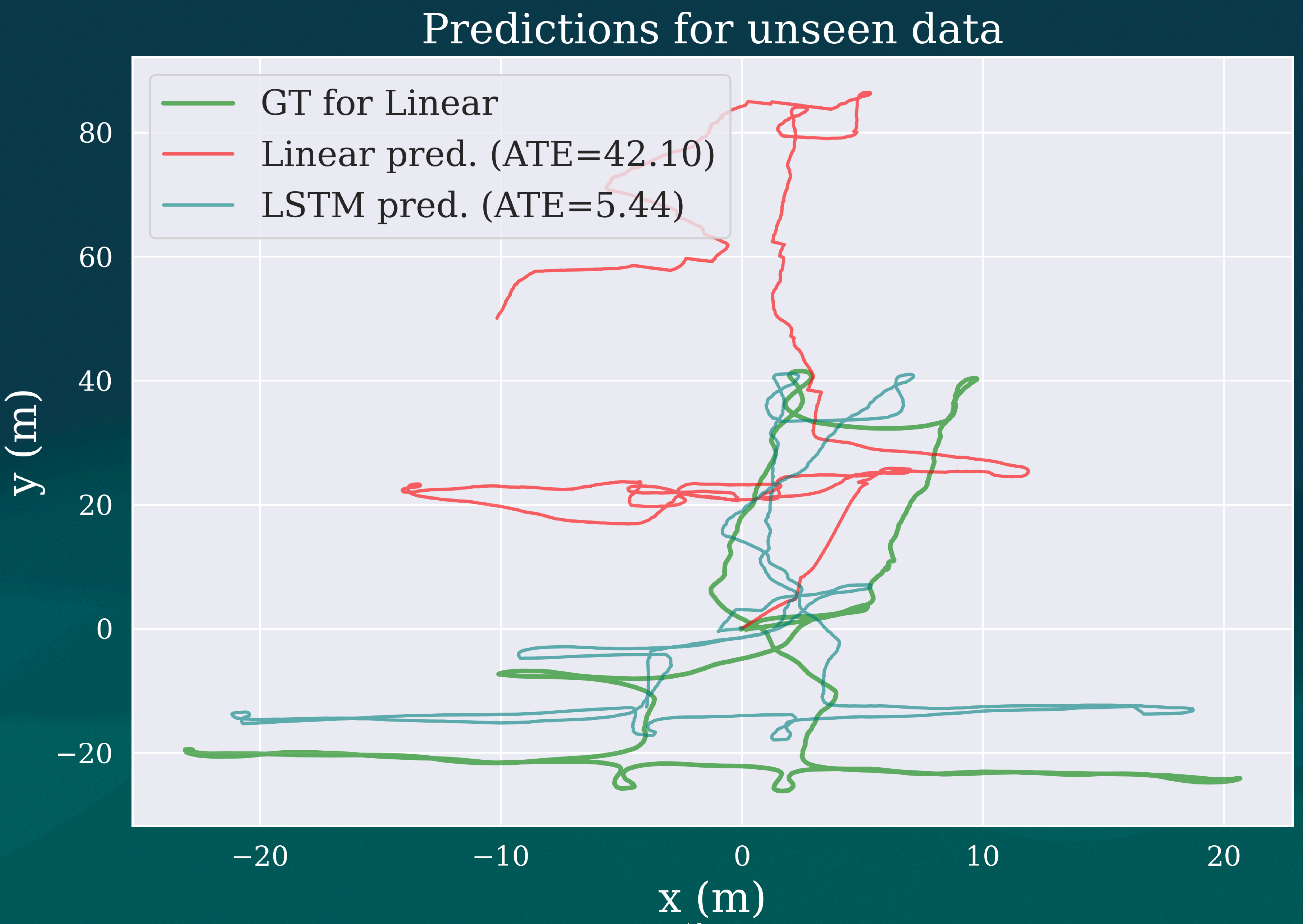




Position estimation using IMU + orientations

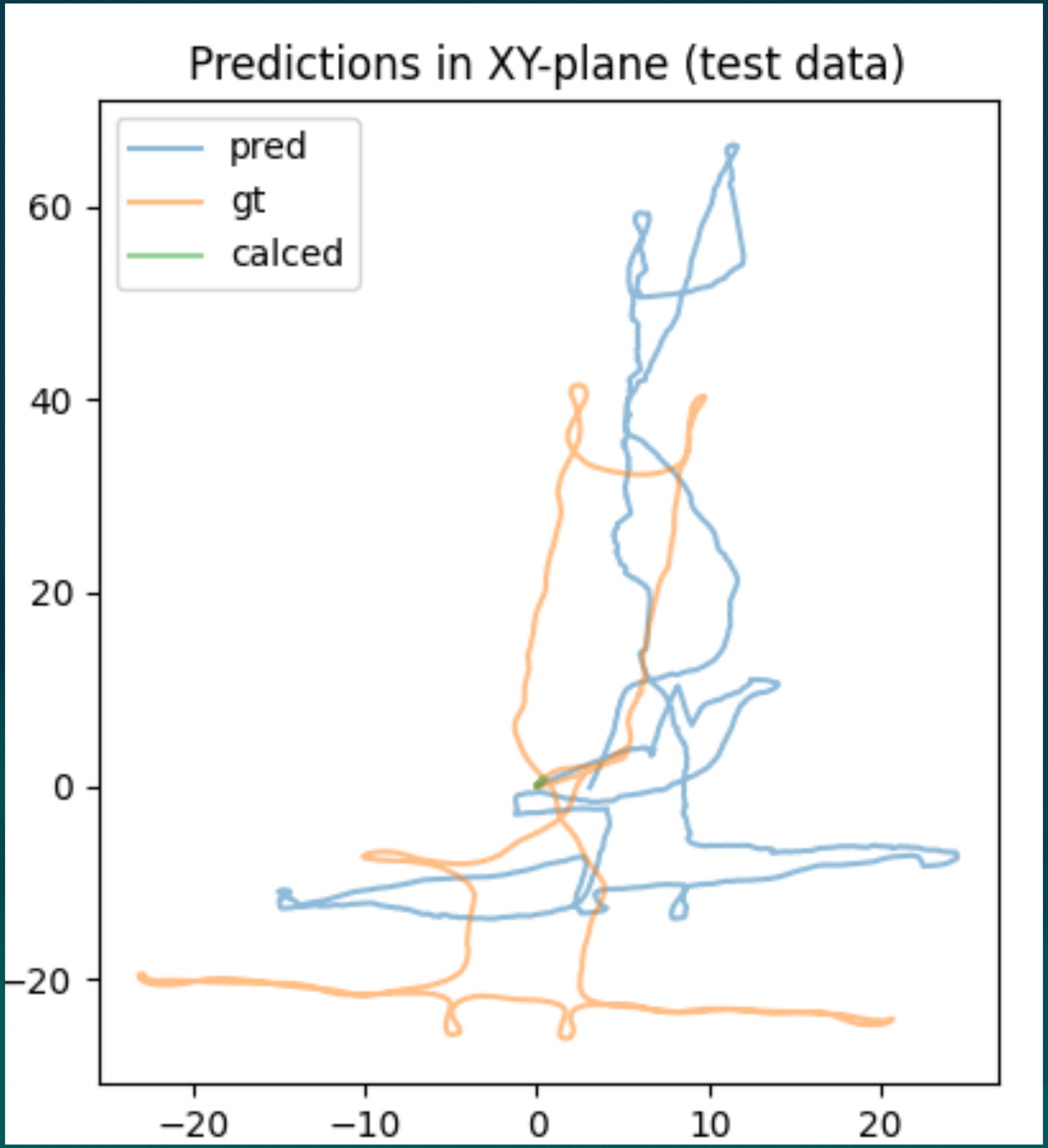
- Position prediction in 'easy mode'
- Architectures: Linear, RNN, GRU, LSTM
- 4.5 million training data points from 75 training sets
- Sequencing data
- Two approaches to predicting

Linear network vs LSTM

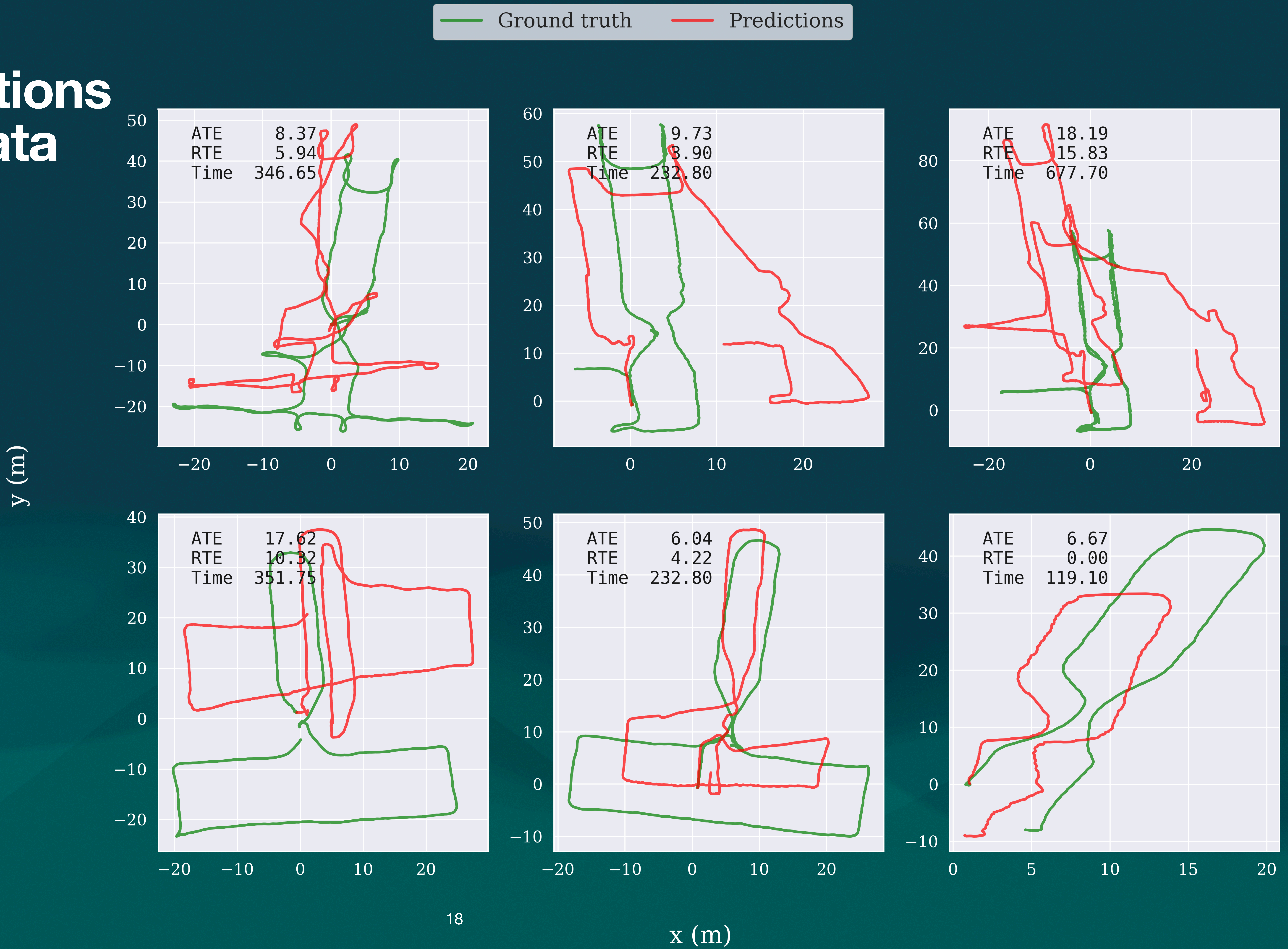


Best run of the linear network

Every once in a blue moon; Anton's linear network did well.



LSTM predictions on various data sets



LSTM, GRU and RNN performance

	Our models			Benchmark models			
Model	LSTM	GRU	RNN	NDR	RIDI	IONET	RoNIN LSTM
ATE (m)	32 ± 17	31 ± 15	30 ± 16	458.06	15.66	32.03	5.32
RTE (m)	26 ± 14	25 ± 13	24 ± 13	117.06	18.91	26.93	3.58
Runtime	3m 20s (NVIDIA 940mx 2GB)	5m 10s (NVIDIA 940mx 2GB)	4m 48s (NVIDIA 940mx 2GB)	?	?	?	12 h (NVIDIA 1080Ti 12GB)

- The results of each of our models are calculated as the average performance on 25 recordings
- It must be emphasized that our models are trained using IMU data + true orientations, whereas the benchmark model are trained just on IMU data
- Information on the benchmark models can be found at https://arxiv.org/abs/1905.12853

Difficulties

- The complexity of the problem necessitated long training times – thus hindering fast exploration.
- Models must be trained on much data (>600,000) in order to learn data structure even roughly
- Other shit

Learnings

- Working with 'Big Data'
- Working with sequential models
- Working with noisy data
- Time series analysis

Appendix

- Analytic quaternion update
- Clustering details
- Effect of supplementing with clustering data
- Filtering method description
- Model descriptions

Link to our Git Repository

Analytical Gyroscope Integration

```
def Omega(w=np.array([1,1,2]).reshape(3,1)):
    w = is_column_vector(w)
    w_cross = np.array([[0, -w[2,0], w[1,0]],
                        [w[2,0], 0, -w[0,0]],
                        [-w[1,0], w[0,0], 0]])
    top = np.hstack((-w_cross, w))
    bottom = np.hstack((-w.T, np.zeros((1,1))))
    return np.vstack((top, bottom))

def u_b(w=np.array([1,1,2]).reshape(3,1), dt=.1):
    return  w*dt

# matrix norm of a 4x4 matrix
def matrix_norm(M):
    return np.sqrt(np.trace(M.T@M))

def vec_norm(v):
    return np.sqrt(v.T@v)

def Theta(w=np.array([1,1,2]).reshape(3,1), dt=.1):
    W = Omega(w)
    u = u_b(w, dt)
    W_norm = matrix_norm(W)
    u_b_norm = vec_norm(u)
    return np.cos(u_b_norm/2)*np.eye(4) +
np.sin(u_b_norm/2)/(u_b_norm/2)*W
```

Given the current orientation, the next orientation is obtained via application of the quaternion update matrix;

$$q_{t+1} = \Phi q_t$$

With theta approximated as;

$$\Theta = \cos\left(\frac{|u|}{2}\right) \cdot I + \sin\left(\frac{|u|}{2}\right) \cdot \frac{2\Phi}{|u|}$$

In which;

$$u = \int \omega dt \quad \text{and} \quad \Omega = \begin{bmatrix} -\omega_\times & \omega^T \\ \omega & 0 \end{bmatrix}$$

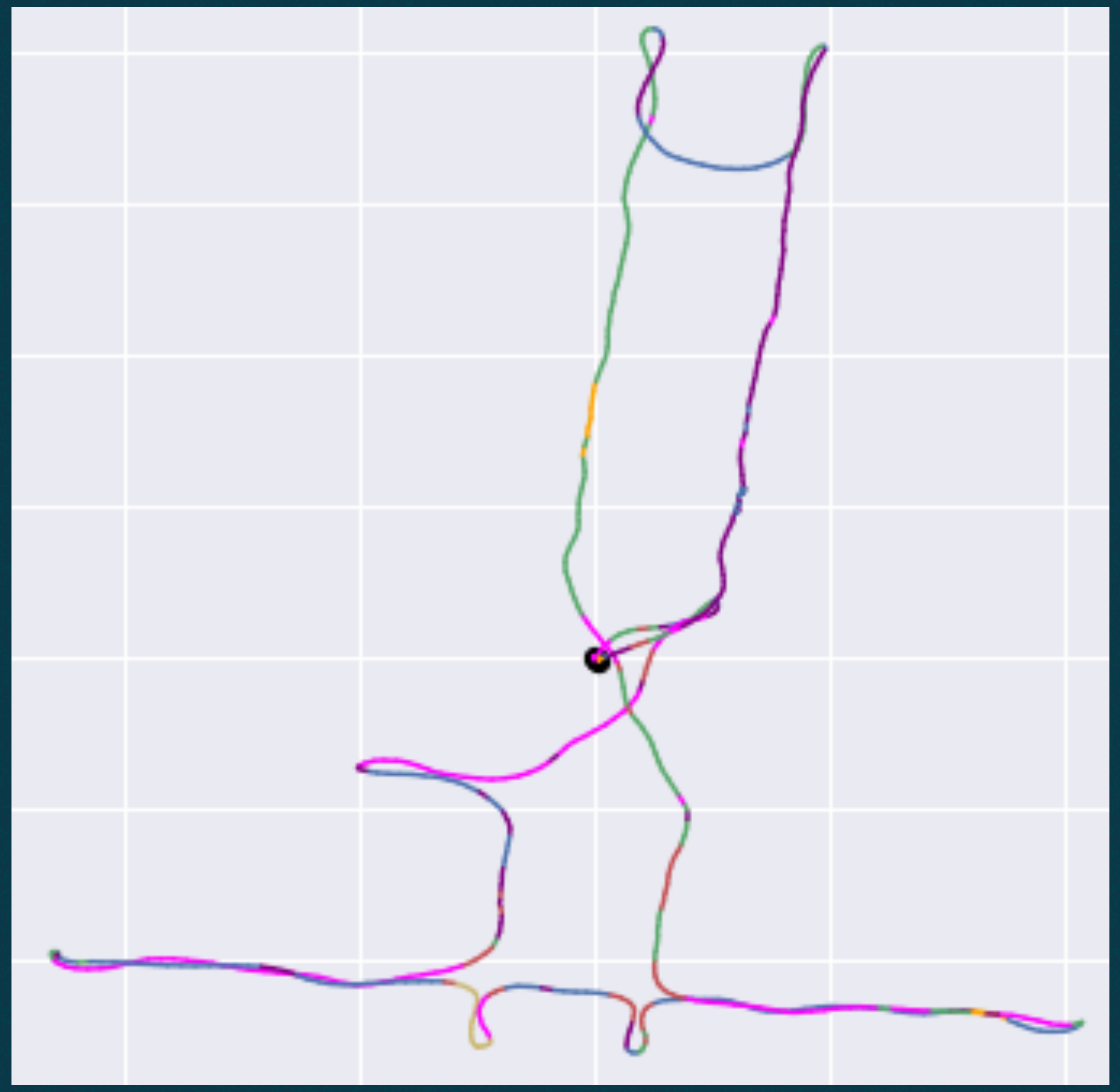
with

$$\omega_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

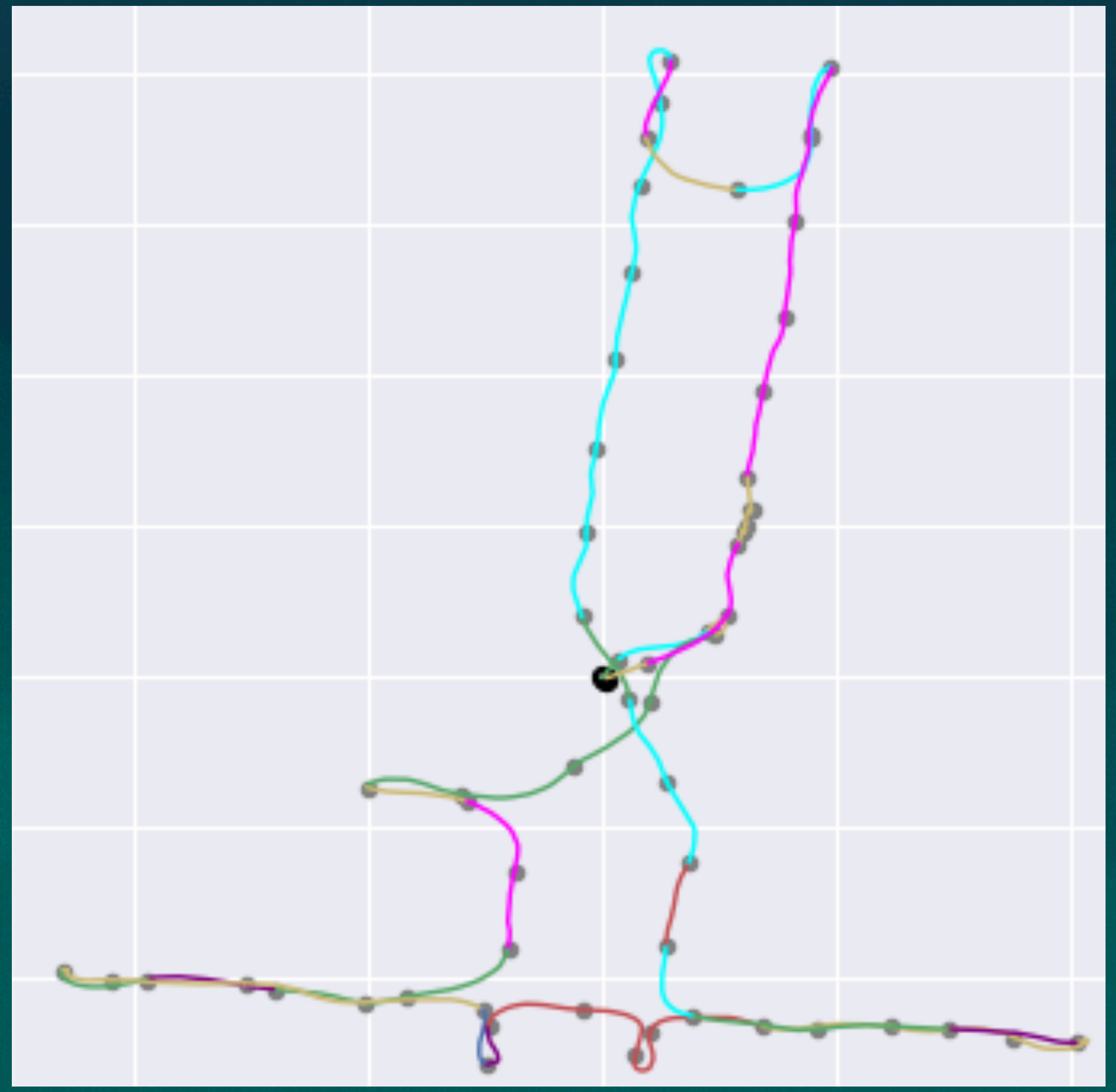
Clustering details

- Feature extension by clustering allows a guiding hand for the NN.
- Will not (and did not) provide additional accuracy for deep NN, as the structures being parsed are already learned by the deep NN.
- Possible to extend with other characterics, like the laplace transform, taylor expansion etc.
- Gave insight into the methodology of true behavior classification from IMU data.
  Which is coming to the phone near you!

Micromovement Clusters



Macromovement Clusters



Kalman filtering

- Use prior knowledge of state to inform measurement dependent state update.
- With user inputs of signal and processing noise, the algorithm takes previous states.
- Noise estimates are updated iteratively
- For documentation regarding the remaining filters see: https://ahrs.readthedocs.io/en/latest/metrics.html

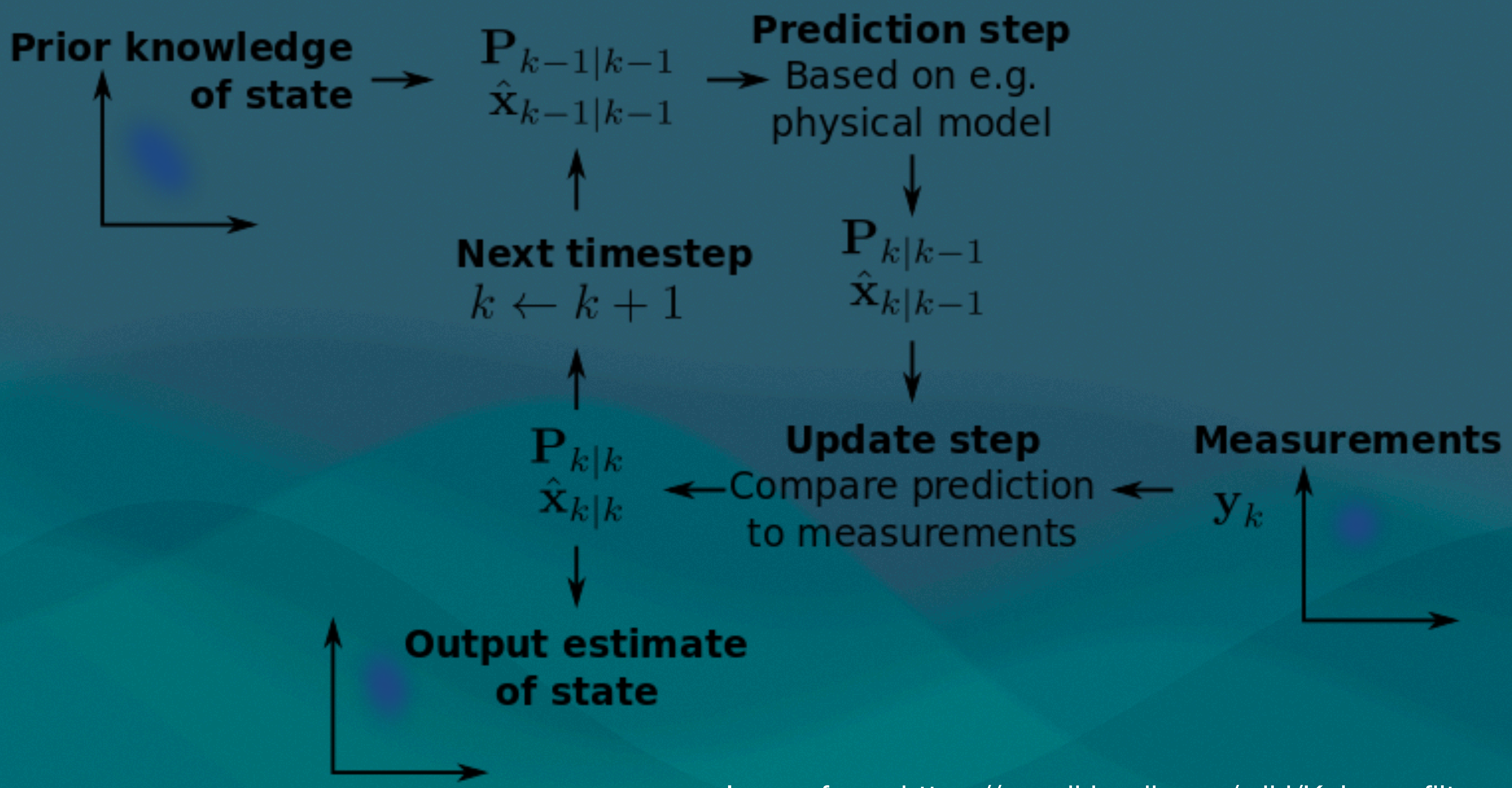


Image from: https://en.wikipedia.org/wiki/Kalman_filter

Does clustering improve performance?

- The 8 different cluster labels were encoded using one hot encoding and used as input features in addition to the IMU data and true orientations for the LSTM

- Each model is trained 3 times on 1.2M data points

- As can be seen below, the inclusion of clustered features did not significantly improve performance, and so we decided not to include them

Av. Performance on test dataset a000_1 (3 trials)	LSTM	
Error	With clustering	Without clustering
ATE (m)	12.4 ± 2.7	8.3 ± 2.6
RTE (m)	7.3 ± 3.9	6.6 ± 0.9

A couple of the things we tried:
Quaternion estimation

- A lot of time was spent on trying to get linear networks, as well as more complicated sequential ones, to predict orientations using just IMU data, as as well a single initial orientation. Most of our models just output constant predictions and were worthless. In our ignorance, we used small sequence lengths and trained on 'toy data' snippets of the full data set. Short sequences meant that the network was almost right when consistently predicting null changes in orientation. Our lack of success prompted us to dive into the world of data filters and custom loss functions.

- We initially tried using previous true orientations as input features during training. These true orientations would then be replaced by the model predictions during training. The models ended up ignoring everything but the orientation features, and failure ensued.

- We tried using any and all combination of filtered features, along with intermediary analytical update steps, ultimately resulting in the results summarized in the 'Dynamic Filter Combination' slide

A couple of the things we tried:

Position estimation using just IMU data

- Much effort was put into building a transformer that could predict positions given solely IMU data, the hope being that this superior architecture would be able to outperform the RoNIN's LSTM model. Transformer based network —> turned out to require more than available compute. The model was unable to learn any relevant structure.

- We tried using various combinations of filtered features, along with intermediary analytical update steps

- We tried giving the networks the initial orientation and tasked them to predict absolute positions -> Fail. We ended up predicting relative positions, both trying to predict 1 to several positional changes per sequence.

A couple of the things we tried:

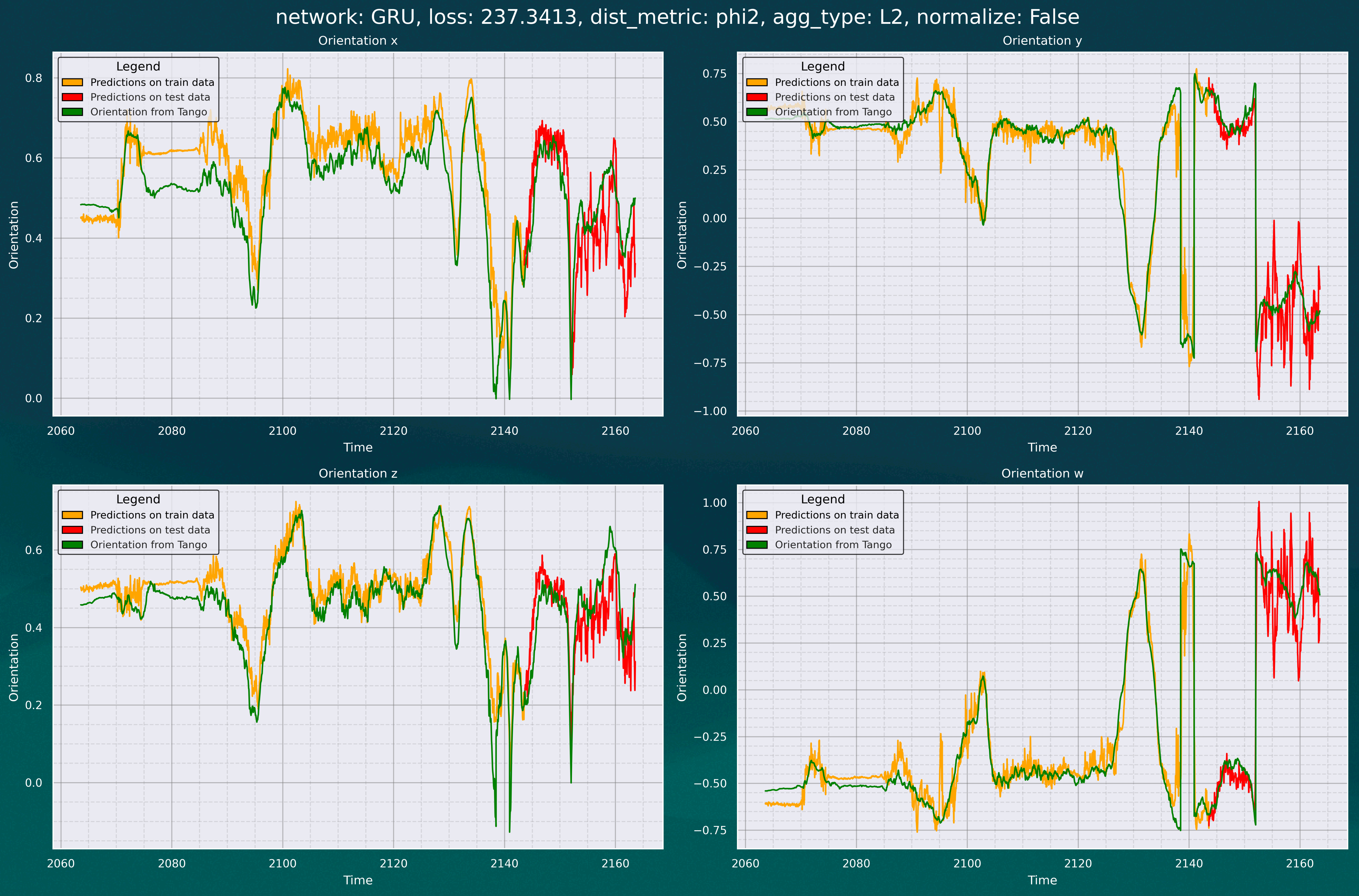
Position estimation using IMU data + true orientations

- We tried using various combinations of filtered and clustered features, along with intermediary analytical update steps
- We tried giving the networks the initial orientation and tasked them to predict absolute positions -> Fail.
- We ended up predicting relative positions, both trying to predict 1 to several positional changes per sequence. Both approaches lead to similar results, but the former approach was much faster to train. It is possible that the latter has more potential if given the proper amount of training time

Dynamic Filter Combination (GRU)

Quaternion estimation

- Poor performance of individual filters led us to implement dynamic filter-weighting
- Filters are combined with an GRU model
- Hyperparameter tuning is essential for a well-performing GRU model



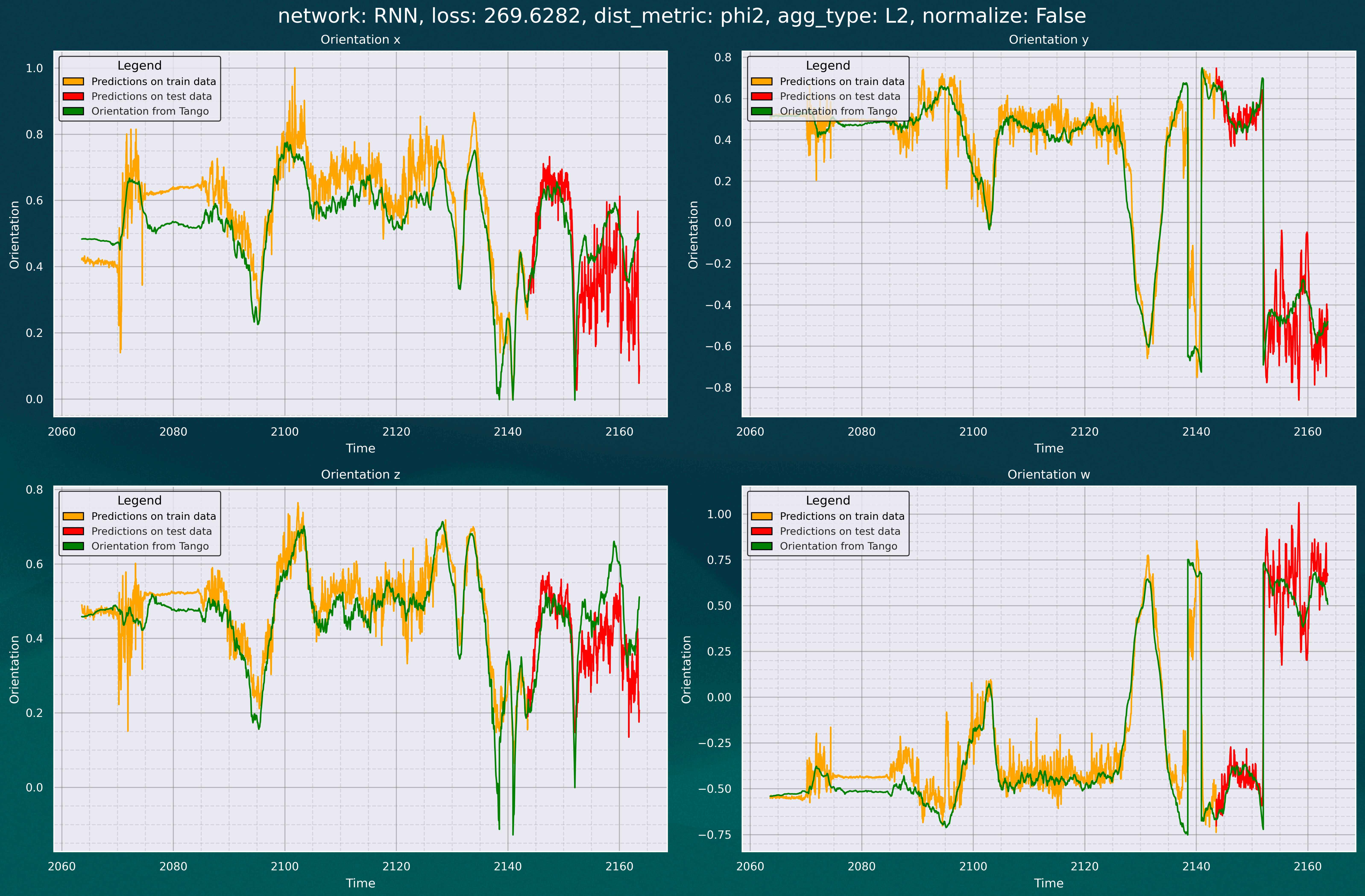
Dynamic Filter Combination (RNN)

Quaternion estimation

- Poor performance of individual filters led us to implement dynamic filter-weighting

- Filters are combined with an RNN model

- Hyperparameter tuning is essential for a well-performing RNN model



Linear network for position estimation (Pytorch)

Model summary

IMU data + true orientations

↓

Rotate acceleration
to world-frame

↓

perform double integration

↓

9 layers
690 —> 500(8) —> 3

Dense linear network

↓

Relative XYZ positions

Name	Description
Architecture	Densely connected linear
Parameter count	2.2M
Loss function	MSE
Learning rate	10^{-6}
Sample count (train/val)	1M (80/20)
Runtime	23m 40s
Epochs	500
Sequence length	300
Sequence overlap	1
Batch size	128

LSTM for position estimation (Pytorch)

Model summary

IMU data + true orientations

3 layers (n=62) — **LSTM**

2 layers (62 -> 290 -> 2) — **Dense linear network**

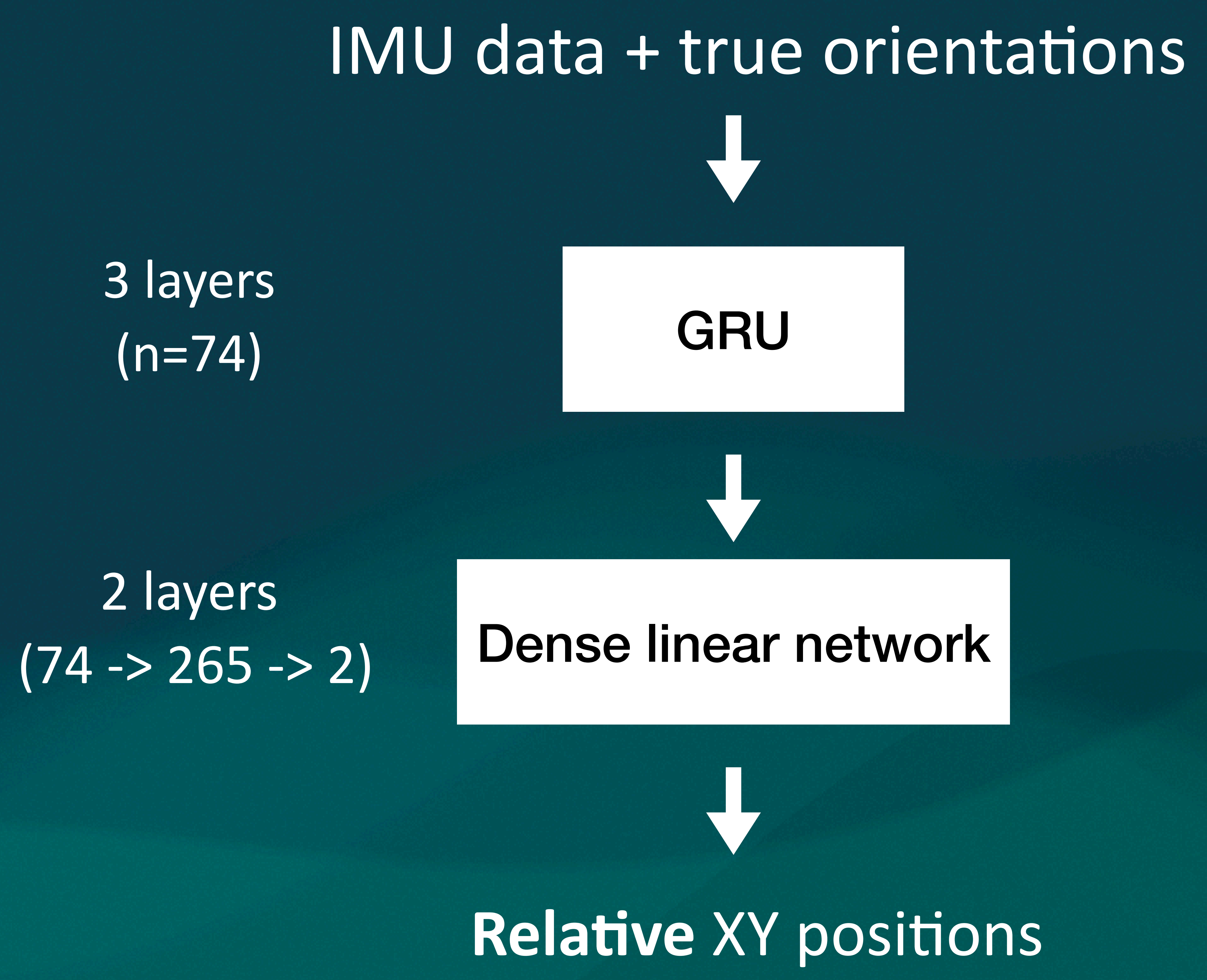
Relative XY positions

Name	Description
Architecture	LSTM
Parameter count	82000
Loss function	MSE
Learning rate	3.9 * 10^{-5}
Sample count (train/val)	1.2M (80/20)
Runtime	3m 20s
Epochs	78
Sequence length	30
Sequence overlap	1
Batch size	64
Regularization	dropout=0.2

Hyperparameter optimzation: Epochs, sequence length, learning rate and N_hidden were optimized using Optuna for 75 trials (runtime = 5 hours on NVIDIA GeForce 940MX)

GRU for position estimation (Pytorch)

Model summary



Name	Description
Architecture	GRU
Parameter count	101744
Loss function	MSE
Learning rate	1.7 * 10^{-4}
Sample count (train/val)	1.2M (80/20)
Runtime	5m 10s
Epochs	101
Sequence length	35
Sequence overlap	1
Batch size	64
Regularization	dropout = 0.2

Hyperparameter optimzation: Epochs, sequence length, learning rate and N_hidden were optimized using Optuna for 75 trials (runtime = 8 hours on NVIDIA GeForce 940MX)

RNN for position estimation (Pytorch)

Model summary

IMU data + true orientations

↓

3 layers (n=58) — **RNN**

↓

2 layers (58 -> 290 -> 2) — **Dense linear network**

↓

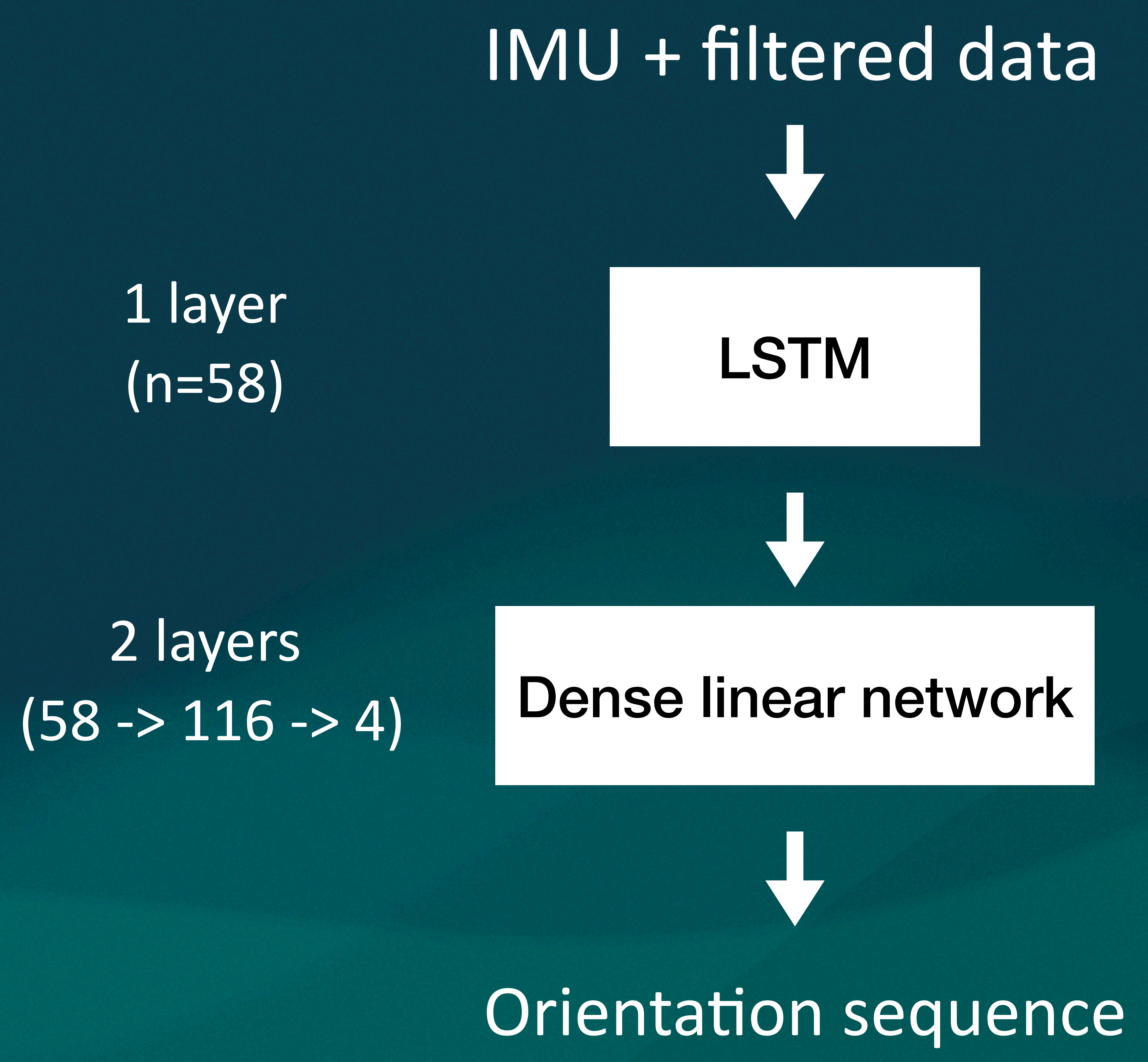
Relative XY positions

Name	Description
Architecture	RNN
Parameter count	35613
Loss function	MSE
Learning rate	3.6 * 10^{-4}
Sample count (train/val)	1.2M (80/20)
Runtime	4m 48s
Epochs	47
Sequence length	30
Sequence overlap	1
Batch size	64
Regularization	dropout = 0.2

Hyperparameter optimzation: Epochs, sequence length, learning rate and N_hidden were optimized using Optuna for 30 trials (runtime = 4 hours)

LSTM for orientation estimation (Pytorch)

Model summary

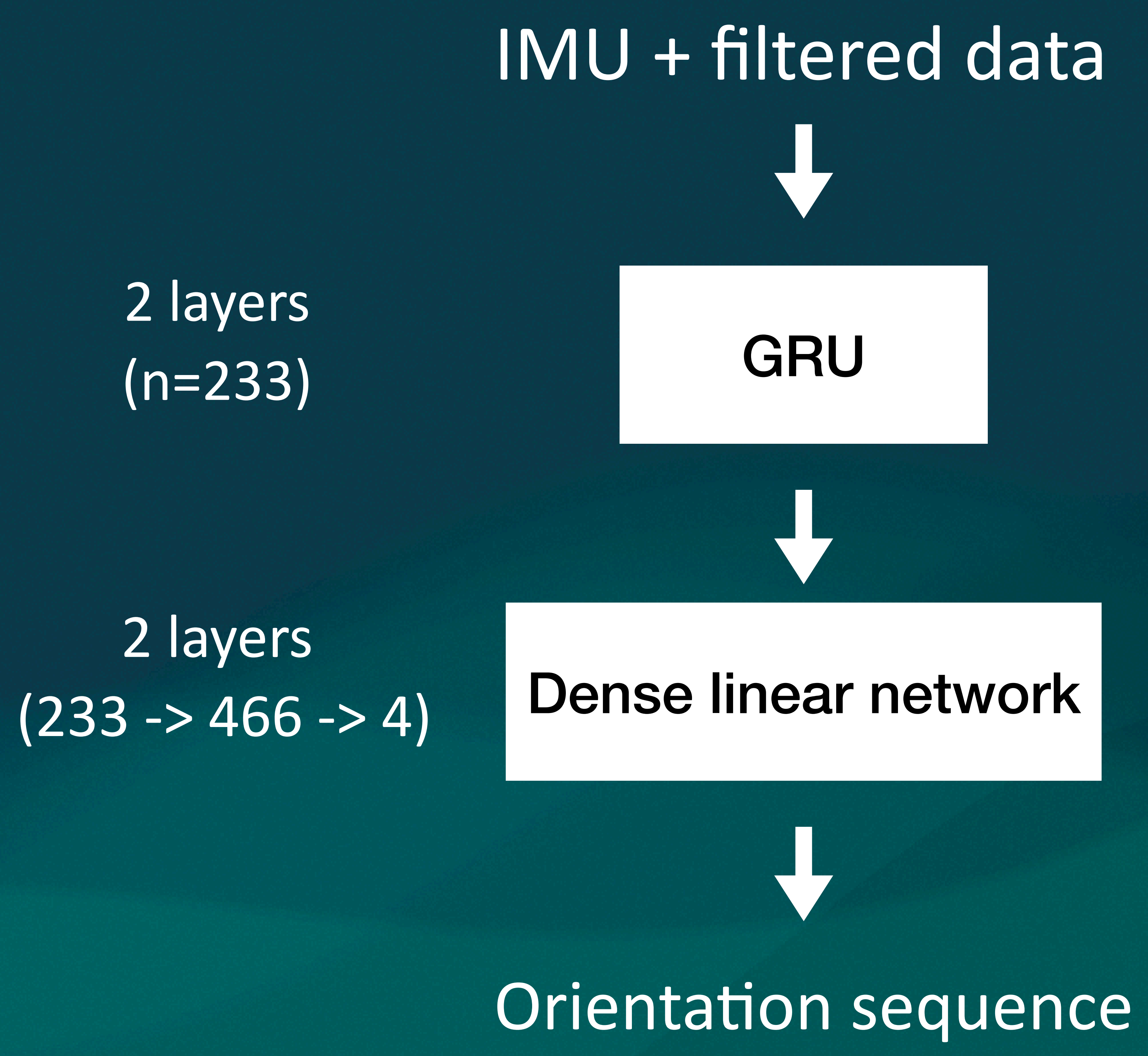


Name	Description
Architecture	LSTM
Bidirectional	TRUE
Parameter count	25348
Loss function	phi2
Optimizer	Adam
Learning rate	0.005
Weight Decay	0.02
Betas	(0.51, 0.97)
Training/Validation Size	16000|4000
Runtime	150 ± 50
Epochs	96

Hyperparameter optimization with 200 trials (optuna and optuna-dashboard): amsgrad, betas, hidden_size, learning_rate, num_layers, optimizer, weight_decay

GRU for orientation estimation (Pytorch)

Model summary



Name	Description
Architecture	GRU
Bidirectional	TRUE
Parameter count	1345346
Loss function	phi2
Optimizer	Adamax
Learning rate	0.0005
Weight Decay	0
Betas	(0.40, 0.46)
Training/Validation Size	16000|4000
Runtime	370 ± 90
Epochs	92

Hyperparameter optimization with 200 trials (optuna and optuna-dashboard): amsgrad, betas, hidden_size, learning_rate, num_layers, optimizer, weight_decay

RNN for orientation estimation (Pytorch)

Model summary

IMU + filtered data

↓

1 layer (n=179) — **RNN**

↓

2 layers (179 -> 358 -> 4) — **Dense linear network**

↓

Orientation sequence

Name	Description
Architecture	RNN
Bidirectional	TRUE
Parameter count	75184
Loss function	phi2
Optimizer	AdamW
Learning rate	0.001
Weight Decay	0.034
Betas	(0.74, 0.88)
Training/Validation Size	16000|4000
Runtime	74 ± 8
Epochs	64

Hyperparameter optimization with 200 trials (optuna and optuna-dashboard): amsgrad, betas, hidden_size, learning_rate, num_layers, optimizer, weight_decay

Example Plot for Early Stopping (RNN for quaternions)