## Motivation/Introduction

1 – Dynamical Systems

* Estimating the state of systems that evolve over time is a fundamental task in signal processing and control, but these states are often hidden from our view or may not be directly measured by us with sensors. Some applications include localization, tracking, and navigation.
* For a more tangible set of examples, consider that any sensor we may use in the real-world is going to be noisy and could have a thin or wide range of measurement values.
* So, then how do we estimate the true value of a measurement? One way is to take measurements across time and to estimate the true state of the sensor by comparing our predictions to those measurements.
* Extra: It could also be a system that a user is trying to model but the state of the system may not be directly measurable. Consider the measurement of a rocket thruster’s output. This cannot be done by putting a sensor inside the thruster; it must be placed in a cooler location.

2 – Measurements Across Time A

* Let’s suppose you measured yourself on a scale for 12 days, but you know that the scale is inaccurate within some range, say 5 pounds. You could then place a “best-fit” line on the measurements that give you a prediction that you’re gaining 1 lb / day.
* On the 13th day, you could take another scale measurement and – based on your prediction – decide that you expect to be at 159 lbs. However, the scale says 164 instead! What do we do with this?
  + If we take only our prediction, then we ignore useful information within the scale’s measurement.
  + If we take only the measurement, then we ignore all the past information and the trend or hypothesis we came up with.
* Instead, we take a mixture of both prediction and measurement where the actual estimate at hat(x)\_t is determined by a weighted combination of the two.
* When using this prior belief and taking noisy measurements for multiple days, we can see that much of the noisiness is taken care of.
  + Remember that predictions are +1 the previous day’s weight.

3 – Measurements Across Time B

* What if your prediction or belief is wrong?
* The filtering process eventually recovers but it takes multiple days, and it’s not clear when the estimations will finally converge to the actual measurements trend.
  + This is with a constant gain that focuses on going closer to the measurement value than the prediction.
* If the gain becomes adaptive, the process model believes the prediction less and trusts measurements more, so estimates converge to the true measurement line.

4 – G-h Filters and the Kalman Filter (KF)

* This class of filters is known as alpha-beta or g-h filters.
  + G – How much emphasis is placed on measurements than predictions. Closer to 1 means we have minimal noise rejection
  + H – How quickly the system adapts to measurement observations. Higher value can increase the noisiness of estimates/predictions. **(Check this)**
* This class of filters also assumes that the underlying sensor works consistently and is not faulty, so given some domain knowledge about the sensor’s characteristics the true state of the system can be estimated.
* The Kalman Filter is within this class of filters, but these values are adapted through an adaptive gain term that compares Gaussians of the predictions and the measurements.
  + Paper calls out a key weakness in Kalman Filter: model mismatch (assumed wrong model of the system’s state or the model changes over time).

5 – (Data-Driven and Model-Based Spectrum)

* When it comes to mixing model-based methods with deep learning methods, we can place them on a spectrum of being entirely data-driven to being entirely dependent on domain knowledge.
* DD methods include RNNs/LSTMs, while the Kalman Filter would clearly be considered a MB method. The other model we’ve discussed through this journal club is Mamba, which is more of a model-aided network.
  + Mamba – a Data-driven method (LSTMs/transformers) is constrained with a state-space model. (Need more here)
* The paper presents a way of incorporating a data-driven approach within an already-established model-based approach, KalmanNet.
  + They specifically focus on replacing the replacing the model-based gain adaptation with data-driven gain adaptation.

## Methodology

System Estimation Pipeline

6 – LTI System Example

* State-space equations whose state and observation transitions do not vary with time. They’re constant-valued matrices. The diagram below shows how the different variables relate to each other.
  + A Kalman filter would be used to approximate the plant response y although it’s been corrupted by the additive noise v.
* On the right is an example of a generated LTI system with noisy observations given by y\_ss. I tried copying values over from Kalman Filtering just to get some examples of the filtering results and how it works.

7 – LTI System Output

* Example of the time series output, where the orange dashed line shows the true system output and the blue is the noisy output.

8 – KF Solution A

* To discuss the Kalman Filter solution as covered in the paper, I’m going to restate the state-space equations as so, where the state transitions and observations come about as functions of the state.
* In the linear example, we can view these functions as linear transition matrices.
* We’ll cover both the prediction and updates then discuss how the KalmanNet method replaces the Kalman Gain part of the Update step.

9 – KF Solution B

* Prediction Eqn. Intuition – In both instances, we are predicting not only the transition and observation but also their respective errors. So, we guess the behavior of the state and the error of our approximation.
  + Predict – There’s some initial condition for the state and for the error covariance matrix. We predict the second-order statistical moments for the error in our state estimations based on our transitions to that next step, and we do something similar for the observations of the state
* Update Eqn. Intuition – We’re now going to scale the correction to a posteriori values based on how far we are from the true measured value y\_t.
* We can essentially look at the Kalman Gain as a ratio of the errors according to state estimation and the observations, so it’s there to ensure we have that trade-off between focusing more on measurements or more on predictions.

10 – KF Approximation of LTI System

* Here’s that same LTI system with my Kalman Filter prediction of the true state, and below is the error of that estimation.
* There may be some parts where the error is greater for the Kalman Filter, but this could be due to the choice of transition functions **f** and **h** which are both set to 1.

11 – LTV System Example

* Here are our LTI state-space equations.
* And we can make a simple LTV system by replacing A with one that transitions across time, A\_t.
* The right-side shows how I made this work in Python using available functions from SciPy.

12 – LTV System Output

* Here’s an example of what the LTV System output is like with the same true and noisy signal setup in orange and blue, respectively.

13 – KF Approximation of LTV System

* And here, I show two examples of my Kalman Filter trying to decrease the noise of the system over time.
* KF still does well, and as in the g-h filter framework, it’s possible to decrease the influence of predictions by assuming a smaller transition value for **f**.
* Note that the noise is further rejected in this instance, and the system estimation is even better than before.

14 – Extended Kalman Filter (EKF)

* Now let’s dig deeper and ask the question, what if our state transitions and observations depend on nonlinear matrices?
  + Answer: Then we must set the **f** and **h** to be nonlinear functions even if we don’t know the underlying model characteristics.
* That’s easy to do. But what about the second-order statistical moments?
* It’s important to remember here that both the states/observations and their errors are being transitioned, so to get an equivalent **F** and **H** for the nonlinear Kalman Filter case, I must approximate the linear transitions.
* The way this is done is by a Jacobian of **f** or **h** w.r.t. the posteriors and priors.
* Jacobians evaluated between time points of the data.
  + Why a Jacobian for approximating the second-order statistical moments?
    - Previously, the transition matrices were linear multiplications on top of each state variable, so the linearization approximates this to get the second-order statistical moments for the error covariance matrix.

14 – EKF Diagram + Issues

* Here’s the diagram they give on the paper, so if anyone has questions let us know on this slide. This way, we can clear up whatever might be causing you trouble here.
* Although KF variations like EKF, UKF, and PF have worked well for nonlinear state estimation problems, there are still some issues that can come up.
  + KG and other components largely depend on **domain knowledge** and an **accurate choice of noise statistics**.
  + Additionally, the **linearization** of a function might not be able to handle **severely nonlinear** systems or very **chaotic** systems.

15 – KalmanNet Diagram

* To address these issues, this entire part of the **update step** is getting replaced with a recurrent neural network.
* This removes the dependence on knowing the underlying noise statistics and on linear approximations of the nonlinear functions.
* So, the authors put off adaptation of the Kalman Gain to a neural network based on inputs and whatever the target should be, which Zhenjiang will go through later.

16 – KalmanNet Input Options

* That said, …
  + From which input features (signals) will the network learn the KG?
  + What should be the architecture of the internal RNN?
  + How will this network be trained from data?
* The authors propose a few ways of incorporating the error information into the network to learn the error covariances and ultimately the Kalman Gain.
  + F1 – Observation Difference
  + F2 – Innovation Difference
  + F3 – Forward Evolution Difference
  + F4 – Forward Update Difference
* In practice, they found that the use of (F1, F2, F4) or (F1, F3, F4) worked best. This makes sense because the network needs knowledge of the state-evolution process and the uncertainty of our state estimates.

17 – KalmanNet Architecture #1

* Less constrained version of KalmanNet with no mention of the Q, Sigma, or S to compute the Kalman Gain

18 – KalmanNet Architecture #2 (probably 2 more slides here)

* More constrained version of KalmanNet that explicitly makes computations dependent on each other and starts off with some initial conditions.

19 – KalmanNet Training

* Optimization of the gradient is done with all different versions of BPTT with each having their advantages and disadvantages.

20 – KalmanNet Training

* Jkl

## Experimental Results

19 – LTV System

20 – Lorenz Attractor

## Final Thoughts

21 – Conclusions