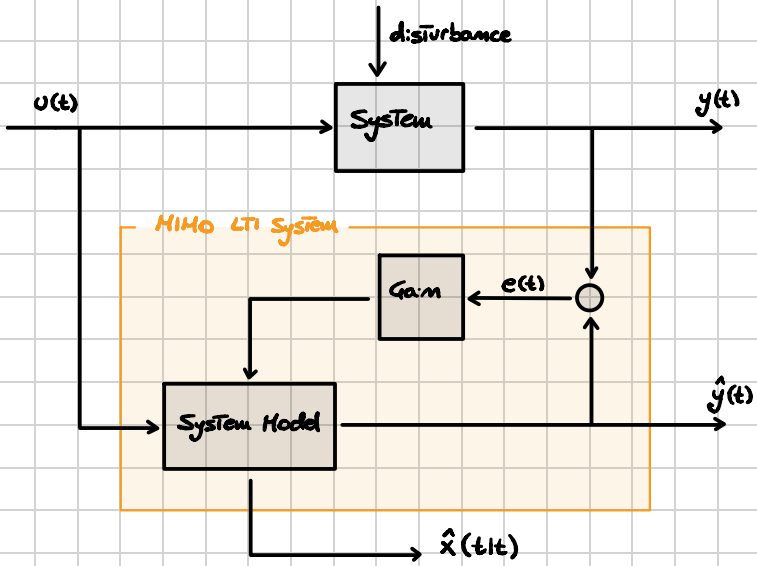
4- Software Sensing with Black-Box Methods

In chapter 3 we have seen software sensing with kalman Filter (model-based with feedback)



Main feature of KF:

- A white-box model is needed
- In principle, a training dataset is NOT required (even if, in practice, a dataset is needed To find V_1 and V_2 empirically)
- It's a constructive method based on a feedback scheme
- Can estimate TOTALLY unmeasurable states

Is it possible To solve The very same problem in a black-box way? Yes, since we can recast The problem as a system identification problem, but The state $x(t)$ To be estimated must be measured in The training stage

Now The starting point is a "classic" Training dataset

$\{u(1) \dots u(N)\}$ Input
$\{y(1) \dots y(N)\}$ Output
$\{x(1) \dots x(N)\}$ States Vectors (must be available for Training - Supervised learning)

We wish To estimate a mathematical model with The I/O relationship $u(t) + y(t) \rightarrow \hat{x}(t)$

We need To estimate a mathematical model with This architecture:

$$\hat{x}(t|t) = \underbrace{TF_{u(t)\rightarrow y(t)}}\, u(t) + \underbrace{TF_{y(t)\rightarrow \hat{x}(t|t)}}\, y(t)$$

So: $\hat{x}(t|t) = S_{ux}(z;\theta)\, u(t-1) + S_{yx}(z;\theta)\, y(t)$

parametric models used to estimate optimal θ

θ: parameter vector containing The coefficient of both S_{ux} and S_{yx}

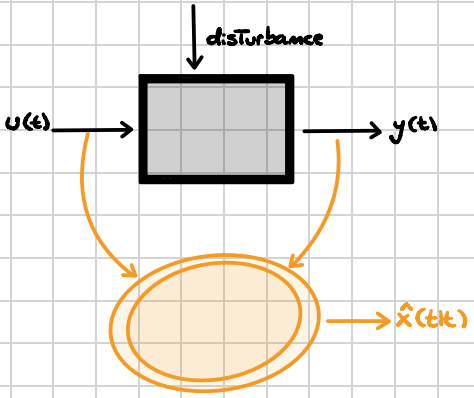
Classical parametric approach: $J_N(\theta) = \frac{1}{N} \sum \left(x(t) - \left(S_{ux}(z;\theta)\, u(t-1) + S_{yx}(z;\theta)\, y(t)\right)\right)^2$

(measured states; Modeled / Estimated state)

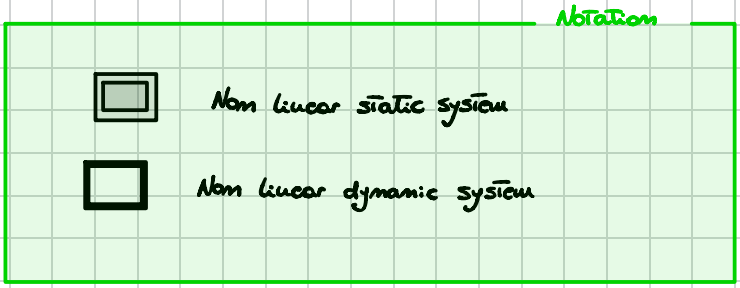
Optimization step: $\hat{\theta}_N = \arg\min_\theta J_N(\theta) \Rightarrow \hat{x}(t|t) = S_{ux}(z;\hat{\theta}_N)\, u(t-1) + S_{yx}(z;\hat{\theta}_N)\, y(t)$

Black-Box Software Sensing

Once This software-sensor has been estimated we can remove The physical sensor of The state $x(t)$ and replace with This algorithm. The above System identification can be done with many different system identification Technique (eg. 4SID)

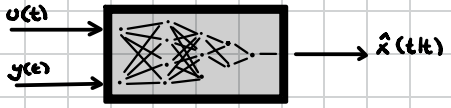


When the system is non linear, we can use the same idea but we need a non-linear models



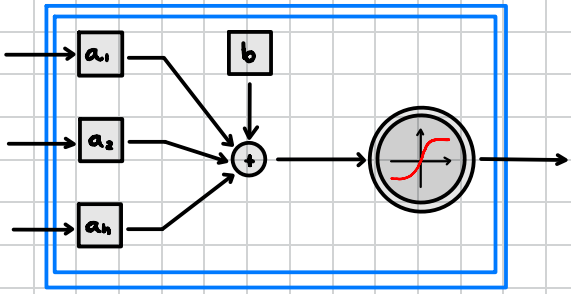
Now we see 3(+1) architectures for non-linear software sensors (emphasis on how to manage the dynamical part of the system)

1. Use Recurrent Dynamical Neural Networks

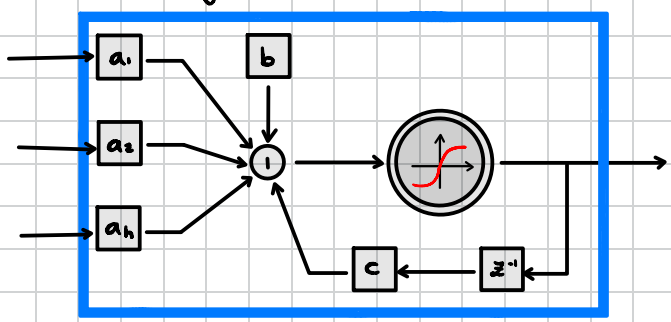


We can have two types of neurons:

Static Neurons



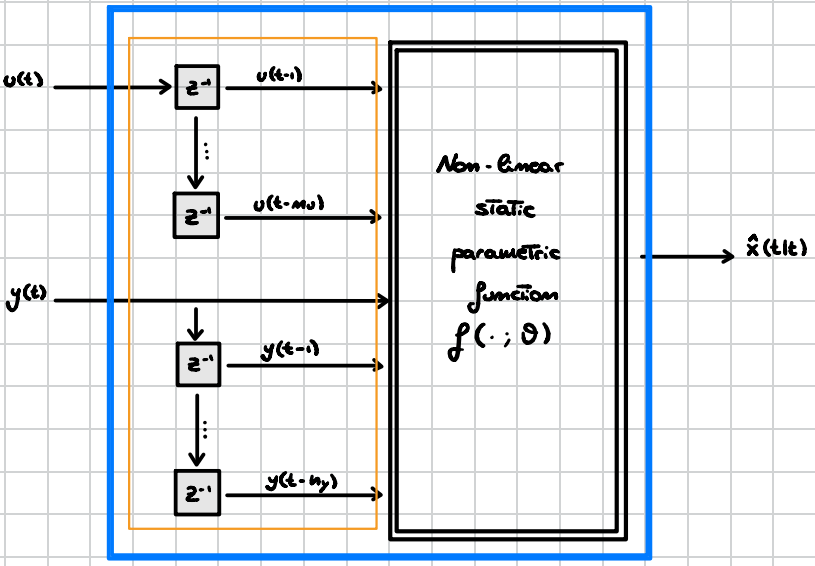
Dynamical Neuron



A network with dynamical neurons is a very general non linear and dynamical system that can be optimized to find optimal SW sensing.

- Most general and intuitive solution
- Not the most used because
  - computational effort for training is very high
  - stability of the obtained solution is very difficult to guarantee

2. Split the software sensor into a dynamic/linear part and a static/non-linear part with a non recursive FIR like scheme



The optimization can be done only on the second part of the system.

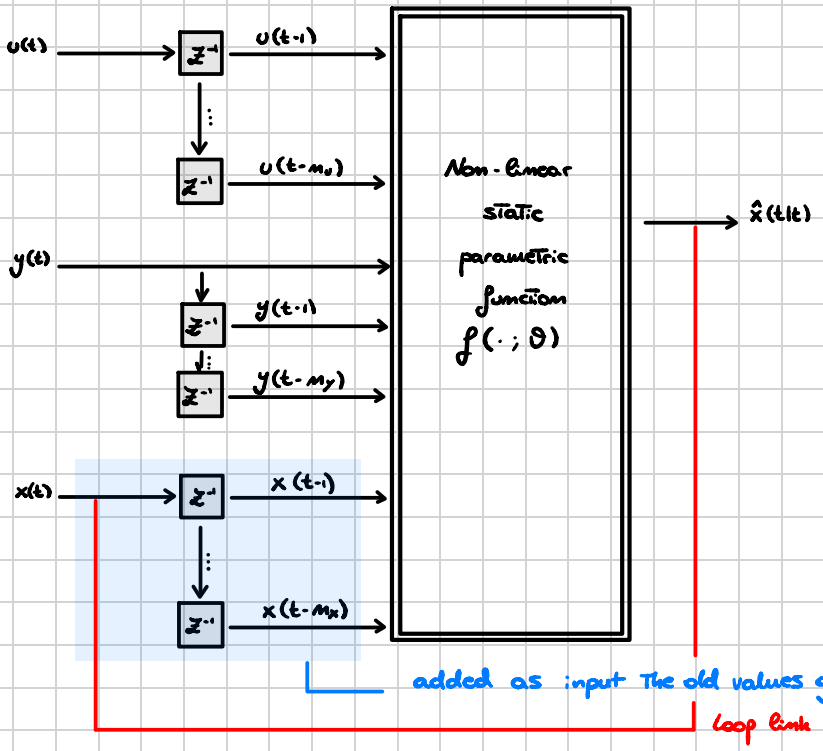
Main features

- stability is guaranteed by construction (FIR-like architecture)

Problem: high dimension of the input vector of neural network function: assume that the system is MIMO

$u(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_{n_u}(t) \end{bmatrix}$ $y(t) = \begin{bmatrix} y_1(t) \\ y_{n_y}(t) \end{bmatrix}$ $x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_{n_x}(t) \end{bmatrix}$ $\Rightarrow f(\cdot;\theta): \mathbb{R}^{(n_u \times m_u)+(n_y \times m_y)} \longrightarrow \mathbb{R}^{n_x}$

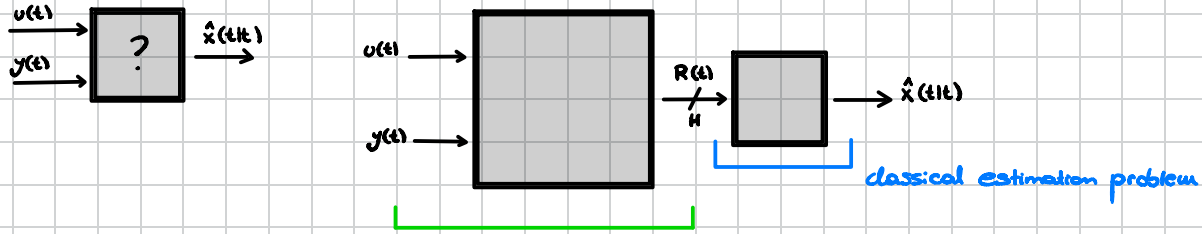
3. Like Architecture 2 but with a Recursive feedback (like IIR filters)



Advantage: m_u and m_y can be much smaller (Typical advantage of IIR over FIR. With FIR, to build memory of the past we need a lot of delays; with IIR the recursive part brings memory of the past - just few delays are enough)

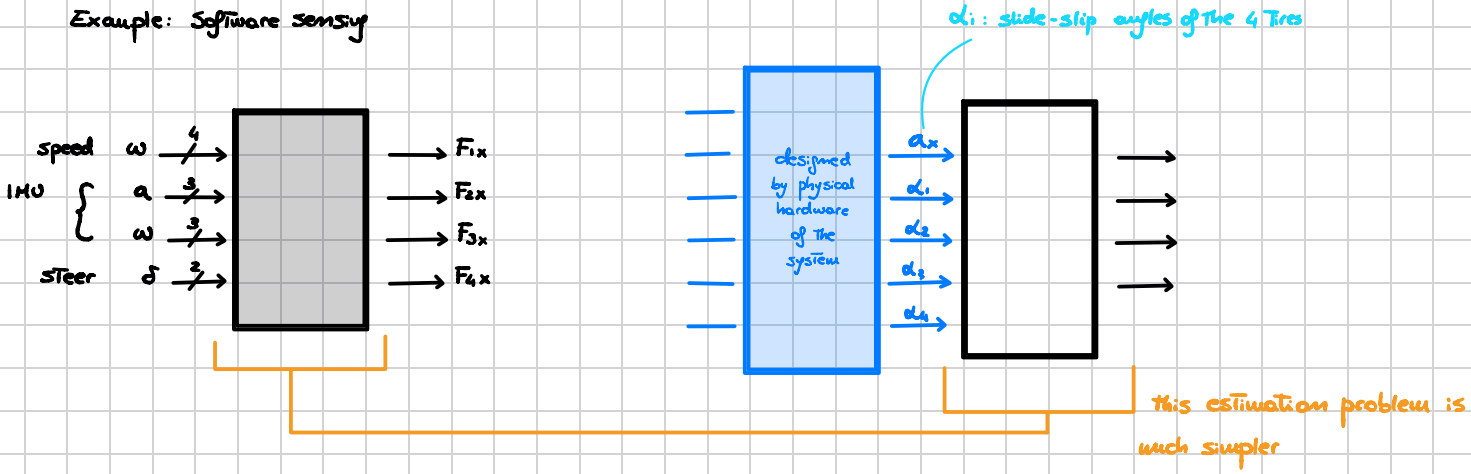
Disadvantage: in Training we can directly use the measured $x(t)$; in "production" $x(t)$ must be replaced with feedback on the estimated $\hat{x}(t|t)$. We introduce a feedback loop that can be critical for stability of the system

4. Meta Architecture: must be applied to #1, #2, #3 and requires a priori knowledge of the system



pre processing Stage: using physical know-how of the system, we move from large number of $u(t)$ and $y(t)$ signals to a small number of new build signals called regressors

Example: Software sensing



The preprocessing block need an expert of the specific system

Usually the number of regressors is much smaller than the number of u and y signals. Usually, also the relationship regressor → $\hat{x}$ is simpler (ex. nonlinear and not dynamic, or only dynamic and not nonlinear)

Try #4 if you can!

Summary comparison of model based and black box software sensing

	kalman filter	Black Box
Need of a physical model of the system	✓	x
Need of a training dataset	In principle No, in practice Yes	✓
Interpretability of the obtained SW-sensor	✓	x
Easy retuning of a similar system	✓	x
Accuracy of software sensor	✓	✓✓
Can be used for unmeasurable states	✓	x