https://www.halvorsen.blog

PID + Kalman Filter + MPC LabVIEW Applications

Hans-Petter Halvorsen

PID + Kalman Filter + MPC LabVIEW Applications

Overview:

- Part 1 – Introduction to LabVIEW Application
- Part 2 – Building Simulator
- Part 3 – PID Control
- Part 4 – Feedforward using Kalman Filter
- Part 5 – MPC (with Kalman Filter)

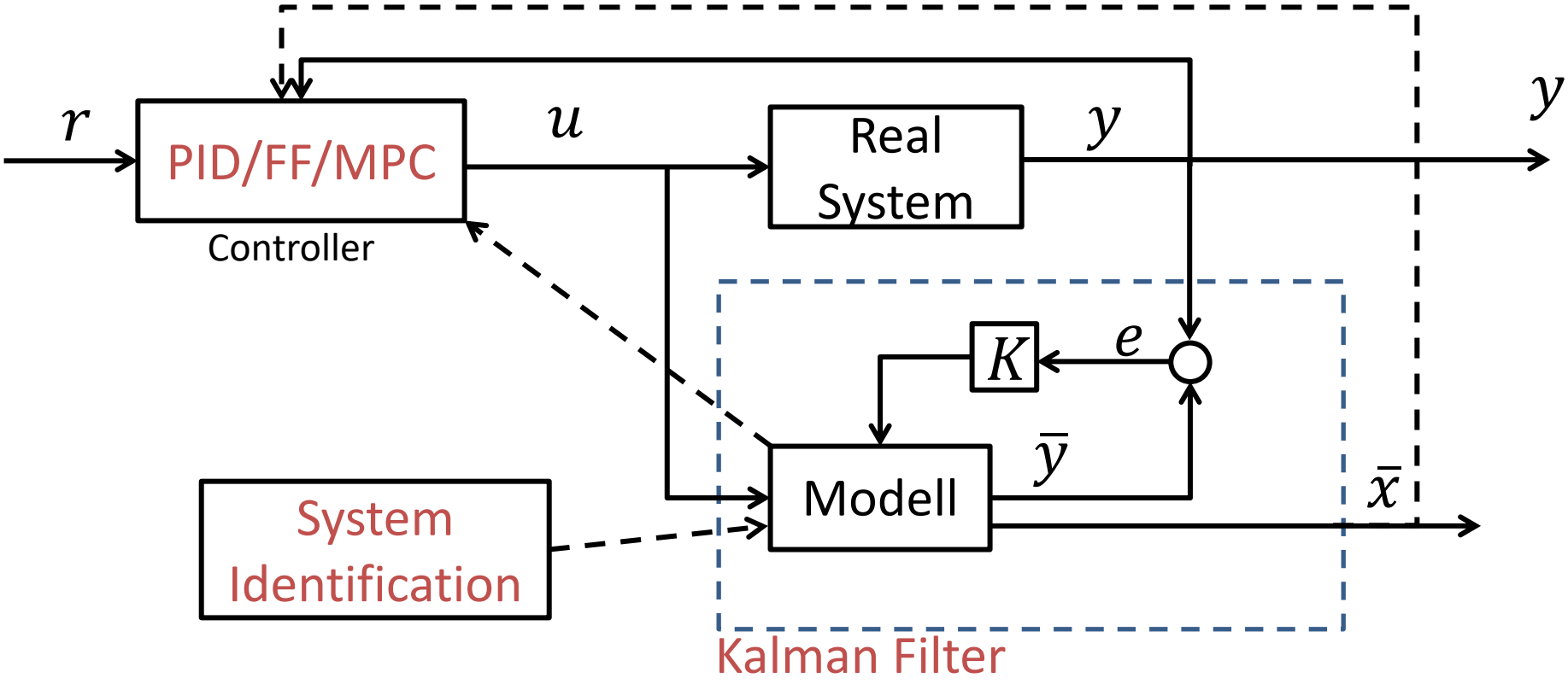
Introduction

In this Video we will Control a Level Tank System using different control strategies:

- PID
- Use Kalman Filter for Estimation of Unknown Process Variables/Measurements
- PID + Feedforward (with help of Kalman Filter)
- MPC (+ Kalman Filter)

We will not go in depth of the theory, but focus on the practical implementation in LabVIEW

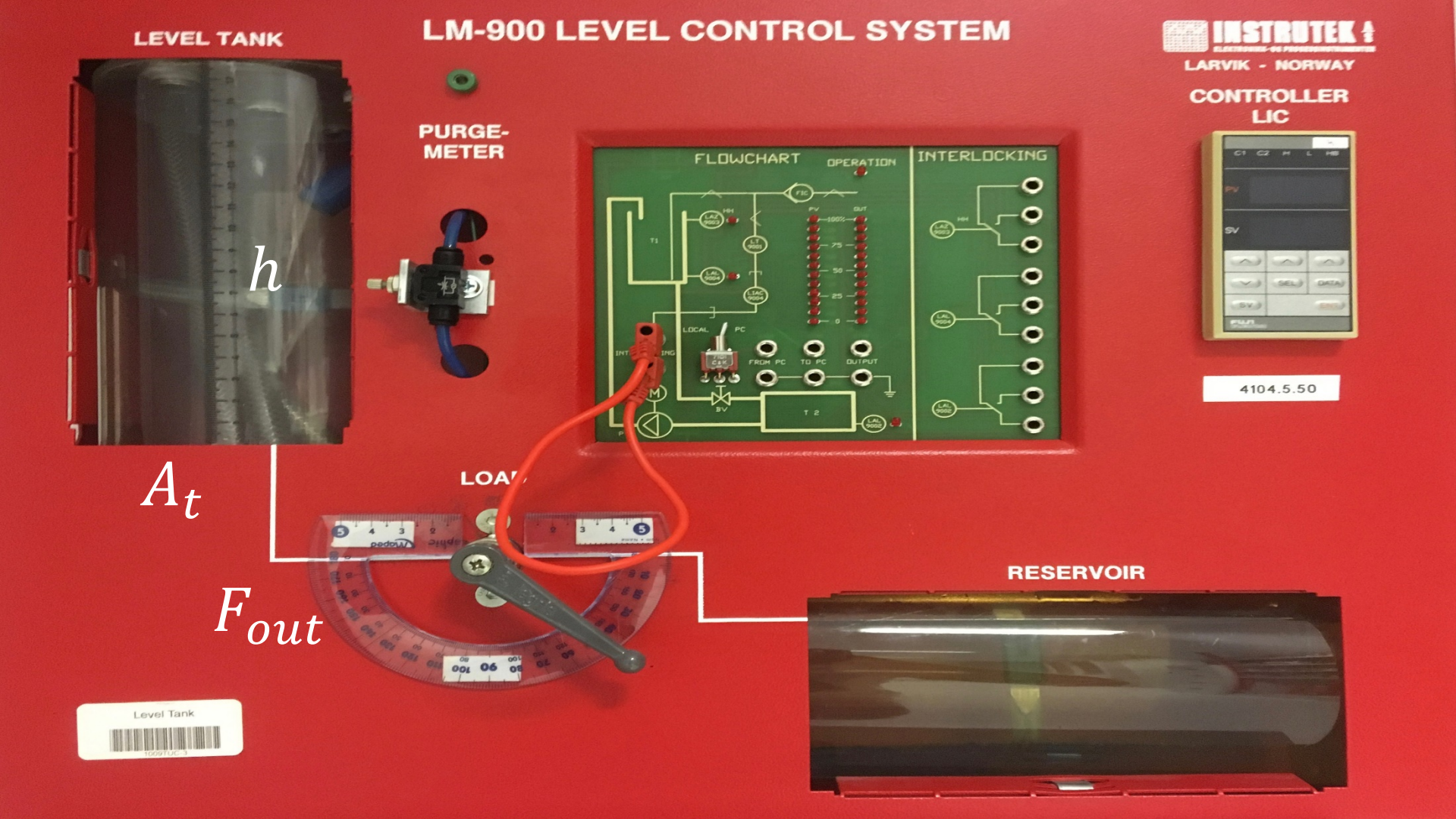
System Overview



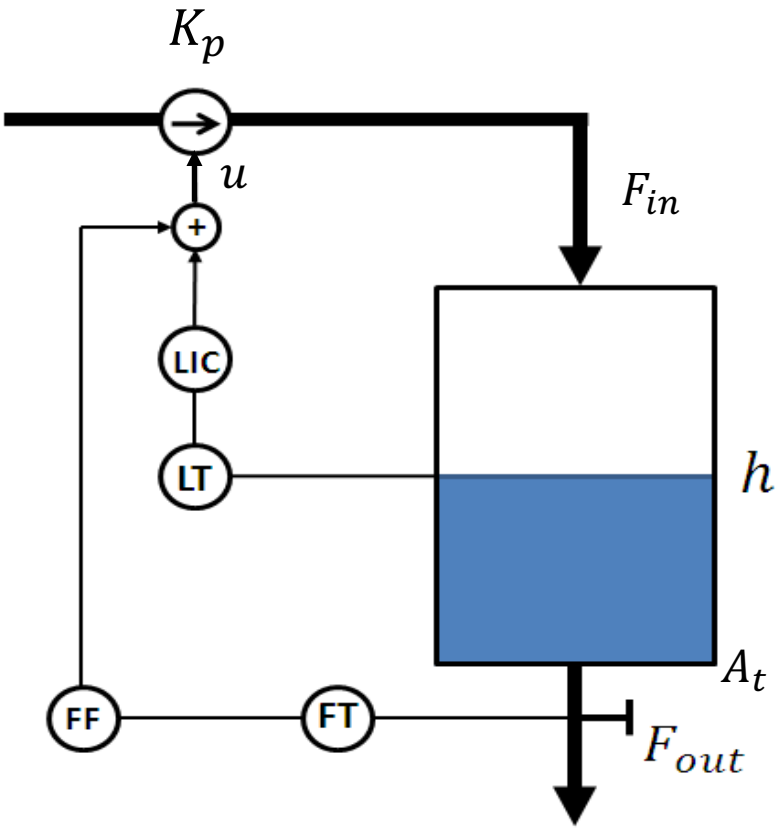
https://www.halvorsen.blog

Level Tank Process

Hans-Petter Halvorsen


LEVEL TANK
LM-900 LEVEL CONTROL SYSTEM
INSTRUTEK
LARVIK - NORWAY
CONTROLLER
LIC
PURGE-
METER
FLOWCHART
OPERATION
INTERLOCKING
LOCAL
PC
FROM PC
TO PC
OUTPUT
4104.5.50
h
A_t
F_{out}
RESERVOIR
Level Tank

Level Tank



$$A_t \frac{dh}{dt} = F_{in} - F_{out}$$

or:

$$\dot{h} = \frac{1}{A_t}(K_p u - F_{out})$$

Where:

- F_{in} - flow into the tank , F_{in}= $K_p u$
- F_{out} - flow out of the tank
- A_t is the cross-sectional area of the tank

Level Tank

$$\dot{h} = \frac{1}{A_t}(K_p u - F_{out})$$

$\dot{F}_{out} = 0$ Assumption: $F_{out} \approx$ constant

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

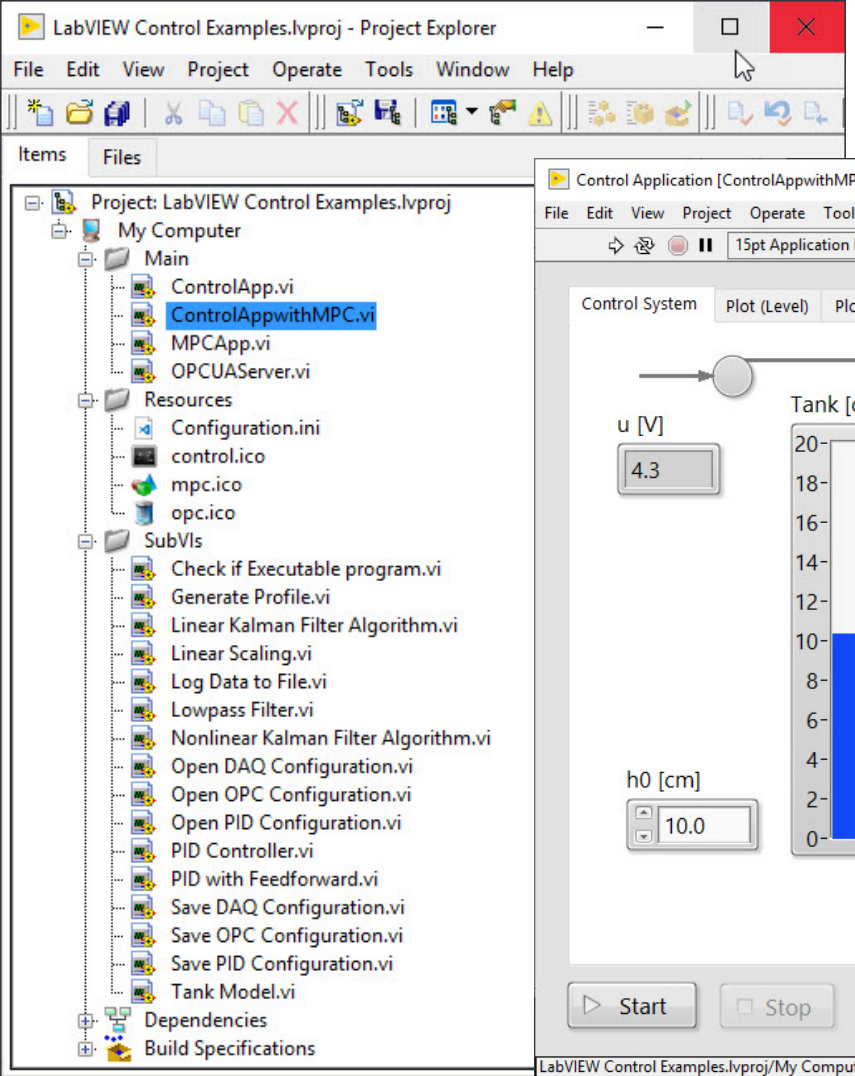
$x_1 = h$
$x_2 = F_{out}$

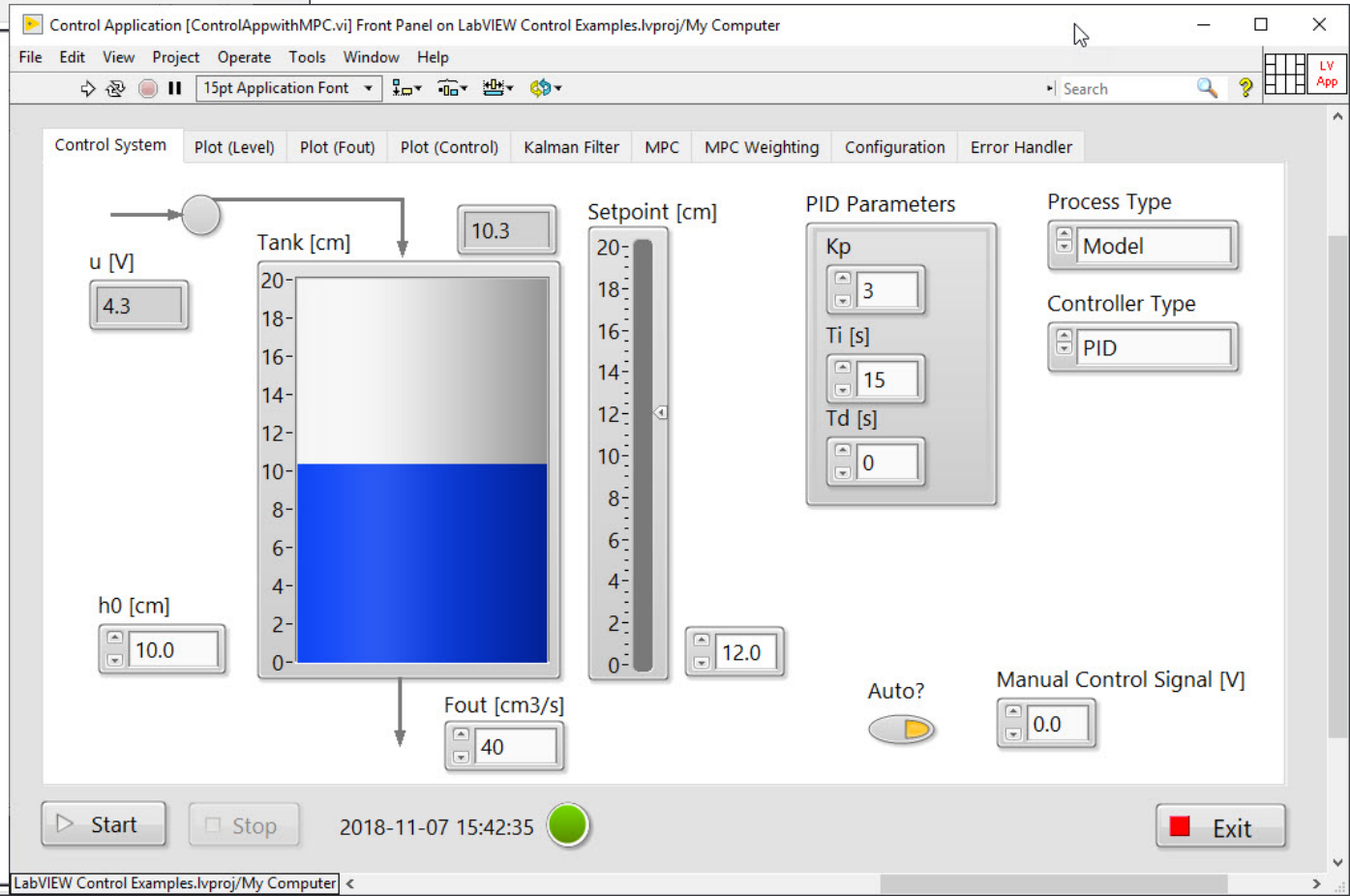
$$\dot{x}_1 = -\frac{1}{A_t}x_2 + \frac{1}{A_t}K_p u$$
$\dot{x}_2 = 0$
$y = x_1$

$$\begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \underbrace{\begin{bmatrix}0 & -\frac{1}{A_t} \\ 0 & 0\end{bmatrix}}_{A}\begin{bmatrix}x_1 \\ x_2\end{bmatrix} + \underbrace{\begin{bmatrix}\frac{K_p}{A_t} \\ 0\end{bmatrix}}_{B}u$$

$$y = \underbrace{[1 \quad 0]}_{C}\begin{bmatrix}x_1 \\ x_2\end{bmatrix}$$

LabVIEW Application





Demo

https://www.halvorsen.blog

https://www.halvorsen.blog

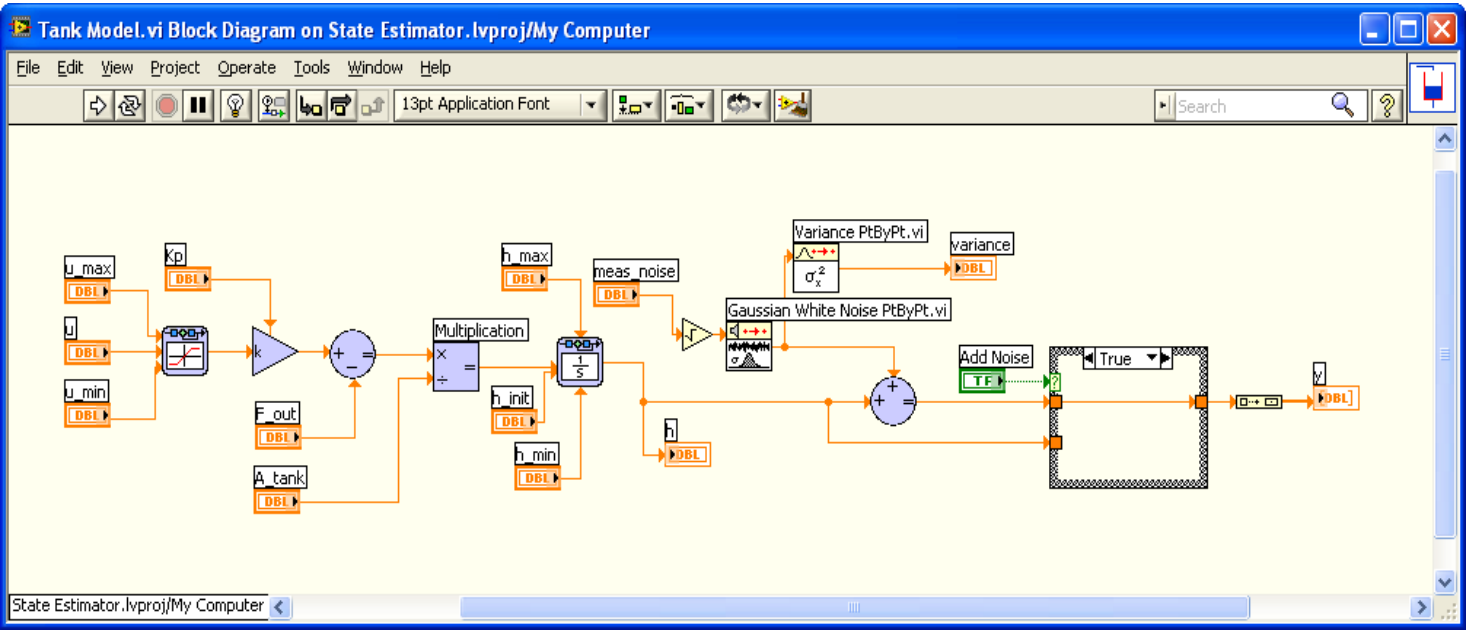
Level Tank Simulator

Hans-Petter Halvorsen

Level Tank Model Implementation in LabVIEW

We have implemented it as a Simulation Subsystem in LabVIEW:

$$\dot{h} = \frac{1}{A_t}\left[K_p u - F_{out}\right]$$



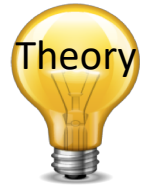
We need to find unknown Parameters -> System Identification

https://www.halvorsen.blog

System Identification

Hans-Petter Halvorsen

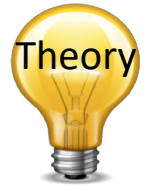
System Identification and Estimation



- **System Identification:** System Identification uses statistical methods to build mathematical models of dynamical systems from measured data
- **State Estimation:** Use of mathematical models in order to estimate the internal states of a process. Example: Kalman Filter

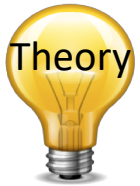
LabVIEW has built-in functionality for both System Identification and State Estimation

System Identification Categories



We have 2 main categories of System Identification:

- **Parameter Estimation** based on that we have developed a mathematical model using the laws of physics (Mechanistic Models) and you want to find the unknown model parameters. Here we can use least squares method, etc. The unknown parameters are then found from experimental data.
- **Black-box / Subspace methods**: System Identification based on that you do not have a mathematical model available. The models (Empirical Models) are found from experimental data only using advanced algorithms.



System Identification

Datalogging from Real System (Experimental Data)

Physical Knowledge

Mechanistic Models

Empirical Models

PLS/PCR, Black-box, Subspace, Wavelet, etc.

Finding mathematical model(s) using the laws of physics/first principles

The model is found from experimental data

Empirical modelling refers to any kind of (computer) modelling based on empirical observations rather than on mathematically describable relationships of the system modelled.

Datalogging from Real System (Experimental Data)

Parameter Estimation

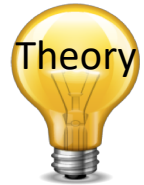
The unknown Parameters within the model(s) needs to be found

Example of unknown Parameters: Pump gain, Valve constants, etc.

Trial and Error, Step Response, Least Square Methods, etc.

Some of these can be found in data sheets, etc., while others is not so easy to find. Then Parameter Estimation is a good method to find these.

System Identification Methods

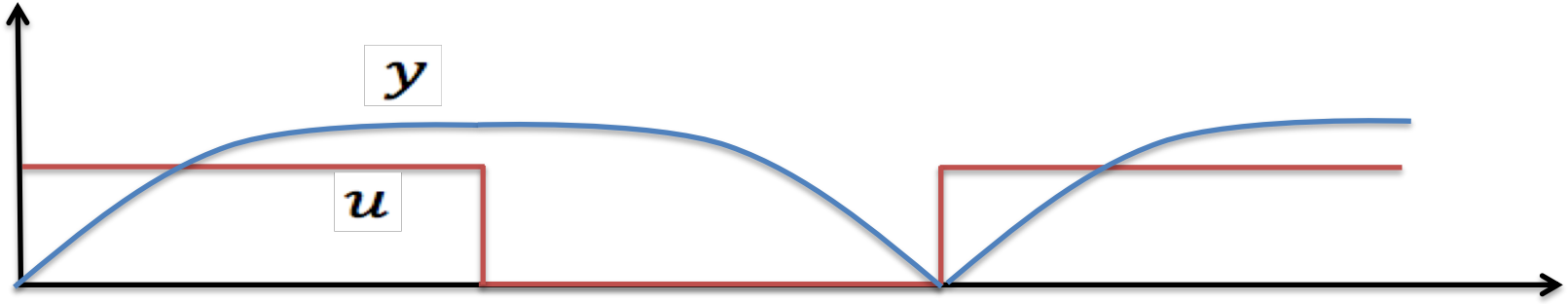


Suggestions: Find the Model Parameters using:

- The **Least Square Method** $\theta_{LS} = (\Phi^{T}\Phi)^{-1}\Phi^{T}Y$
- Then adjust and fine-tune the Model Parameters using the "**Trial and Error**" method if necessary
- It is advised that you use at least 2 different methods for comparison.
- Other relevant methods may be: "Step Response" method, "Sub-space" methods, built-in methods in LabVIEW/MATLAB, etc.

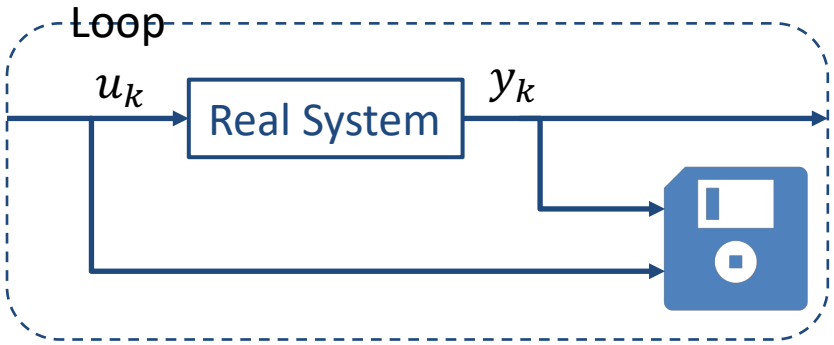
System Identification – Step by step

1. Exite the Real System, e.g.:



2. Log Data

Log Data to File, Database, etc.



3. Use the Logged Data to find the model or the model parameters

4. Validate the Model against logged data and/or the real system

Split the Logged Data into 2 parts:

- One part should be used for finding the Model (Parameters)
- The other parts should be used for Model Validation

Least Square Example

Given:

$$\dot{x} = ax + bu$$

We want to find the unknown *a* and *b*.

This gives:

$$\underbrace{\dot{x}}_{y} = \underbrace{[x \quad u]}_{\varphi} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\theta}$$

i.e.,:

$$\theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

Then we need to discretize:

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s}$$

This gives:

$$\underbrace{\frac{x_{k+1} - x_k}{T_s}}_{y} = \underbrace{[x_k \quad u_k]}_{\varphi} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\theta}$$

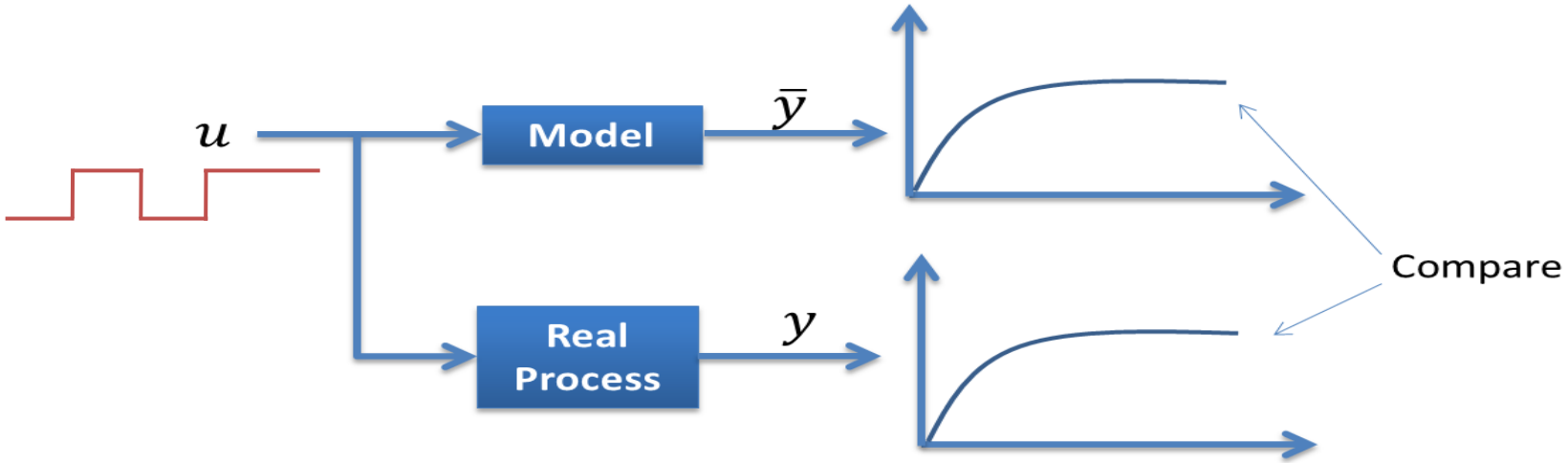
Based on logged data we get:

$$\underbrace{\begin{bmatrix} \vdots \\ \frac{x_{k-1} - x_{k-2}}{T_s} \\ \frac{x_k - x_{k-1}}{T_s} \\ \frac{x_{k+1} - x_k}{T_s} \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} \vdots & \vdots \\ x_{k-2} & u_{k-2} \\ x_{k-1} & u_{k-1} \\ x_k & u_k \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\theta}$$

The we find the uknows a and b using LS:

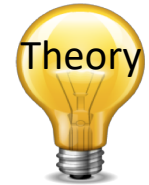
$$\theta_{LS} = (\Phi^T \Phi)^{-1} \Phi^T Y$$

Trial & Error Method



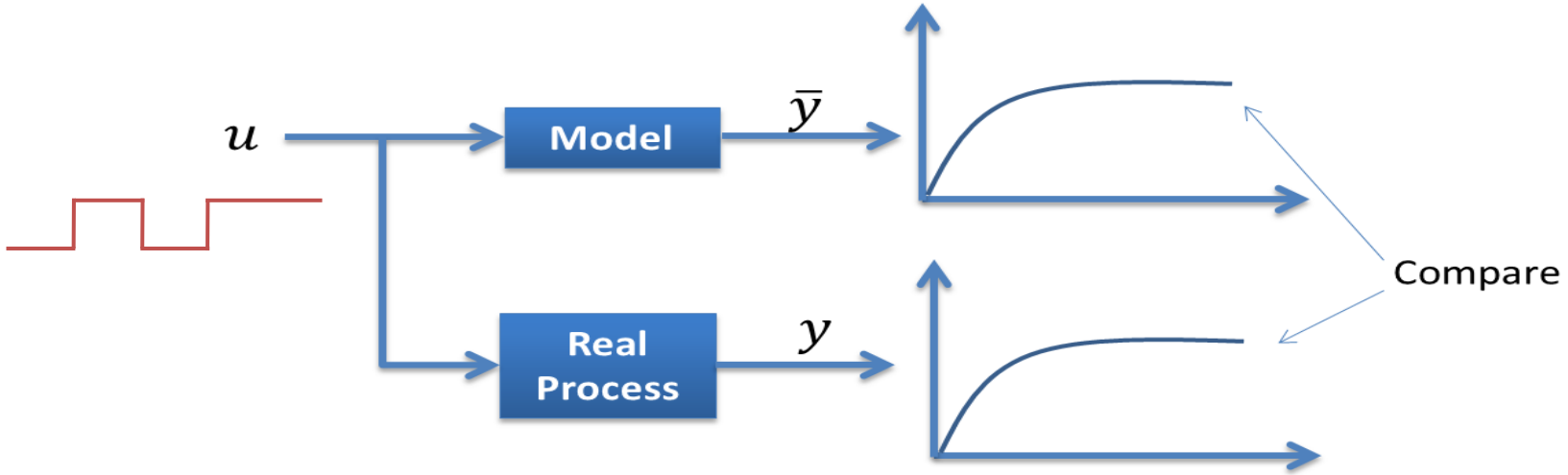
Adjust model parameters and then compare the response from the real system with the simulated model. If they are “equal”, you have probably found a good model (at least in that working area)

Model Validation



Make sure to validate that your model works as expected

Example of simple model validation:



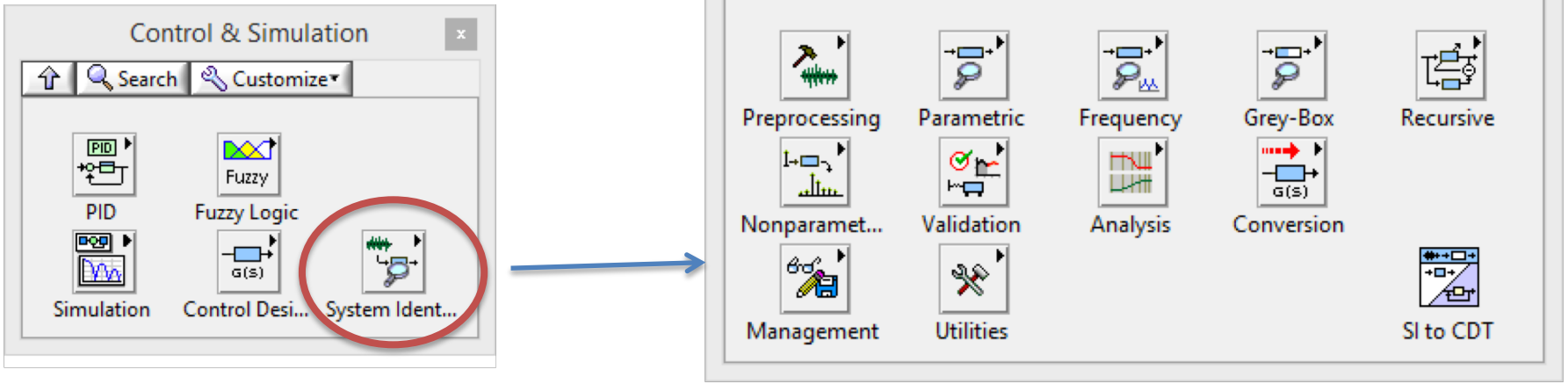
https://www.halvorsen.blog

System Identification in LabVIEW

Hans-Petter Halvorsen

System Identification in LabVIEW

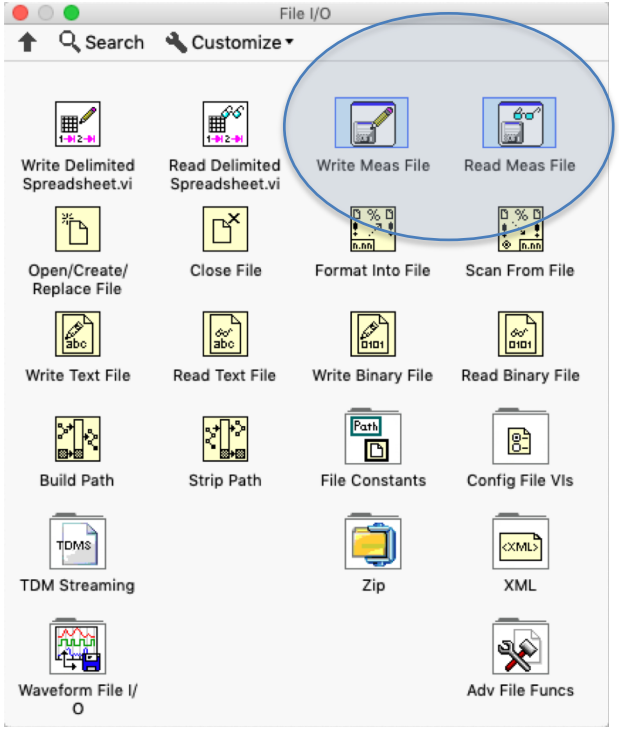
"LabVIEW Control Design and Simulation Module" has built-in features for System Identification



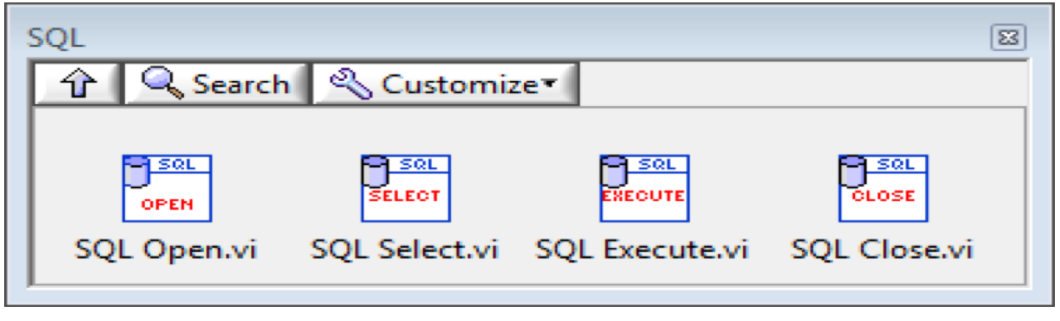
System Identification in LabVIEW

Datalogging

Measurement File

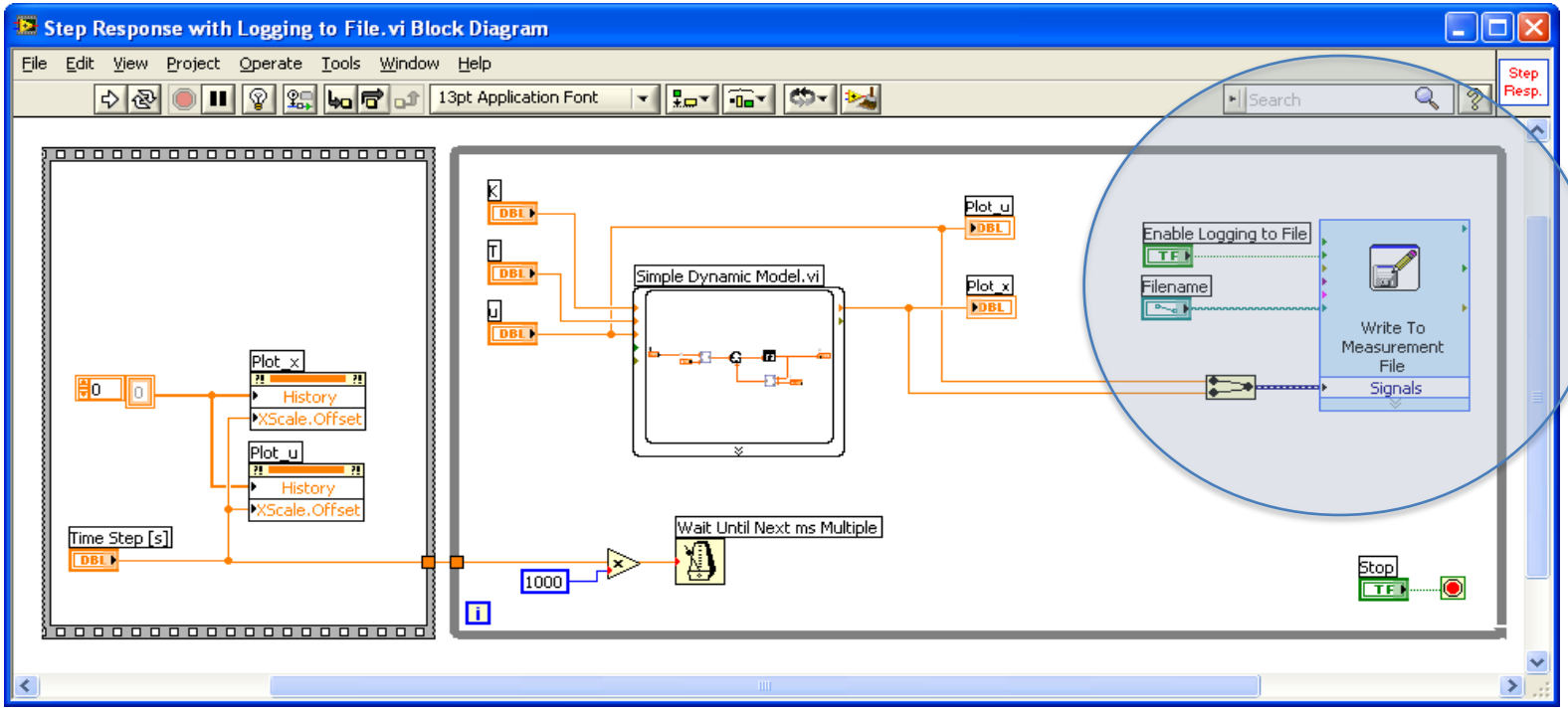


Database



https://www.halvorsen.blog

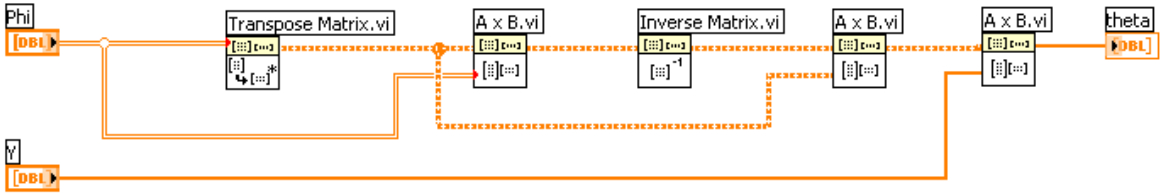
Datalogging Example in LabVIEW



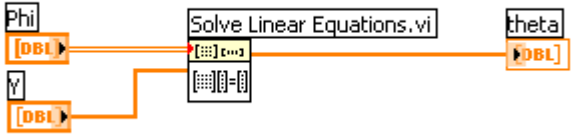
Least Square Example in LabVIEW

$$\theta_{LS} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\mathrm{Y}$$

Least Square Formula implemented from scratch:

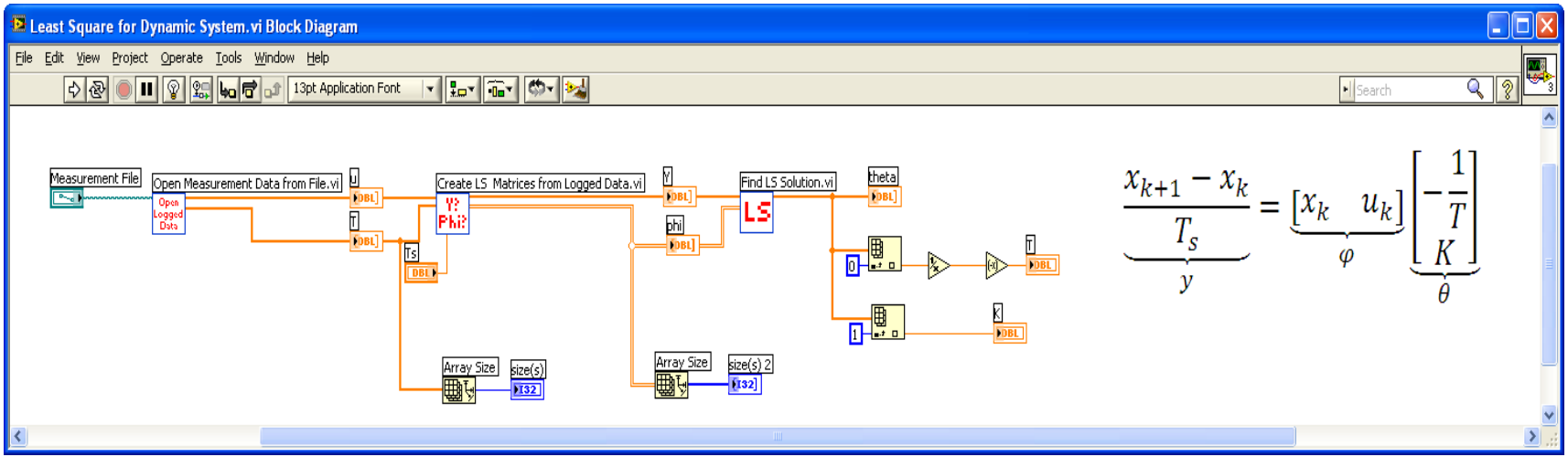


We can also use the built-in “Solve Linear Equations.vi” directly:



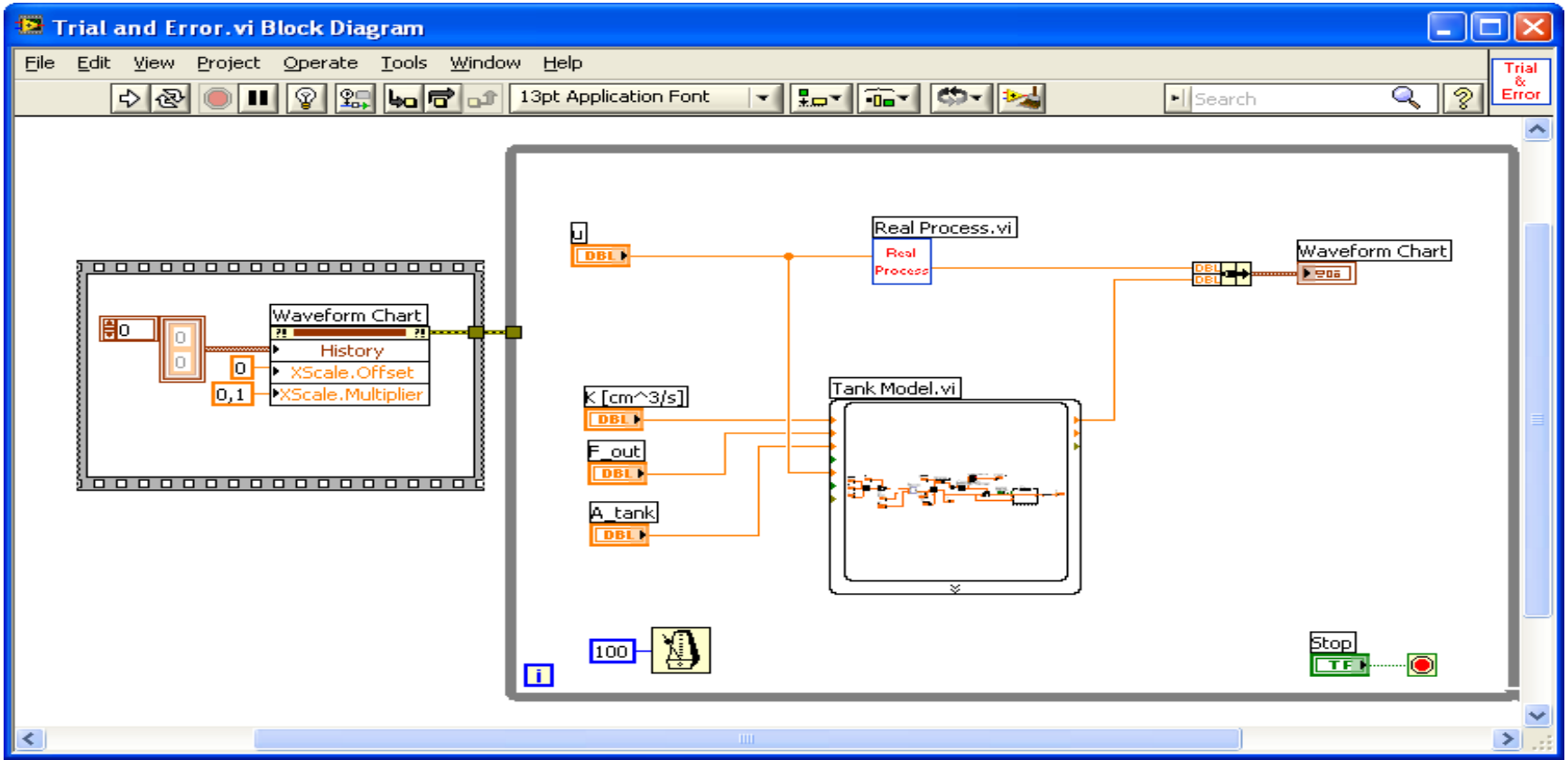
Least Square Example in LabVIEW

$$\theta_{LS} = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{\mathrm{T}}\mathrm{Y}$$



$$\underbrace{\frac{x_{k+1}-x_k}{T_s}}_{y} = \underbrace{\begin{bmatrix} x_k & u_k \end{bmatrix}}_{\varphi} \underbrace{\begin{bmatrix} -\frac{1}{T} \\ K \end{bmatrix}}_{\theta}$$

Trial & Error in LabVIEW



Water Tank Model Values

The following values was found using System Identification:

$$\dot{h} = \frac{1}{A_t}[K_p u - F_{out}]$$

$$A_t = 78.5\ cm$$

$$K_p = 16.5\ cm^3/s$$

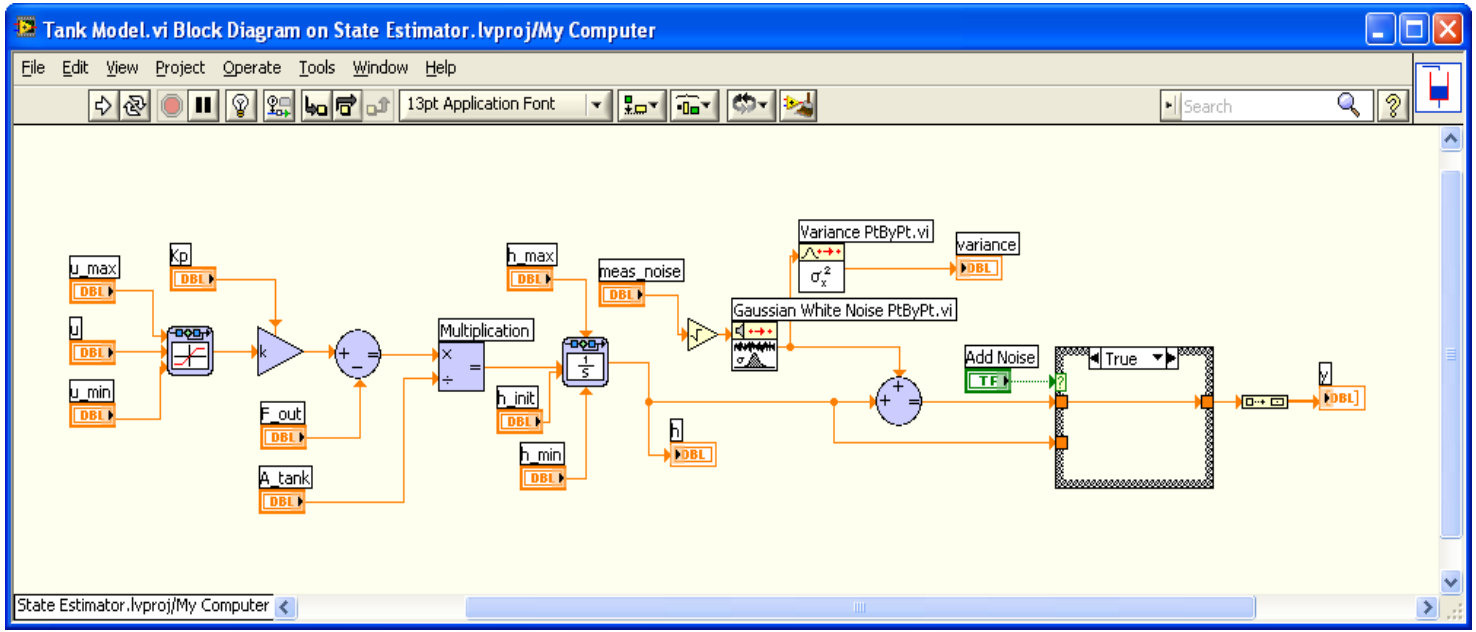
F_{out} should be adjustable from your Front Panel

The range for F_{out} could, e.g., be $0 \leq F_{out} \leq 40 cm^3/s$

Level Tank Model Implementation in LabVIEW

We have implemented it as a Simulation Subsystem in LabVIEW:

$$\dot{h} = \frac{1}{A_t}\left[K_p u - F_{out}\right]$$

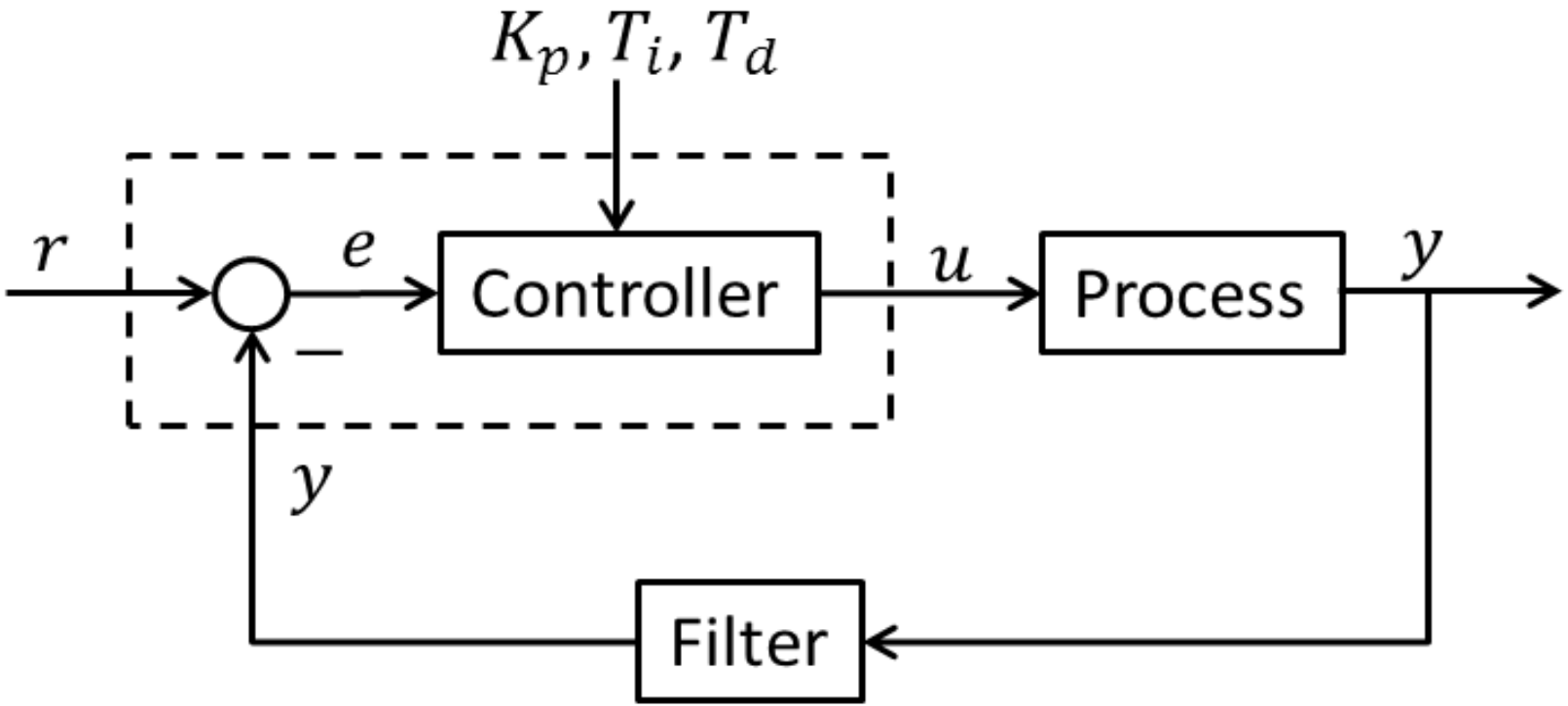


https://www.halvorsen.blog

PID Control

Hans-Petter Halvorsen

Feedback (PID) Control



The PID Algorithm

$$u(t) = K_p e + \frac{K_p}{T_i}\int_0^t e d\tau + K_p T_d \dot{e}$$

Where u is the controller output and e is the control error:

$$e(t) = r(t) - y(t)$$

r is the Reference Signal or Set-point

y is the Process value, i.e., the Measured value

Tuning Parameters:

K_p Proportional Gain

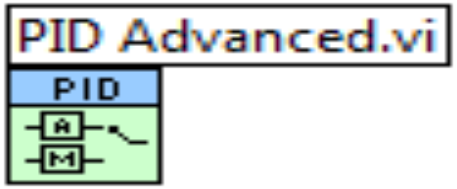
T_i Integral Time [sec.]

T_d Derivative Time [sec.]

https://www.halvorsen.blog

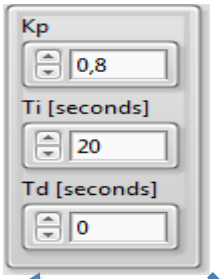
PID in LabVIEW

Hans-Petter Halvorsen



PID in LabVIEW

Front Panel

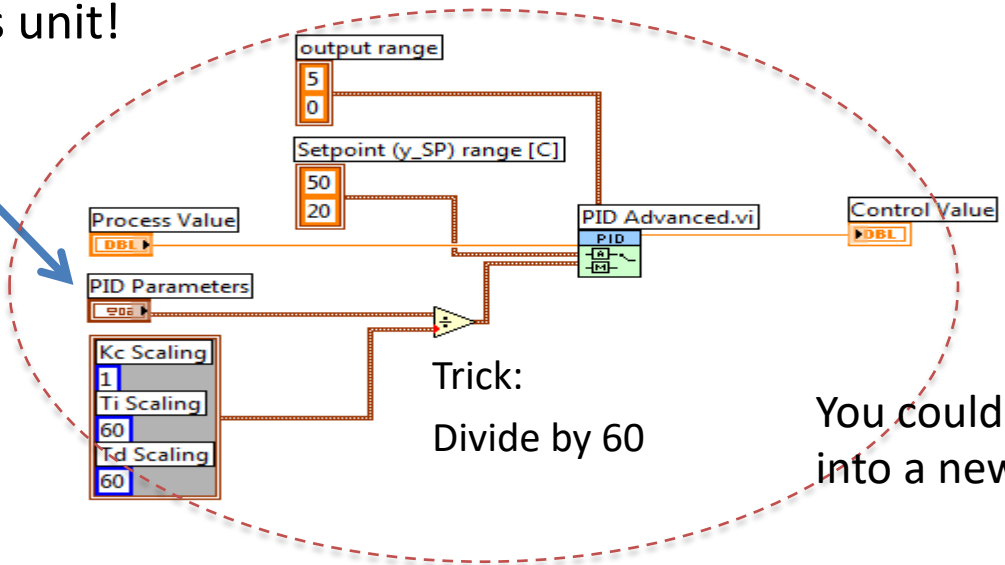


Cluster

Normally we use <u>seconds</u> as unit for Ti and Td (which is recommended!)

But the built-in PID algorithm in LabVIEW uses <u>minutes</u> as unit!

Block Diagram:



Trick:

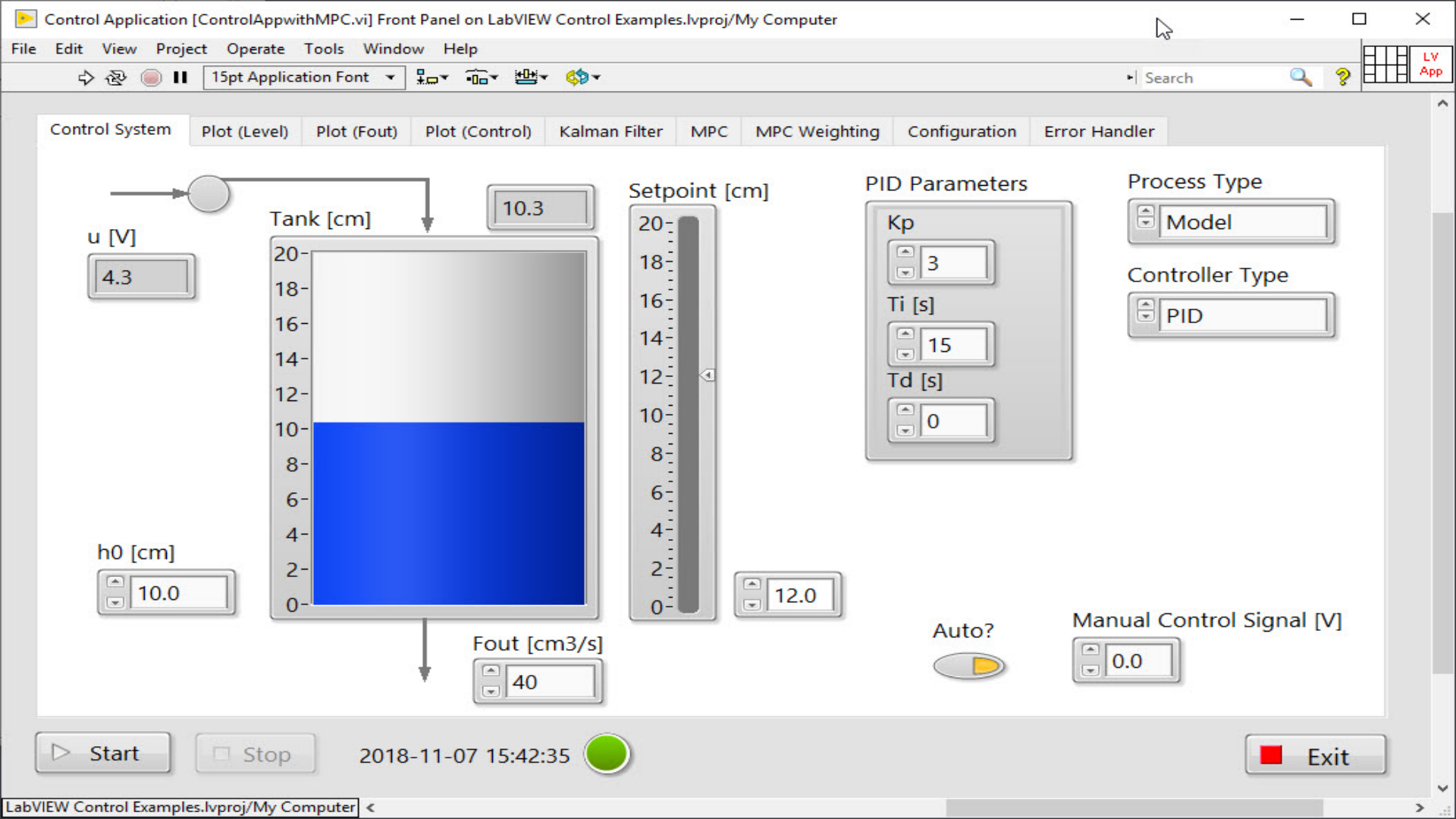
Divide by 60

You could also put this code into a new SubVI

https://www.halvorsen.blog

PID
LabVIEW Application

Hans-Petter Halvorsen


Control Application [ControlAppwithMPC.vi] Front Panel on LabVIEW Control Examples.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
LV App
Control System
Plot (Level)
Plot (Fout)
Plot (Control)
Kalman Filter
MPC
MPC Weighting
Configuration
Error Handler
u [V]
4.3
Tank [cm]
10.3
Setpoint [cm]
PID Parameters
Kp
3
Ti [s]
15
Td [s]
0
Process Type
Model
Controller Type
PID
h0 [cm]
10.0
12.0
Fout [cm3/s]
40
Auto?
Manual Control Signal [V]
0.0
Start
Stop
2018-11-07 15:42:35
Exit
LabVIEW Control Examples.lvproj/My Computer

Demo

https://www.halvorsen.blog

https://www.halvorsen.blog

Feedforward Control

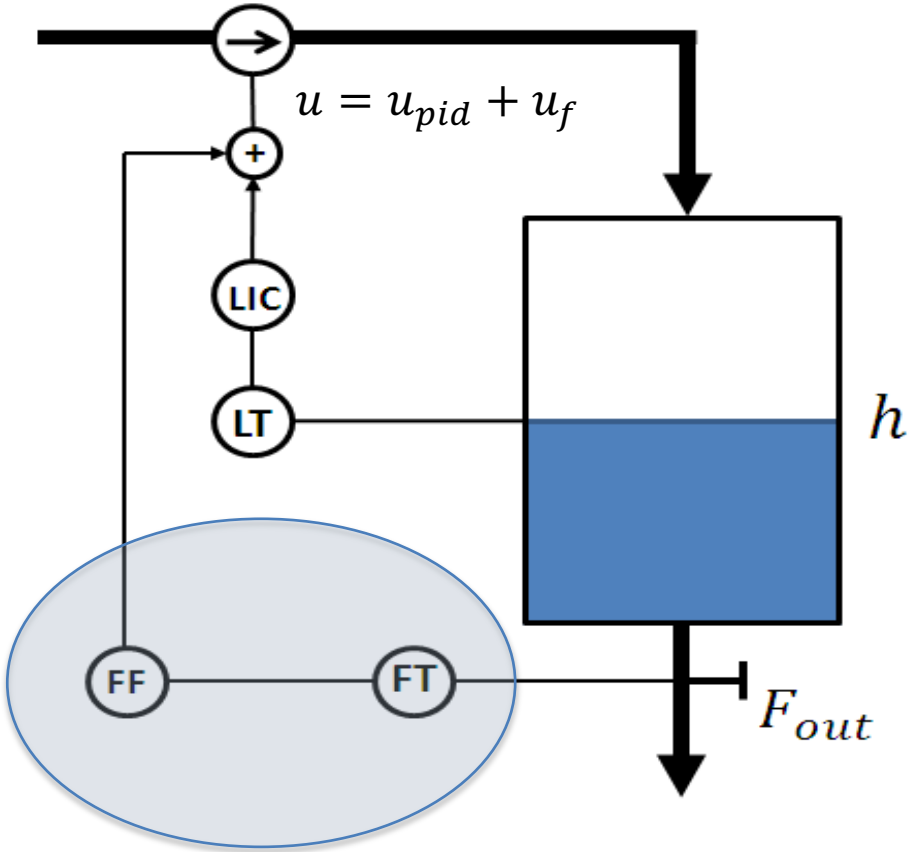
Hans-Petter Halvorsen

Feedforward Control

In this system is F_{out} a disturbance that we want to remove by using Feedforward

In our Real Level Tank System, only the Level is measured.

We will use Kalman Filter to find Estimates for the outflow (F_{out})



Feedforward Control

- In this system is F_{out} a noise signal/disturbance that we want to remove by using Feedforward.
- We want to design the Feedforward controller so that F_{out} is eliminated.
  - Solve for the control variable u, and substituting the process output variable h by its setpoint h_{sp}.
  - F_{out} is not measured, so you need to use the estimated value instead. Assume that the setpoint is constant.

We will use Feedforward Control in order to improve the control, compared to ordinary Feedback Control.

Design Feedforward Control for Level Tank

$$\dot{h} = \frac{1}{A_t}[K_p u - F_{out}]$$

We want to design the Feedforward controller so that F_{out} is eliminated.

We solve for the control variable u, and substituting the process output variable h by its setpoint h_{sp}. This gives the following feedforward controller u_f:

$$u_f = \frac{A\dot{h}_{sp}}{K_p} + \frac{F_{out}}{K_p}$$

Note! K_p is here the Pump Gain – NOT K_p in the PID Controller

We assume that the setpoint is constant, i.e. $\dot{h}_{sp} = 0$. This gives:

$$u_f = \frac{F_{out}}{K_p}$$

F_{out} is not measured, so we have to use the estimated value instead. This gives the following Feedforward controller:

$$\boldsymbol{u_f = \frac{F_{out,est}}{K_p}}$$

Note! Without feedforward control the control signal range of the PID controller is normally $[0, 5]$. With feedforward the output signal can be set to have the range $[-5, +5]$, so the contribution u_{pid} from the PID controller can be negative. If u_{pid} cannot be negative, the total signal $u = u_{pid} + u_f$ may not be small enough value to give proper control when the outflow is small.

https://www.halvorsen.blog

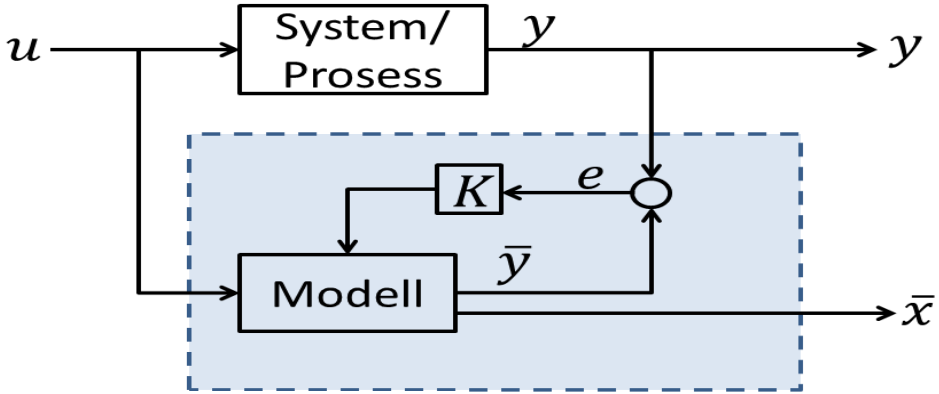
Kalman Filter

Hans-Petter Halvorsen

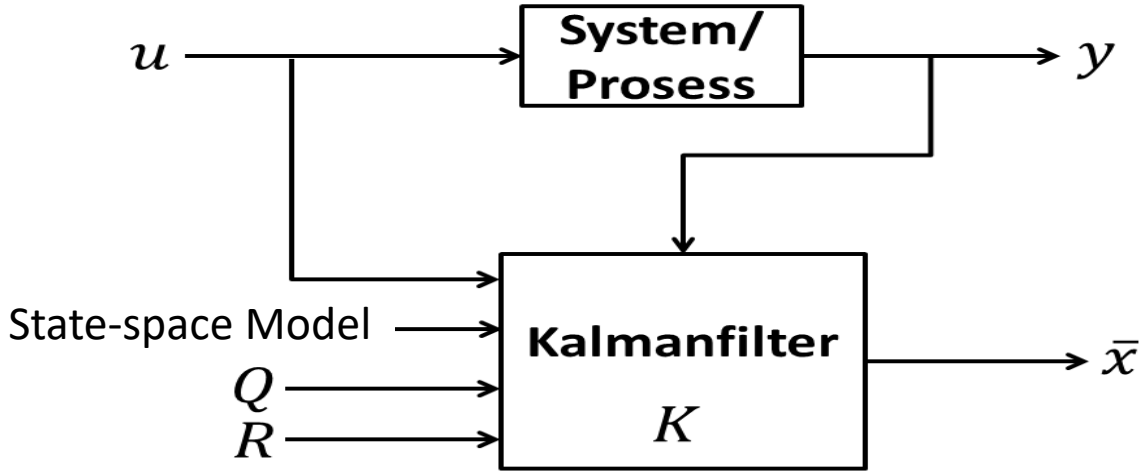
Kalman Filter

- The Kalman Filter is a commonly used method to estimate the values of state variables of a dynamic system that is excited by stochastic (random) disturbances and stochastic (random) measurement noise.
- We will estimate the process variable(s) using a Kalman Filter.
- We will use one of the built-in Kalman Filter algorithms in LabVIEW, but in addition we will create our own Kalman Filter algorithms from scratch.

Kalman Filter



Kalman Filter



Kalman Filter Algorithm

Pre Step: **Find the steady state Kalman Gain K**

K is time-varying, but you normally implement the steady state version of Kalman Gain K.

Init Step: **Set the initial Apriori (Predicted) state estimate**

$$\bar{x}_0 = x_0$$

Step 1: **Find Measurement model update**

$$\bar{y}_k = g(\bar{x}_k, u_k)$$

For Linear State-space model:

$$\bar{y}_k = C\bar{x}_k + Du_k$$

Step 2: **Find the Estimator Error**

$$e_k = y_k - \bar{y}_k$$

Step 3: **Find the Aposteriori (Corrected) state estimate**

$$\hat{x}_k = \bar{x}_k + Ke_k$$

Where K is the Kalman Filter Gain. Use the steady state Kalman Gain or calculate the time-varying Kalman Gain.

Step 4: **Find the Apriori (Predicted) state estimate update**

$$\bar{x}_{k+1} = f(\hat{x}_k, u_k)$$

For Linear State-space model:

$$\bar{x}_{k+1} = A\hat{x}_k + Bu_k$$

Step 1-4 goes inside a loop in your program.

Testing the Kalman Filter

- As with every model-based algorithm you should test your Kalman Filter with a simulated process before applying it to the real system.
- You can implement a simulator in LabVIEW since you already have a model (the Kalman Filter is model-based).
- In the testing, you can start with testing the Kalman Filter with the model in the simulator (without noise).
- Then you can introduce some noise in your simulator.
- You could also introduce some reasonable model errors by making the simulator model somewhat different from the Kalman Filter model, and check if the Kalman Filter still produces usable estimates.

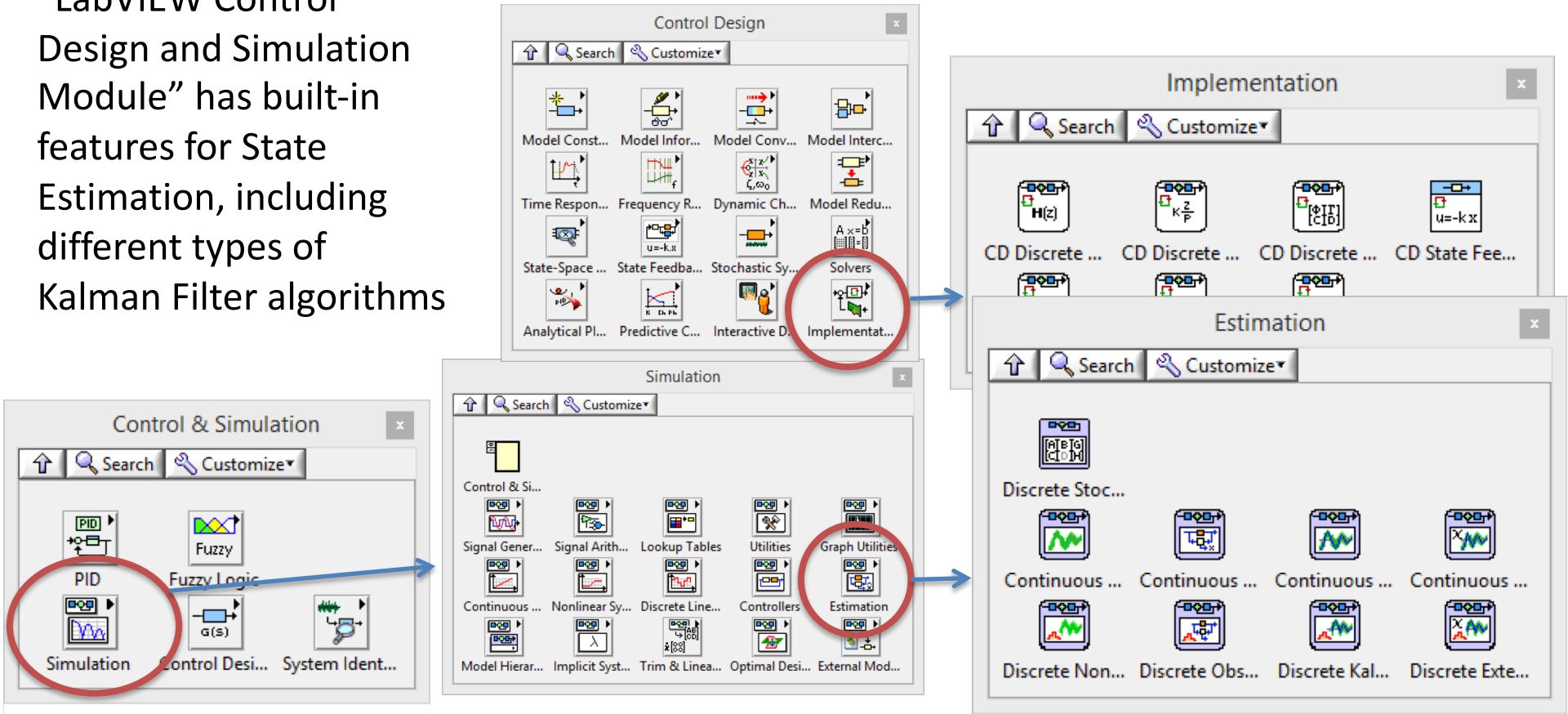
https://www.halvorsen.blog

Estimation and Kalman Filters in LabVIEW

Hans-Petter Halvorsen

State Estimation in LabVIEW

"LabVIEW Control Design and Simulation Module" has built-in features for State Estimation, including different types of Kalman Filter algorithms



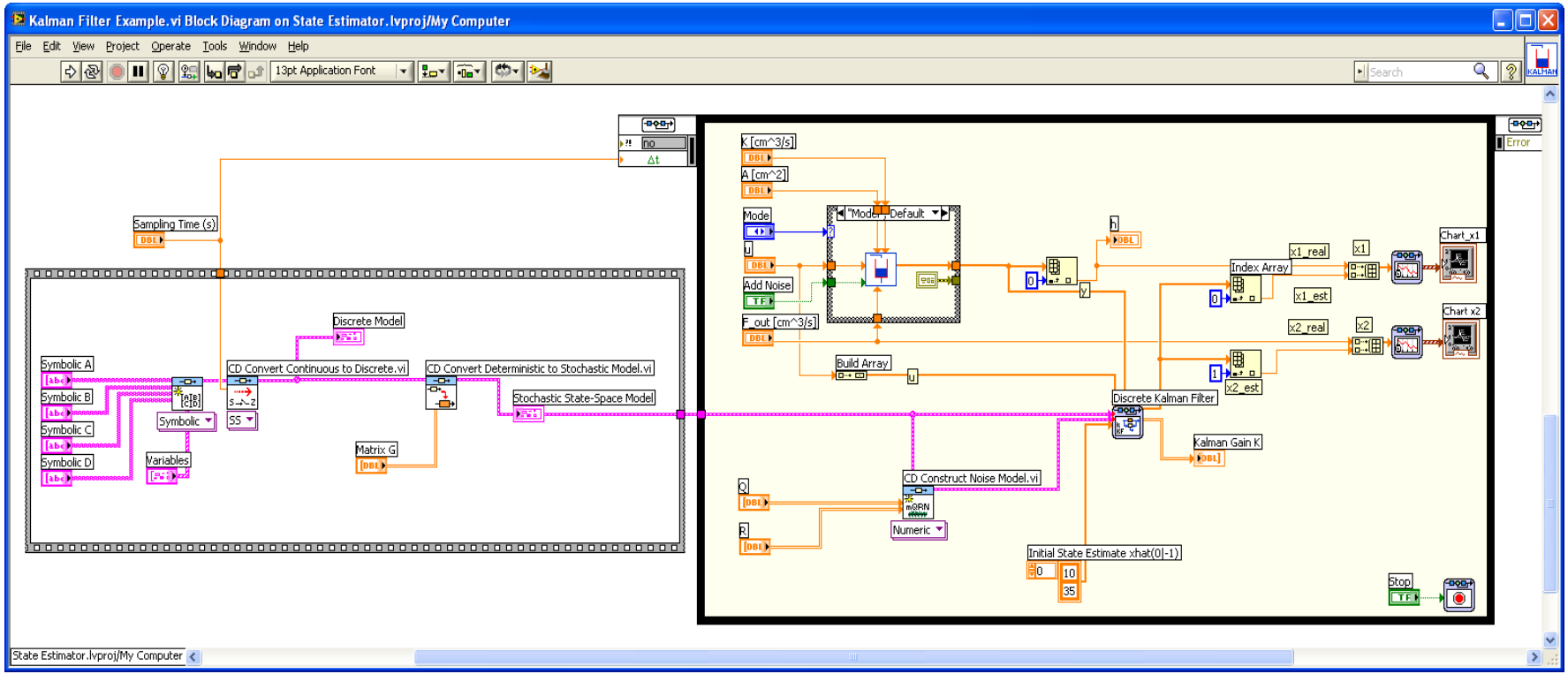
https://www.halvorsen.blog

Kalman Filter
LabVIEW Application

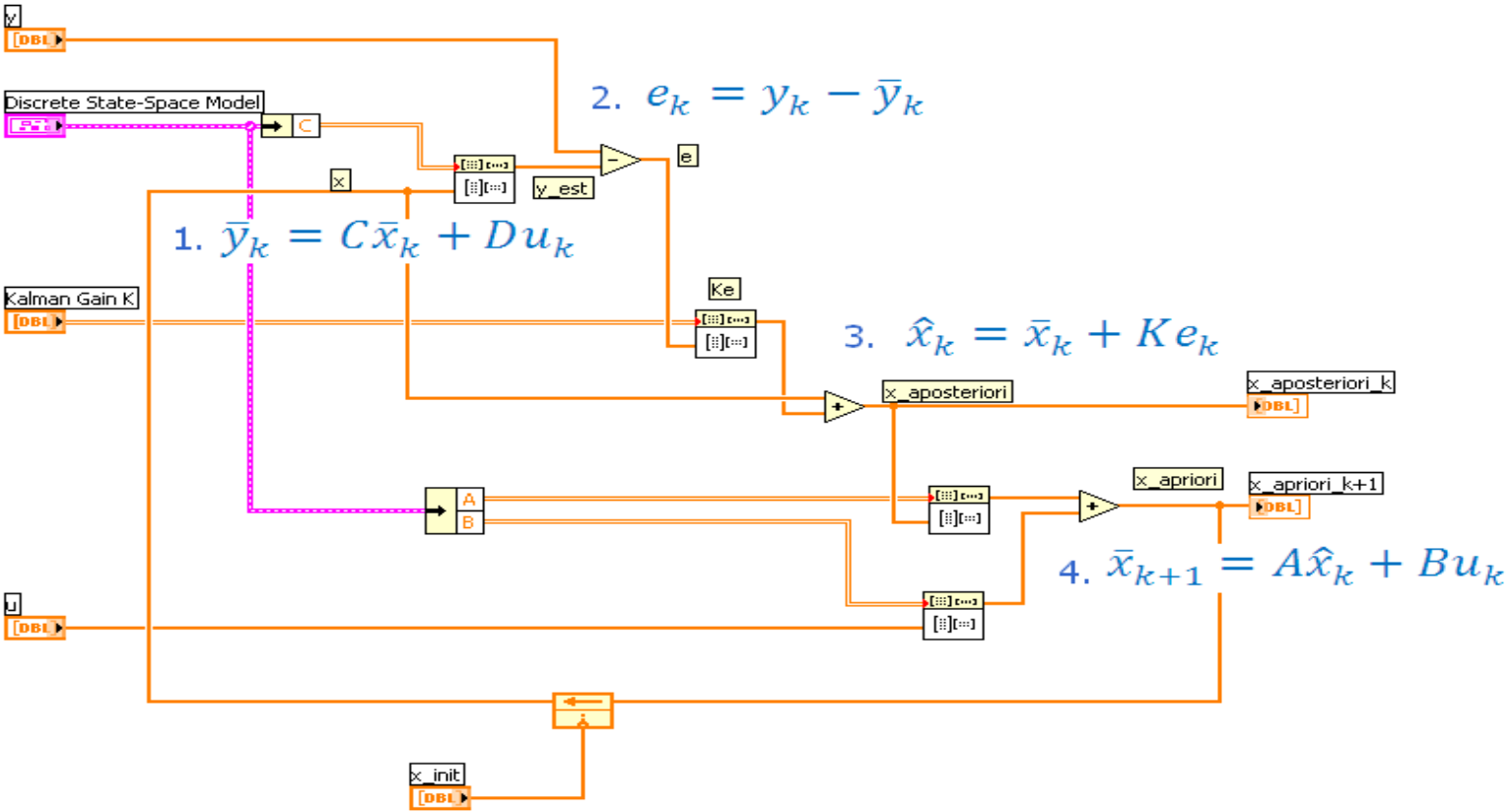
Feedforward Control

Hans-Petter Halvorsen

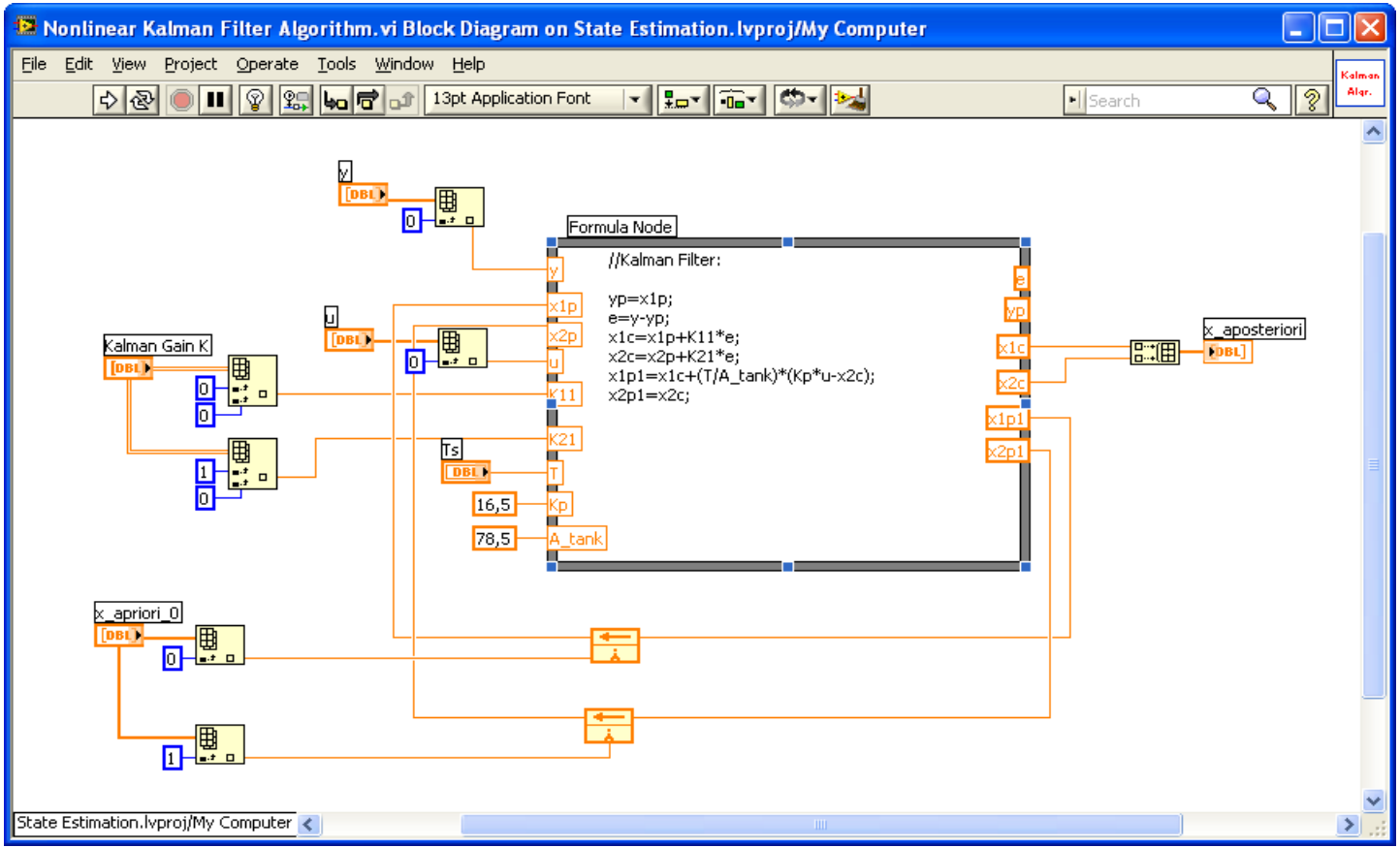
Built-in Kalman Filter in LabVIEW

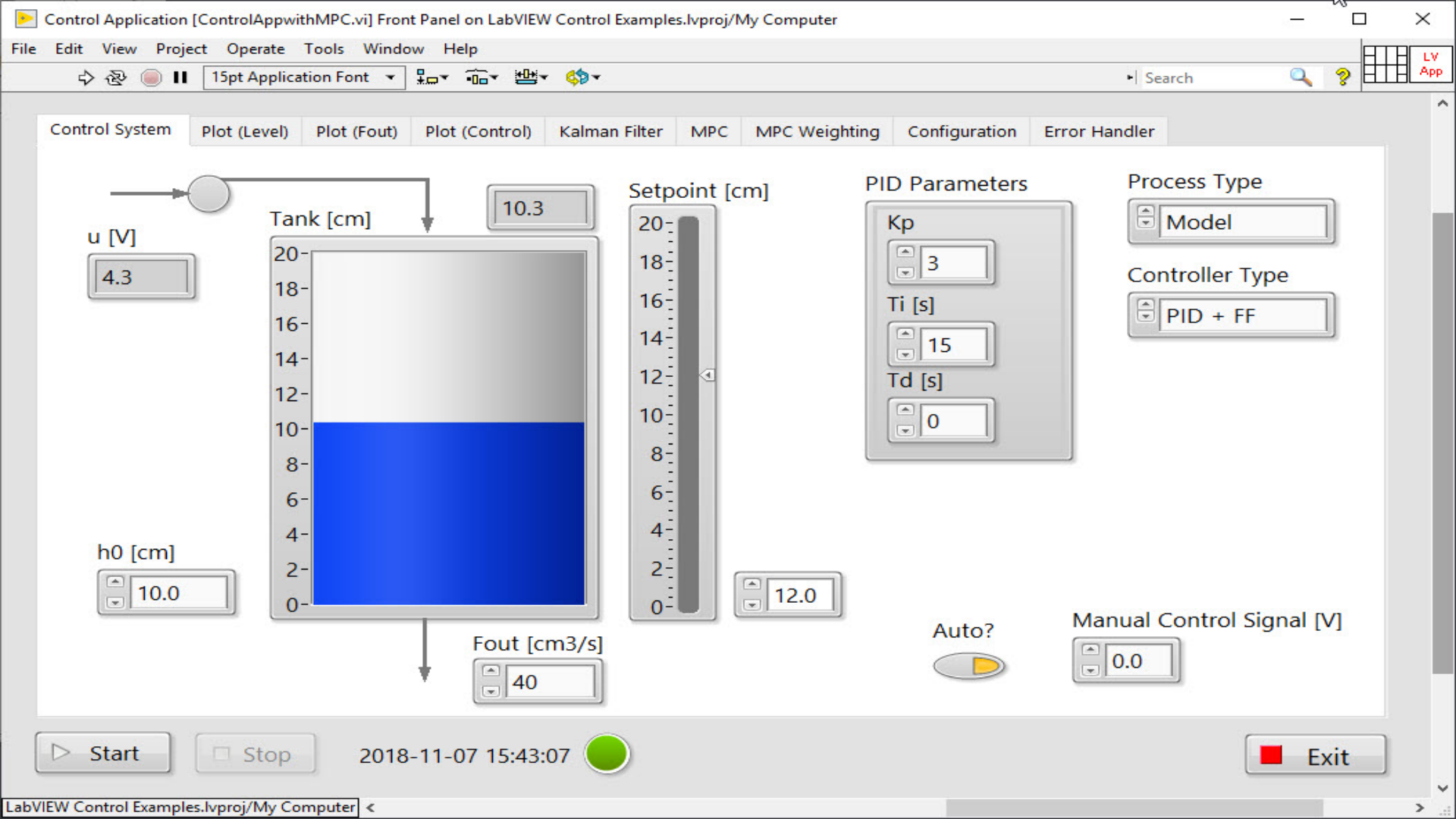


Linear Kalman Filter Implementation from scratch



Nonlinear Kalman Filter Implementation from scratch




Control Application [ControlAppwithMPC.vi] Front Panel on LabVIEW Control Examples.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
LV App
Control System
Plot (Level)
Plot (Fout)
Plot (Control)
Kalman Filter
MPC
MPC Weighting
Configuration
Error Handler
u [V]
4.3
Tank [cm]
10.3
Setpoint [cm]
PID Parameters
Kp
3
Ti [s]
15
Td [s]
0
Process Type
Model
Controller Type
PID + FF
h0 [cm]
10.0
12.0
Fout [cm3/s]
40
Auto?
Manual Control Signal [V]
0.0
Start
Stop
2018-11-07 15:43:07
Exit
LabVIEW Control Examples.lvproj/My Computer

Demo

https://www.halvorsen.blog

https://www.halvorsen.blog

MPC

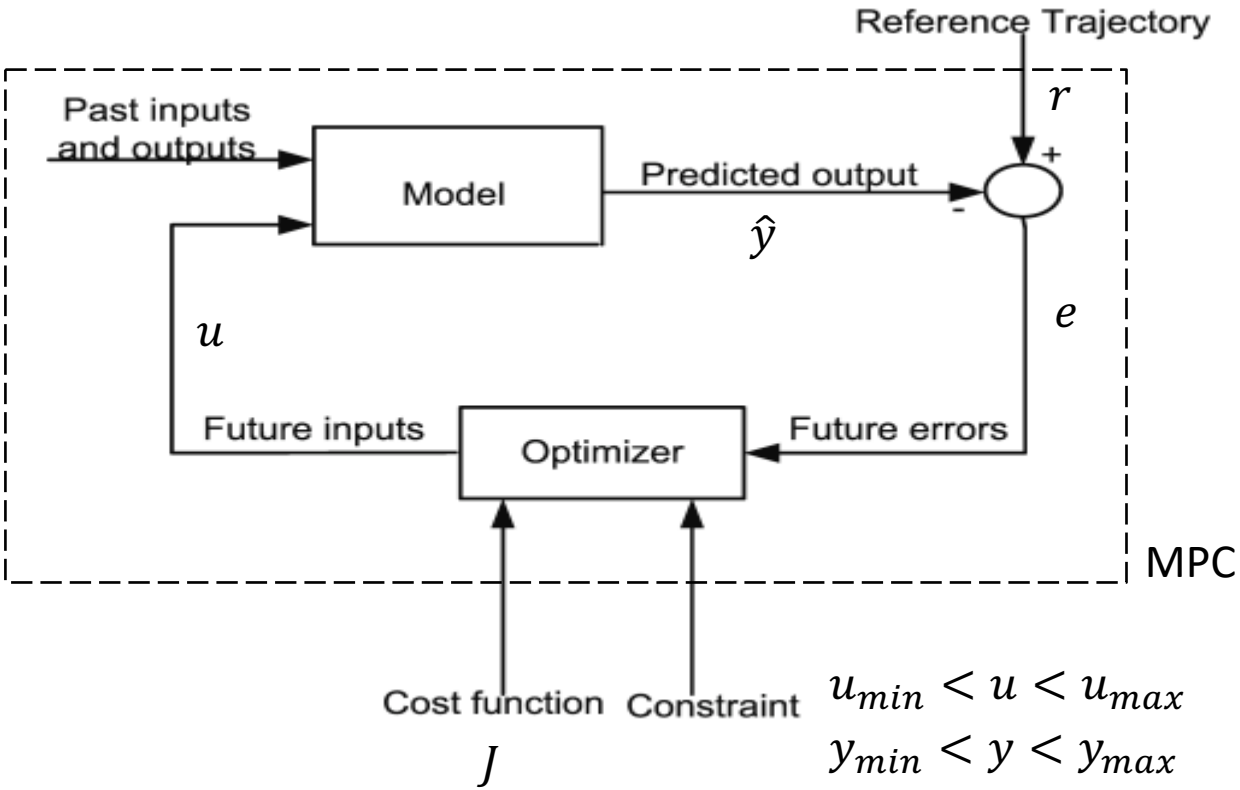
Hans-Petter Halvorsen

Model Predictive Control (MPC)

- Model predictive control (MPC) is an advanced method of process control that has been in use in the process industries since the 1980s.
- Model Predictive Control (MPC) is a multivariable control algorithm.
- Model predictive controllers rely on dynamic models of the process, most often linear empirical models obtained by system identification.
- MPC is based on iterative, finite-horizon optimization of a plant model.
- This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot. MPC has the ability to anticipate future events and can take control actions accordingly.

https://en.wikipedia.org/wiki/Model_predictive_control

Model Predictive Control (MPC)



[Wikipedia]

MPC Controller

MPC consists of:

- A **Model** of the process
  - Typically a State-space model, e.g.:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

- A **Cost function**, e.g.:

$$J = \sum_{k=0}^{N_p} (\hat{y} - r)^T Q\,(\hat{y} - r) + \sum_{k=0}^{N_c} \Delta u^T R\,\Delta u$$

- **Constraints**, e.g.:

$$u_{min} \leq u \leq u_{max}$$
$$y_{min} \leq y \leq y_{max}$$

MPC Cost Function

The cost function often used in MPC is like this (a **Linear Quadratic (LQ)** function):

$$J = \sum_{k=0}^{N_p} (\hat{y} - r)^T Q\,(\hat{y} - r) + \sum_{k=0}^{N_c} \Delta u^T R\,\Delta u$$

Where:

N_p – Prediction horizon, N_c – Control horizon

r – Set-point

$\hat{y}$ – Predicted process output

Δu – Predicted change in control value, $\Delta u_k = u_k - u_{k-1}$

Q – Output error weight matrix

R – Control weight matrix

So the basic problem is to solve:

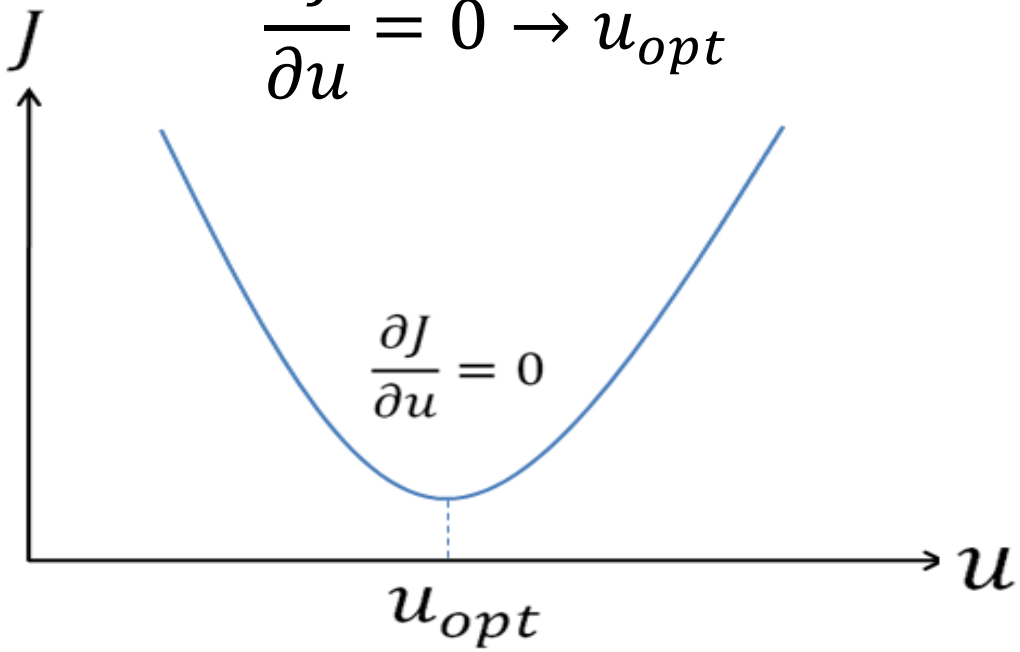
$$\frac{\partial J}{\partial u} = 0 \rightarrow u_{opt}$$

[National Instruments, LabVIEW Control Design user Manual, 2008. Available: http://www.ni.com/pdf/manuals/371057f.pdf]

The Cost Function – Optimization

So the basic problem is to solve:

$$\frac{\partial J}{\partial u} = 0 \rightarrow u_{opt}$$



By solving this we get the future optimal control.

Solving $\frac{\partial J}{\partial u} = 0$ is quite complex and will not be part of this tutorial, but in the figure below we see an illustration of the problem.

LabVIEW, MATLAB, etc. have built-in functions and algorithms which we will use.

Constraints

- All physical systems have constraints.
- We have physical constraints like actuator limits, etc. and we have safety constraints like temperature and pressure limits.
- Finally we have performance constraints like overshoot, etc.

In MPC you normally define these constraints:
Constraints in the outputs:

$$y_{min} \leq y \leq y_{max}$$

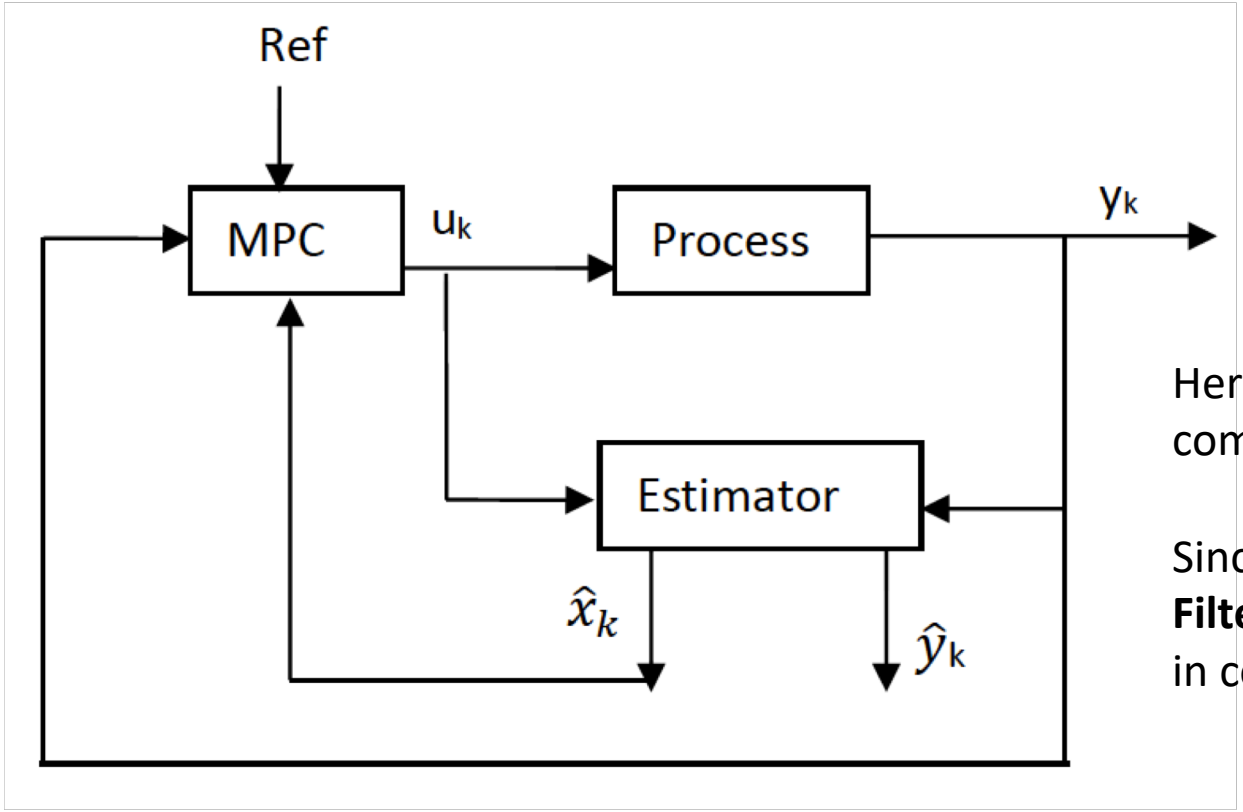
Constraints in the inputs:

$$\Delta u_{min} \leq \Delta u \leq \Delta u_{max}$$
$$u_{min} \leq u \leq u_{max}$$

Note! $\Delta u_k = u_k - u_{k-1}$

The MPC controller takes all these constraints into consideration when calculating the future controls.

Model Predictive Control (MPC)



Here you see MPC used in combination with an Estimator.

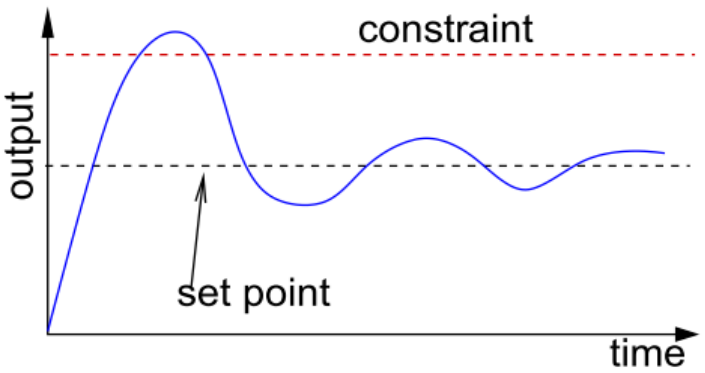
Since we already use a **Kalman Filter** Estimator, we can use that in combination with MPC

PID vs. MPC

- MPC is often used in addition to traditional control like PID – not as a replacement.
- In large plants MPC is not a replacement for traditional PID, but used in addition to PID controllers.
- PID controllers are used as single-loop controllers, while MPC is used as an overall system.
- PID handles only a single input and a single output (SISO systems), while MPC is a more advanced method of process control used for MIMO systems (Multiple Inputs, multiple Outputs).

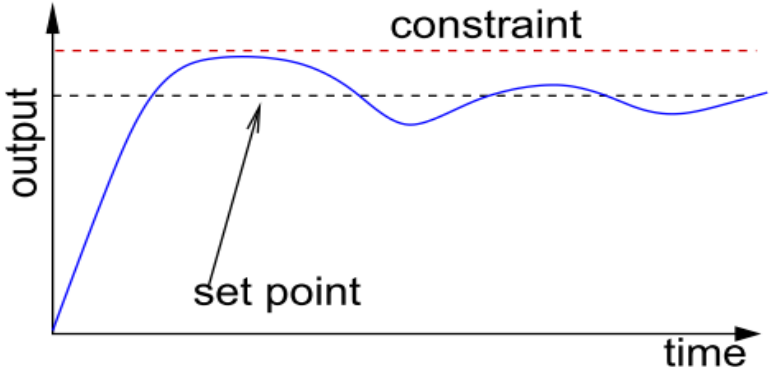
PID vs. MPC

Traditional Control (PID)



- No knowledge about constraints
- Set-point far from constraints
- Not optimal process operation
- SISO systems
- A mathematical model is not needed

MPC



- Constraints included in the design
- Set-point can be closer to constraints
- Improved process operation
- MIMO systems
- A mathematical model is needed

Another advantage of MPC is that cross coupling in multiple input and multiple output (MIMO) systems are taken into consideration in an optimal way. MPC is a simple method for controlling MIMO systems.

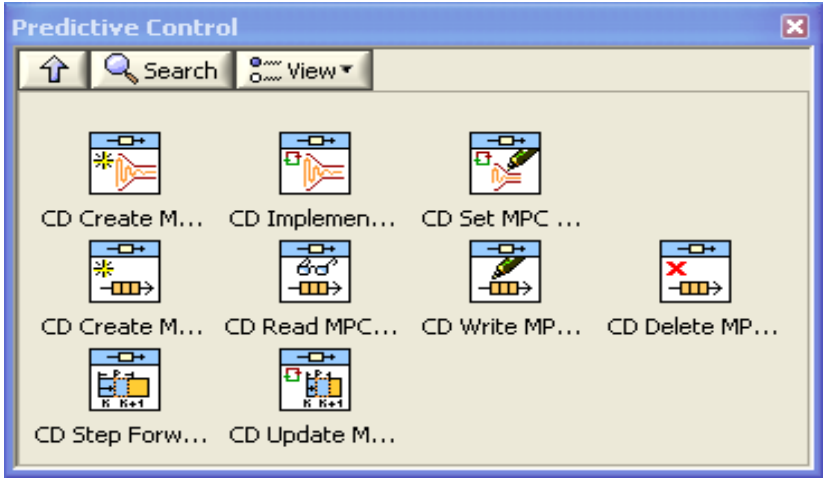
https://www.halvorsen.blog

MPC in LabVIEW

Hans-Petter Halvorsen

MPC in LabVIEW

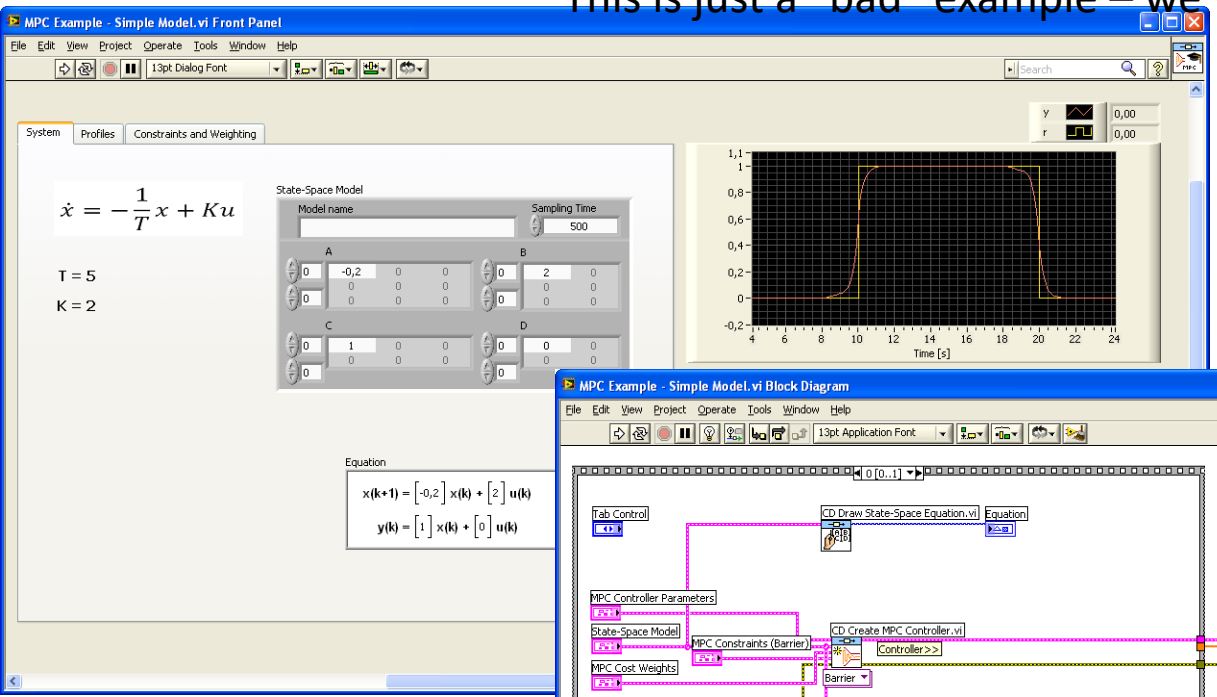
In LabVIEW you have the following Predictive Control palette:

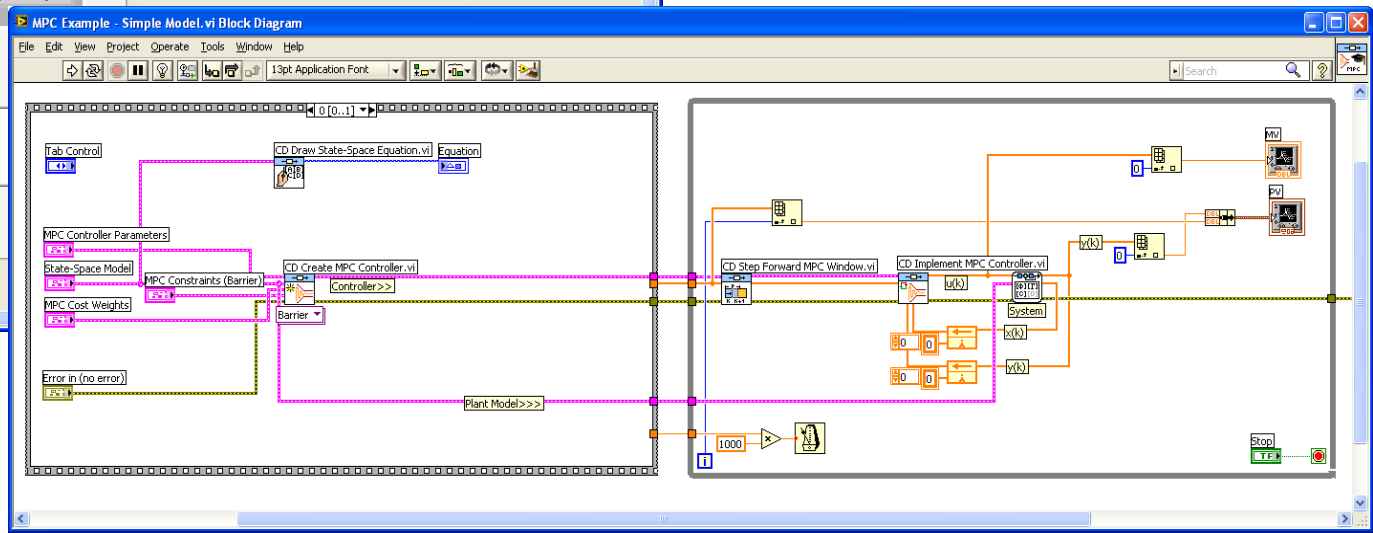


- You use the "**CD Create MPC Controller**" VI to create an MPC controller. This VI bases the MPC controller on a state-space model of the plant that you provide.
- The "**CD Implement MPC Controller**" is used to calculate the control values for each sampling time and is normally implemented in a loop, e.g., a While Loop.

MPC Example in LabVIEW

This is just a “bad” example – we will create a better application



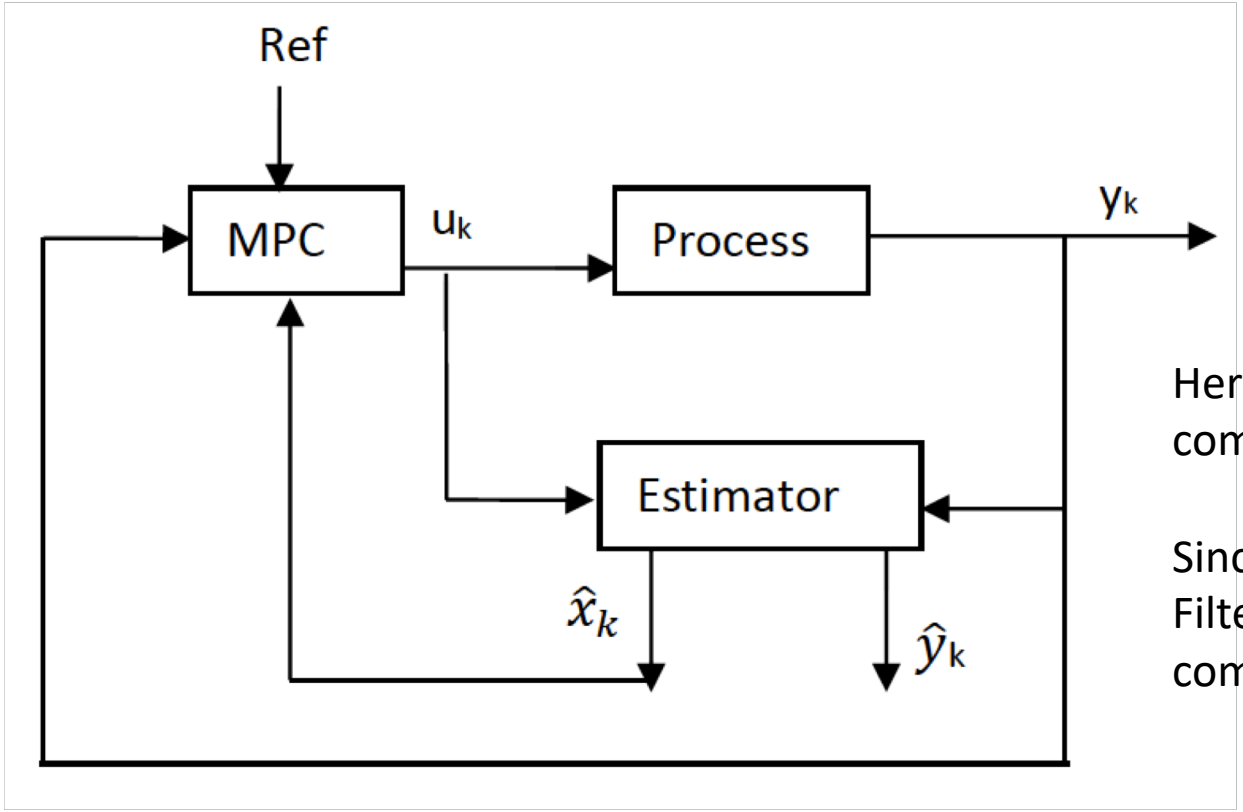


https://www.halvorsen.blog

MPC
LabVIEW Application

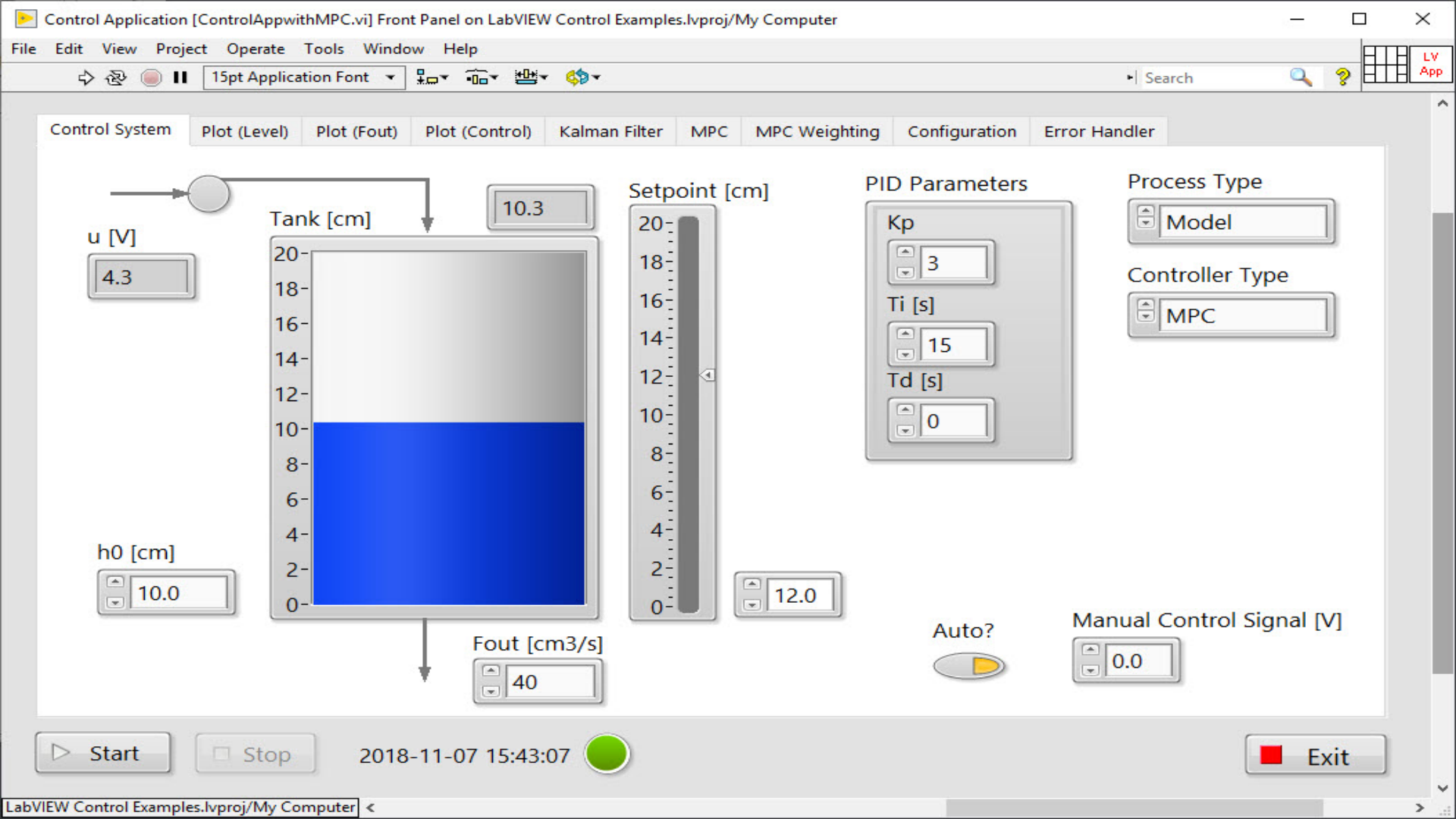
Hans-Petter Halvorsen

Model Predictive Control (MPC)



Here you see MPC used in combination with an Estimator.

Since we already use a Kalman Filter Estimator, we can use that in combination with MPC


Control Application [ControlAppwithMPC.vi] Front Panel on LabVIEW Control Examples.lvproj/My Computer
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
LV App
Control System
Plot (Level)
Plot (Fout)
Plot (Control)
Kalman Filter
MPC
MPC Weighting
Configuration
Error Handler
u [V]
4.3
Tank [cm]
10.3
Setpoint [cm]
PID Parameters
Kp
3
Ti [s]
15
Td [s]
0
Process Type
Model
Controller Type
MPC
h0 [cm]
10.0
12.0
Fout [cm3/s]
40
Auto?
Manual Control Signal [V]
0.0
Start
Stop
2018-11-07 15:43:07
Exit
LabVIEW Control Examples.lvproj/My Computer

Demo

https://www.halvorsen.blog

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog