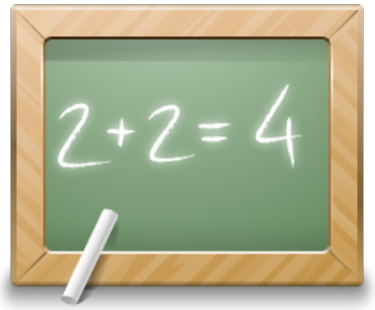
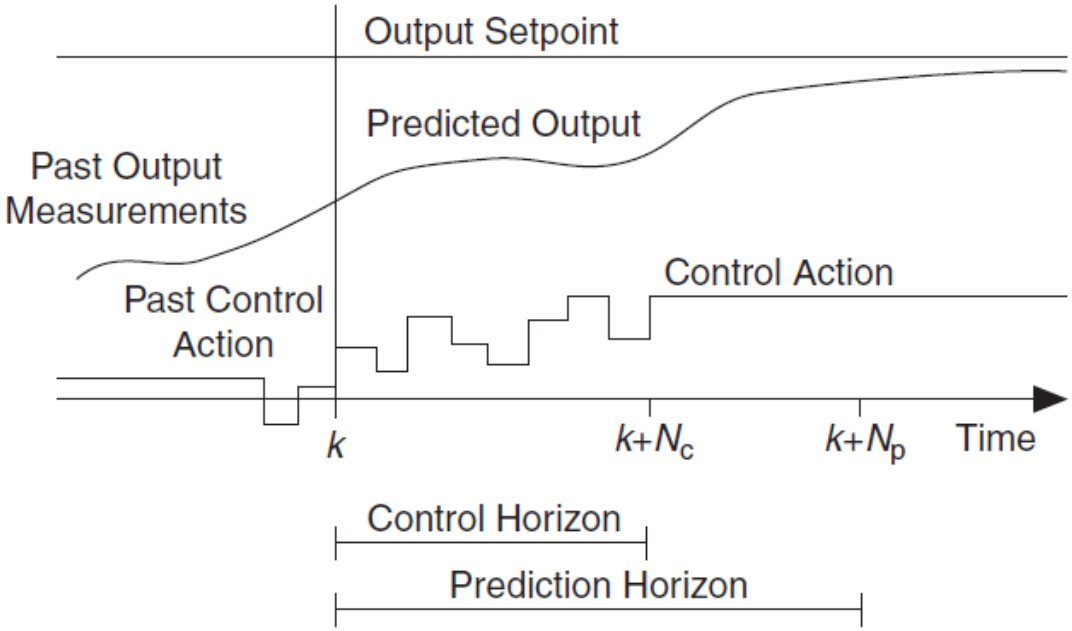
https://www.halvorsen.blog



Model Predictive Control in LabVIEW

Hans-Petter Halvorsen



https://www.halvorsen.blog/documents/automation/mpc/

Model Predictive Control in LabVIEW

Hans-Petter Halvorsen

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

Web: https://www.halvorsen.blog

https://www.halvorsen.blog

Preface

Model Predictive Control, or MPC, is an advanced method of process control that has been in use in the process industries such as chemical plants and oil refineries since the 1980s. Model predictive controllers rely on dynamic models of the process, most often linear empirical models obtained by system identification.

Model predictive control (MPC) refers to a class of computer control algorithms that utilize an explicit process model to predict the future response of a plant. At each control interval an MPC algorithm attempts to optimize future plant behavior by computing a sequence of future manipulated variable adjustments. The first input in the optimal sequence is then sent into the plant, and the entire calculation is repeated at subsequent control intervals. Originally developed to meet the specialized control needs of power plants and petroleum refineries, MPC technology can now be found in a wide variety of application areas including chemicals, food processing, automotive, and aerospace applications.

Programming tools like, e.g., MATLAB (Model Predictive Control Toolbox) and LabVIEW (Control Design and Simulation Module) has MPC functionality.

DeltaV, which is a DCS (Distributed Control System) system has MPC functionality (DeltaV Predict/ DeltaV Predict Pro).

These are just a few examples, but mentioned here because these tools and systems are availble at the university.

In this Tutorial we will use the Predictive Control functionality which is part of the LabVIEW Control Design and Simulation Module.

The scope with this Tutorial is not to go in depth of the theory behind MPC, but to use and give an overview of the MPC implementation in LabVIEW.

[Figure on title page: National Instruments, LabVIEW Control Design User Manual, 2008]

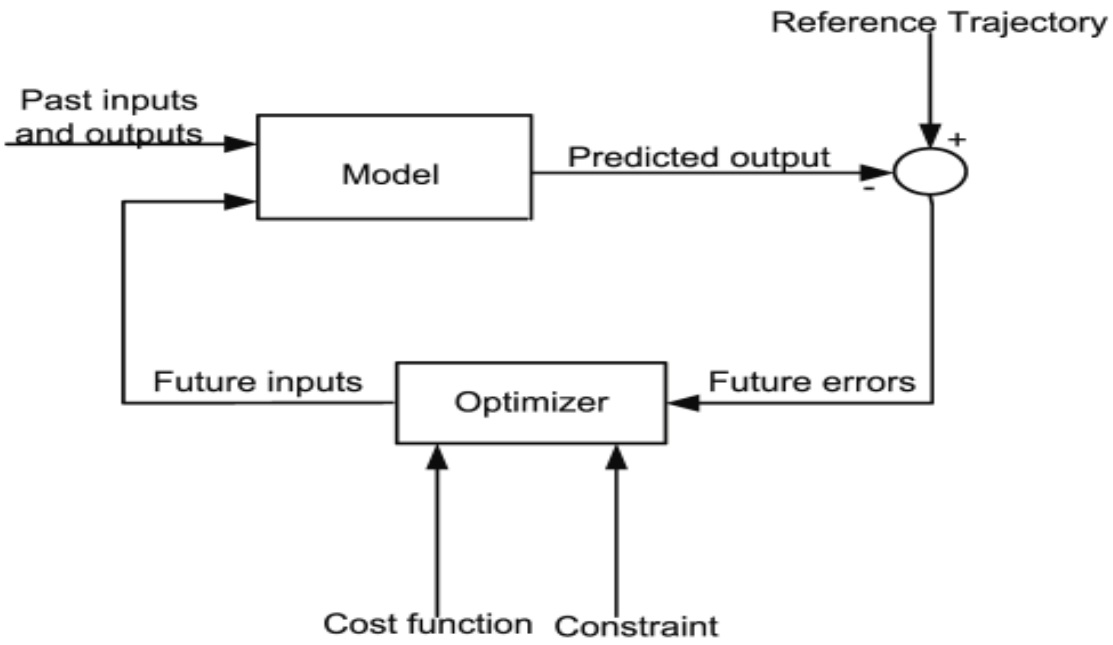
Table of Contents

1 Introduction to Model Predictive Control

1.1 Introduction

Traditional feedback controllers operate by adjusting control action in response to a change in the output setpoint of a system. Model predictive control (MPC) is a technique that focuses on constructing controllers that can adjust the control action before a change in the output setpoint actually occurs. This predictive ability, when combined with traditional feedback operation, enables a controller to make adjustments that are smoother and closer to the optimal control action values.

Below we see the basic structure of MPC:



[Wikipedia]

Model Predictive Control (MPC) is a control strategy which is a special case of the optimal control theory developed in the 1960 and lather. MPC consists of an optimization problem at each time instants, k.

The main point of this optimization problem is to compute a new control input vector, u_k, to be feed to the system, and at the same time take process constraints into consideration (e.g., constraints on process variables).

An MPC algorithm consists of:

- A Cost function
- Constraints
- A Model of the process

These things will be explained in detail below.

1.2 Prediction and Control Horizons

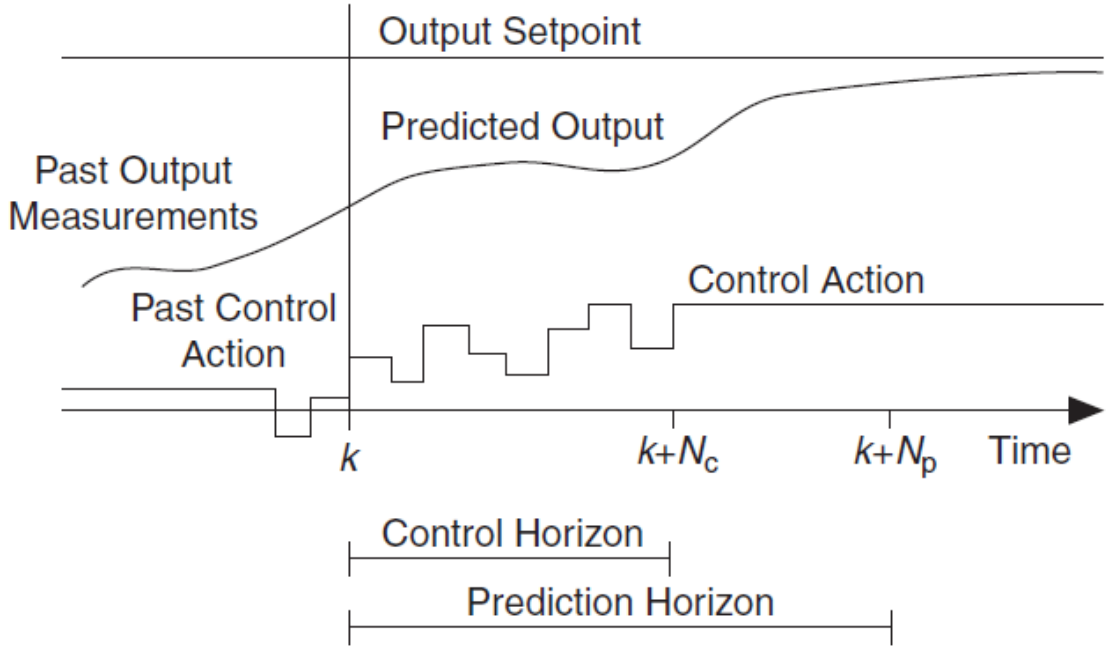
Prediction horizon (N_p) - The number of samples in the future the MPC controller predicts the plant output.

Control horizon (N_c) – The number of samples within the prediction horizon where the MPC controller can affect the control action.

Note!

$$N_c \leq N_p$$

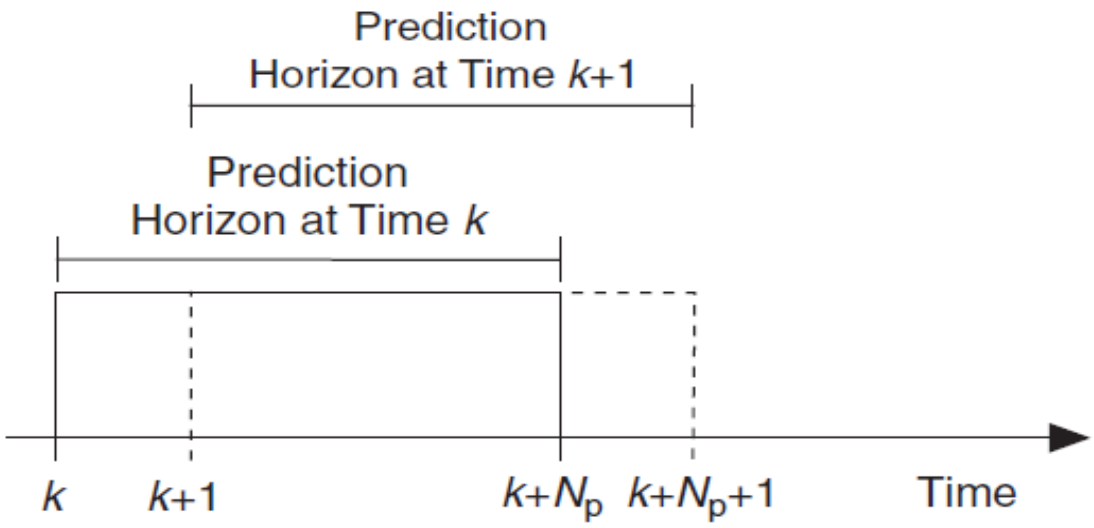
Below we see the Prediction and Control Horizons:



[Figure: National Instruments, LabVIEW Control Design User Manual, 2008]

For time k the MPC controller predicts the plant output for time $k + N_p$. We see from the figure that the control action does not change after the control horizon ends.

The first input in the optimal sequence is then sent into the plant, and the entire calculation is repeated at subsequent control intervals. For each iteration the prediction horizon is moving forward in time and the MPC controller again predicts the plant output.



[Figure: National Instruments, LabVIEW Control Design User Manual, 2008]

Prediction horizon:

A short prediction horizon reduces the length of time during which the MPC controller predicts the plant outputs. When the prediction horizon is short the MPC controller works more like a traditional feedback controller.

A long prediction horizon increases the predictive ability of the MPC controller, but the performance poorer due to extra calculations.

Control horizon:

A short control horizon means more carefully changes in the control action.

A long control horizon means more aggressive changes in the control action.

1.3 Model

The main drawback with MPC is that a model for the process, i.e., a model which describes the input to output behavior of the process, is needed. Mechanistic models derived from conservation laws can be used. Usually, however in practice simply data-driven linear models are used.

In MPC it is assumed that the model is a discrete state-space model of the form:

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

1.4 Cost Function

The main idea with MPC is that the MPC controller calculates a sequence of future control actions such that a cost function is minimized.

The cost function often used in MPC is like this (a linear quadratic function) [National Instruments, LabVIEW Control Design User Manual, 2008] :

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

Where:

N_p – Prediction horizon

r – Setpoint

$\hat{y}$ – 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

This works for MIMO systems (Multiple Input and Multiple Outputs) so we are dealing with vectors and matrices.

For a scalar system we have:

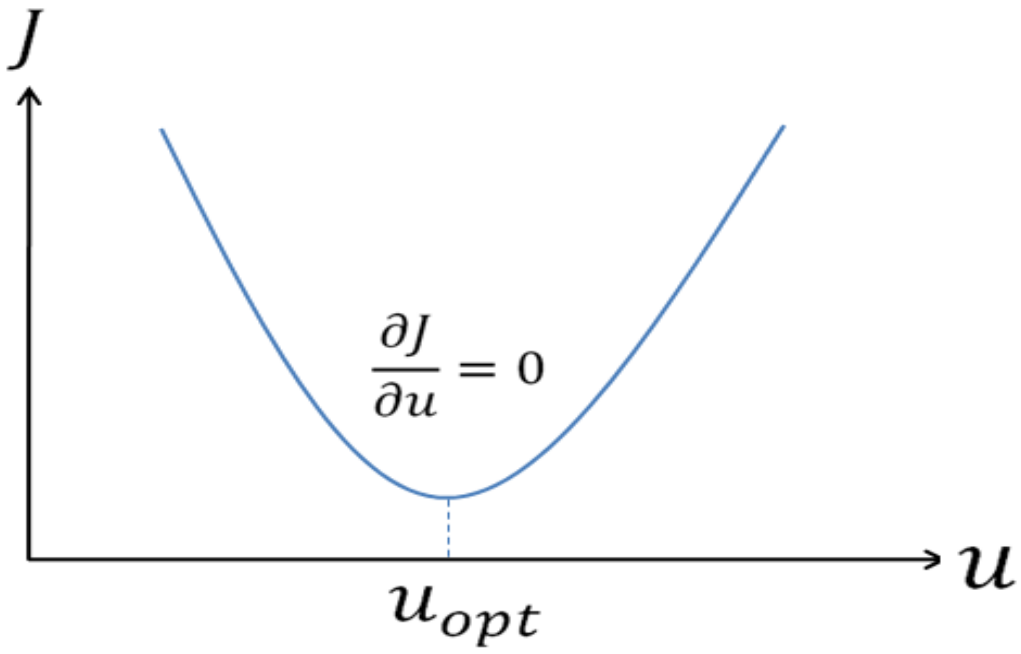
$$J = \sum_{k=0}^{N_p} q(\hat{y} - r)^2 + \sum_{k=0}^{N_p} r\Delta u^2$$

So the basic problem is to solve:

$$\frac{\partial J}{\partial u} = 0$$

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.



1.5 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:

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

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

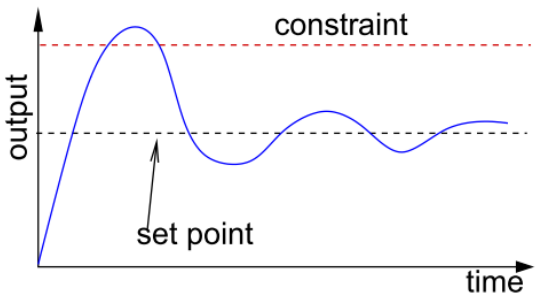
Note! $\Delta u_k = u_k - u_{k-1}$

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

1.6 MPC vs. Traditional Control (PID)

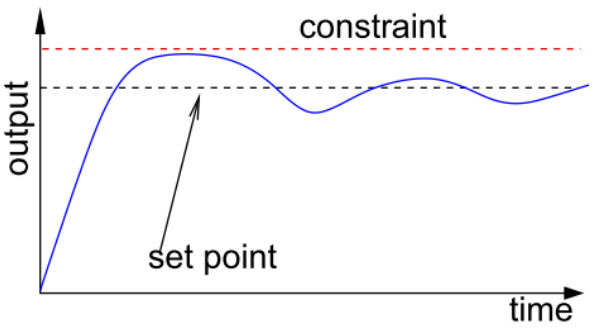
MPC is often used in addition to traditional control like PID. In large plants MPC is not a replacement for traditional PID, but used in addition to PID controllers. PID controllers are used as single-loop controllers, while MPC is used as an overall system. PID handles only a single input and a single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

Traditional Control (PID)



- No knowledge about constraints
- Setpoint far from constraints
- Not optimal process operation
- SISO systems

MPC



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

- A mathematical model is not needed

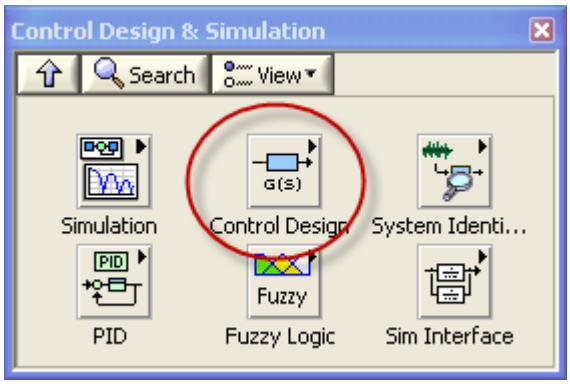
The models used in MPC are generally intended to represent the behavior of complex dynamical systems. The additional complexity of the MPC control algorithm is not generally needed to provide adequate control of simple systems, which are often controlled well by PID controllers. Common dynamic characteristics that are difficult for PID controllers include large time delays and high-order dynamics.

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.

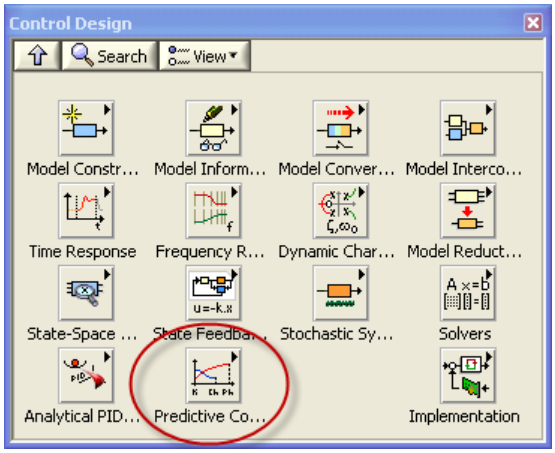
2 LabVIEW Control and Simulation Module

The MPC functionality in LabVIEW is part of the “Control Design and Simulation Module”.

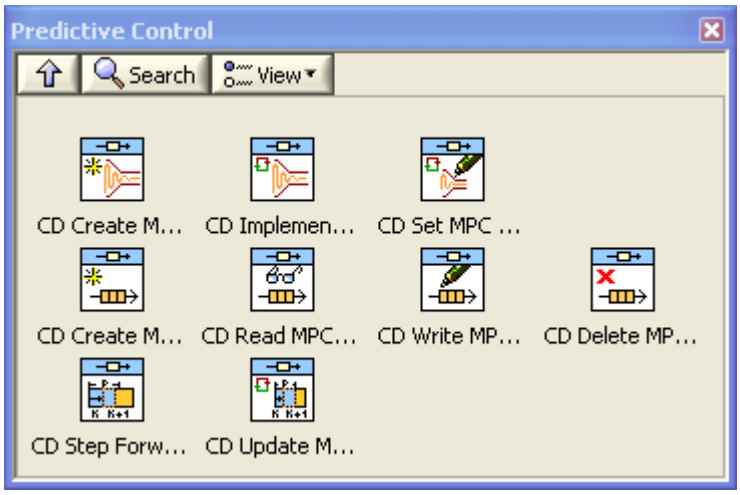
The “Control Design and Simulation” palette in LabVIEW:



The “Control Design” palette in LabVIEW:



The “Predictive Control” palette in LabVIEW:



Here is a short description of the different VIs:

Palette Object	Description
CD Create MPC Controller	Creates a model predictive control (MPC) controller for a state-space model. You must manually select the polymorphic instance to use.
CD Create MPC FIFO	Creates a queue or real-time (RT) FIFO for an MPC controller. You use this queue or RT FIFO to update setpoint and/or disturbance profiles dynamically.
CD Delete MPC FIFO	Deletes the **MPC FIFO**. After you delete this FIFO, the CD Write MPC FIFO VI stops writing data to the FIFO and the loop that contains this VI terminates.
CD Implement MPC Controller	Calculates the **Control Action u(k)** to apply to the plant. This VI uses the **Output Reference Window**, **Disturbance Window**, and **Control Action Reference Window** parameters to calculate the control action along the control horizon at time *k*.
CD Read MPC FIFO	Reads a portion, or window, of profile values from the **MPC FIFO**.
CD Set MPC Controller	Updates specified parameters of a model predictive control (MPC) controller for a state-space model. You must manually select the polymorphic instance to use.
CD Step Forward MPC Window	Calculates the appropriate portion, or window, of the setpoint and/or disturbance profiles. You wire these windows to the appropriate input(s) of the CD Implement MPC Controller VI.
CD Update MPC Window	Calculates the appropriate portion, or window, of the setpoint or disturbance profile of a signal from time *k* to time *k* + **Prediction Horizon**. You wire these windows to the appropriate input(s) of the CD Implement MPC Controller VI.
CD Write MPC FIFO	Writes a control action setpoint, output setpoint, or disturbance profile window to the **MPC FIFO**. You then use the CD Read MPC FIFO VI to read values from this MPC FIFO.

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.

3 MPC in LabVIEW

In this chapter we will use the Vis in the “Predictive Control” palette in in some example.

3.1 Example 1: Simple 1. order Model

Given the following system:

$$\dot{x} = -\frac{1}{T}x + Ku$$

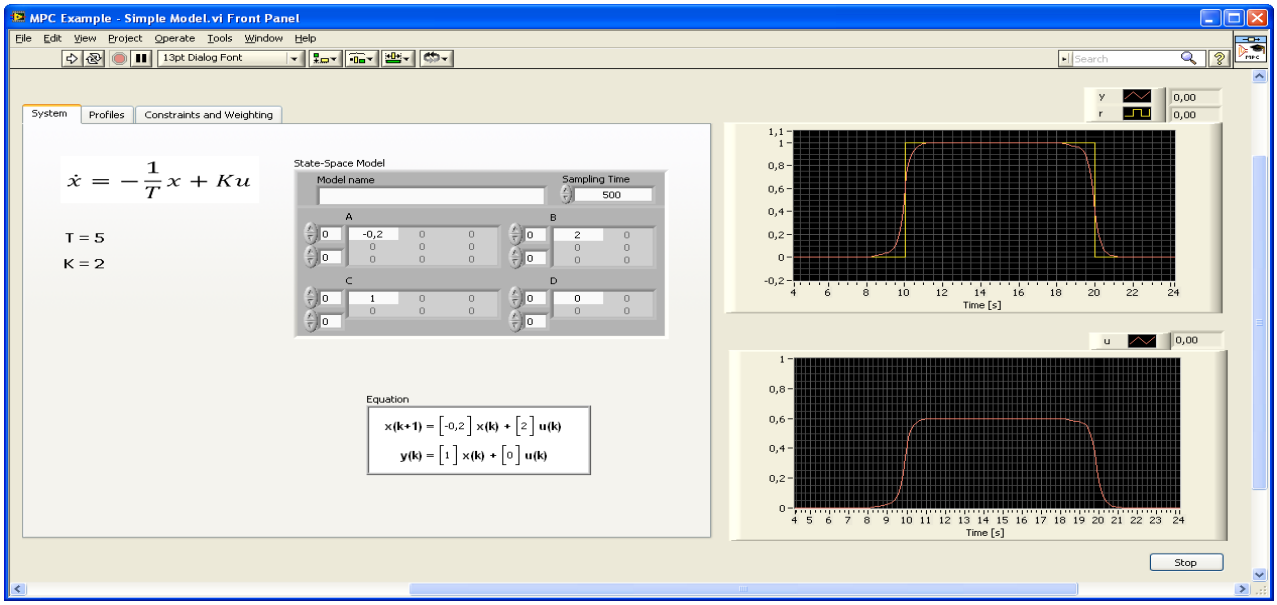
Where

T is the time constant for the system

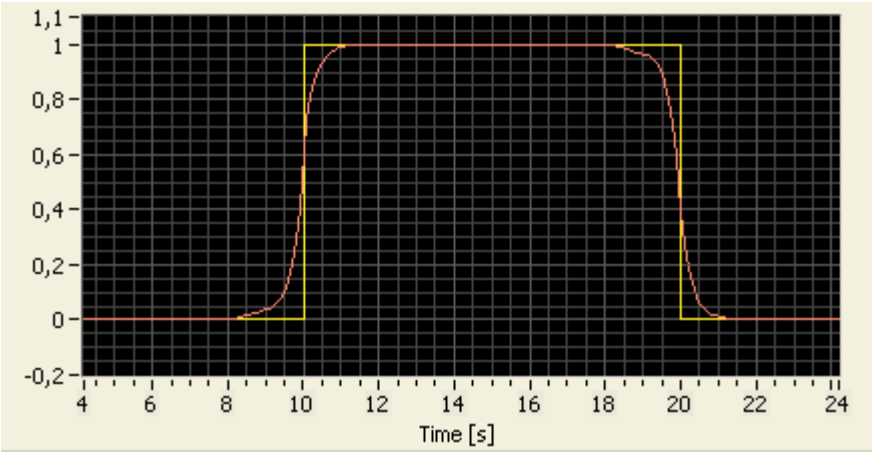
K is, e.g., the pump gain

We set $T = 5s$ and $K = 2$

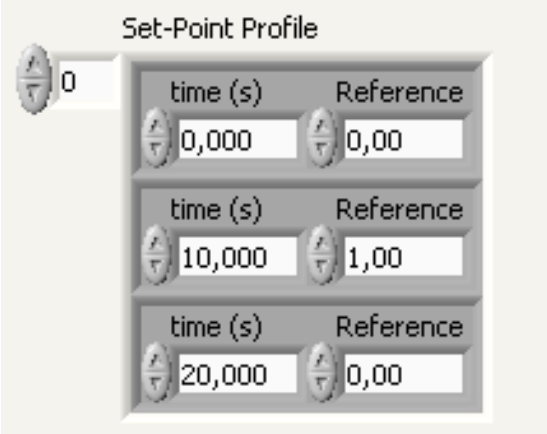
Front Panel:

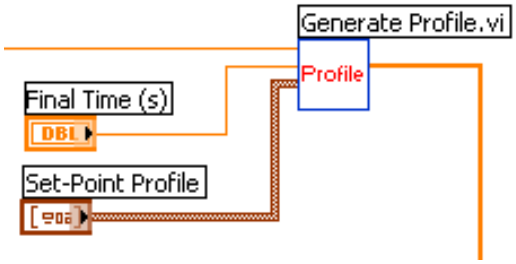


Results:

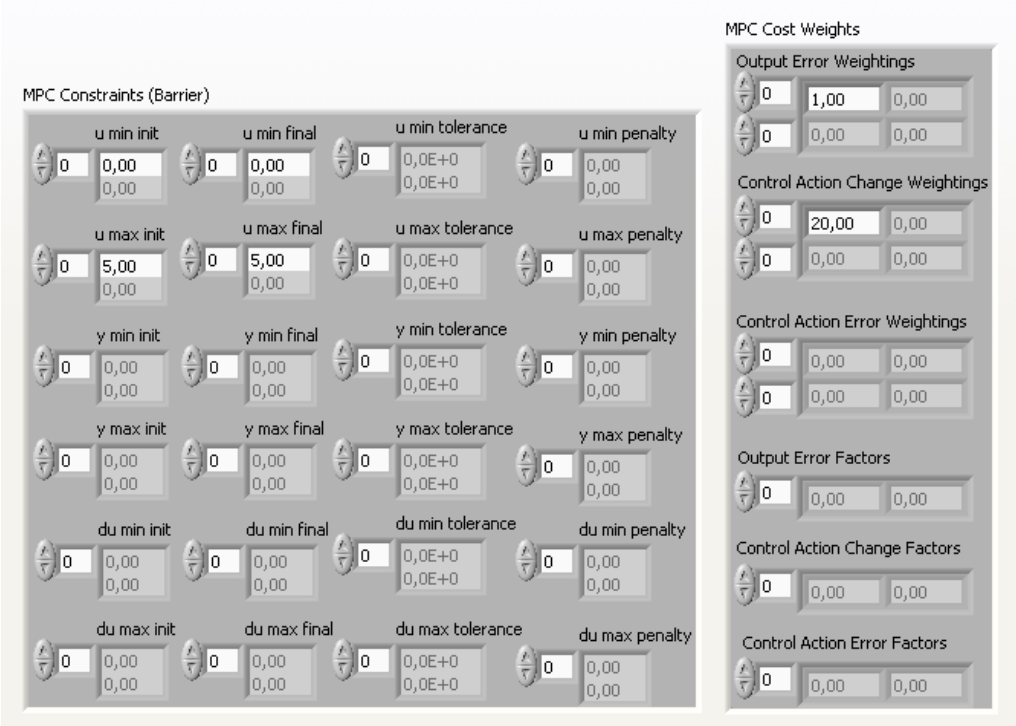


Setpoint Profile:

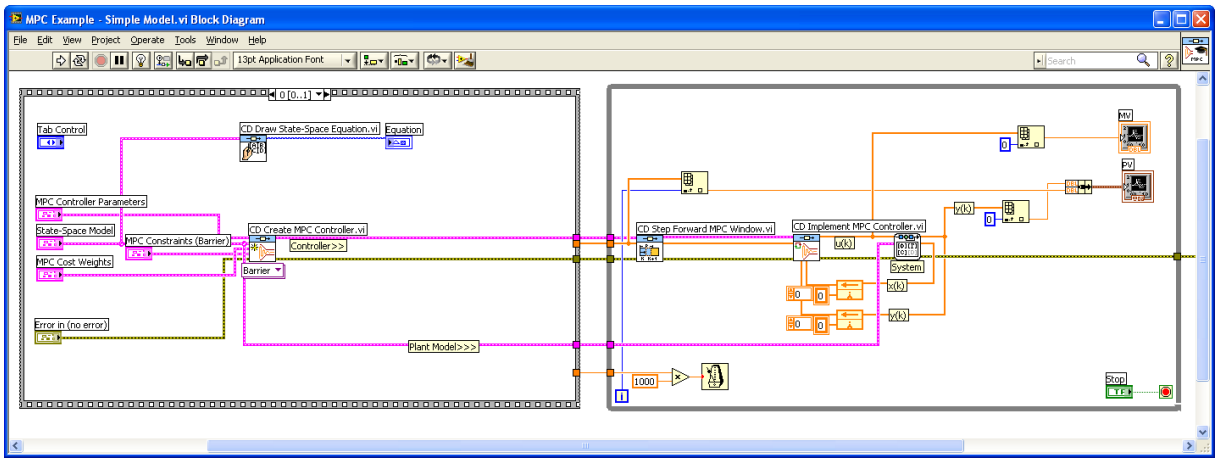




Constraints and Weighting:



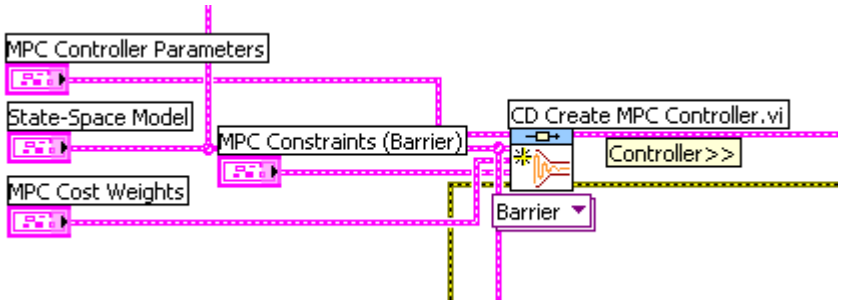
Block Diagram:



We can divide the solution into 2 different parts:

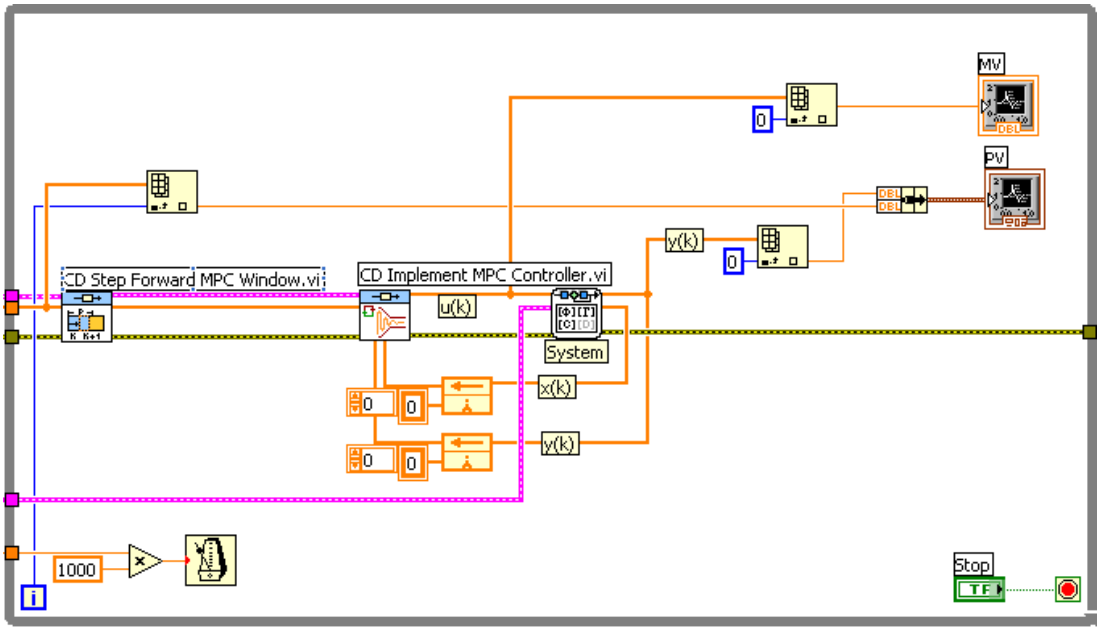
Initialization the MPC Controller:

This is something we do only once when we start the program. We use the “CD Create MPC Controller.vi”.



Run the Controller:

This operation is performed each sample, and this is normally executed inside a loop, e.g. a While Loop. We use the “CD Implement MPC Controller.vi”.



[End of Example]

3.2 Example 2: Model with Time Delay

Given the following system:

$$\dot{x} = -\frac{1}{T}x + Ku(t-\tau)$$

Where

T is the time constant for the system

K is, e.g., the pump gain

τ is the time delay

We set $T = 5s,\ K = 2$ and $\tau = 3s$

The MPC algorithm requires that the model is a linear state-space model, but the time delay causes problems.

A solution could be to transform the differential equation we have to a transfer function. Then we can use built-in functions in LabVIEW to convert it to a linear state-space model.

We use Laplace on the differential equation above:

$$sx(s) = -\frac{1}{T}x(s) + Ku(s)e^{-\tau s}$$

Note! We use the following Laplace transformation:

$$F(s)e^{-\tau s} \Leftrightarrow f(t-\tau)$$

$$sF(s) \Leftrightarrow \dot{f}(t)$$

This gives:

$$sx(s) + \frac{1}{T}x(s) = Ku(s)e^{-\tau s}$$

Next:

$$x(s)\left(s + \frac{1}{T}\right) = Ku(s)e^{-\tau s}$$

Next:

$$\frac{x(s)}{u(s)} = \frac{K}{s + \frac{1}{T}}e^{-\tau s}$$

Finally:

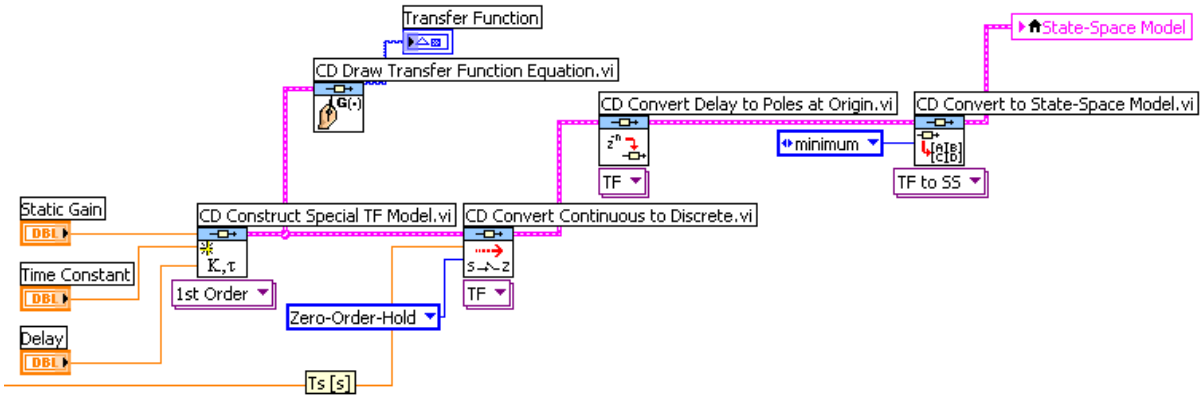
$$H(s) = \frac{x(s)}{u(s)} = \frac{KT}{Ts+1}e^{-\tau s} = \frac{K_{tot}}{Ts+1}e^{-\tau s}$$

With values ($T = 5,\ K = 2,\ \tau = 3$):

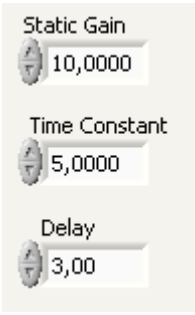
$$H(s) = \frac{x(s)}{u(s)} = \frac{10}{5s+1} e^{-3s}$$

LabVIEW application:

We can use the “CD Construct Special TF Model.vi” in order to create the transfer function. Then we use miscellaneous functions in order to end up with a discrete state-space model that handles the time delay (additional states are added).

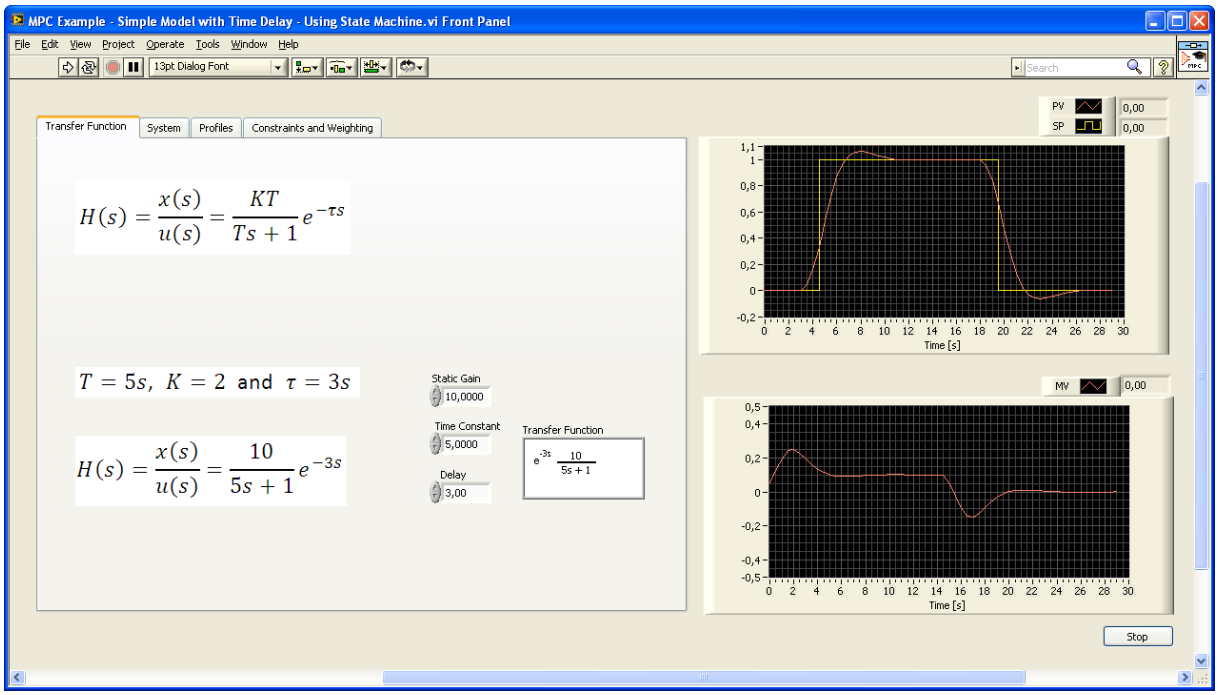


Where



Rest of the code is similar to previous example, except that we have been using a state machine in order to implement the code.

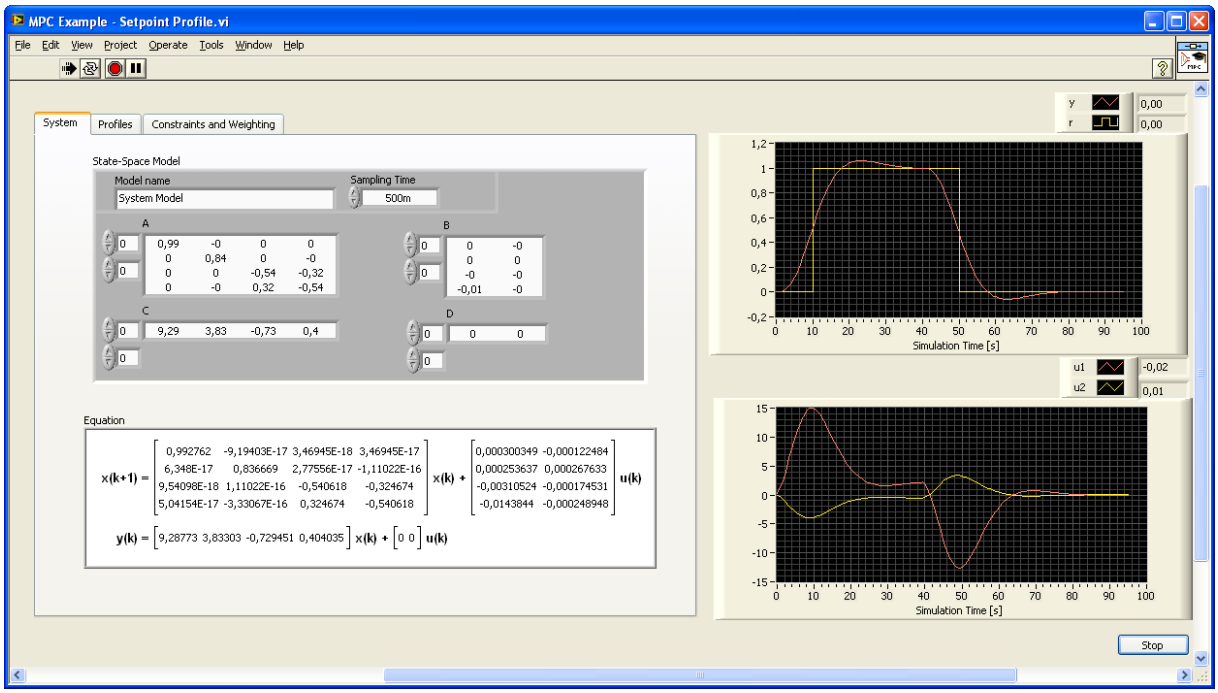
Below we see the front panel:



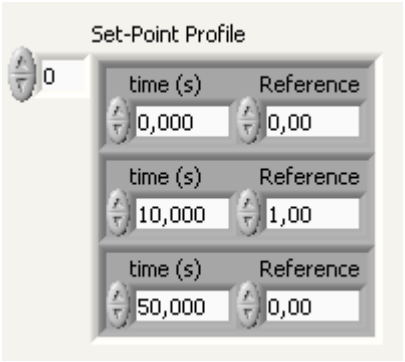
[End of Example]

3.3 Example: Multiple Inputs

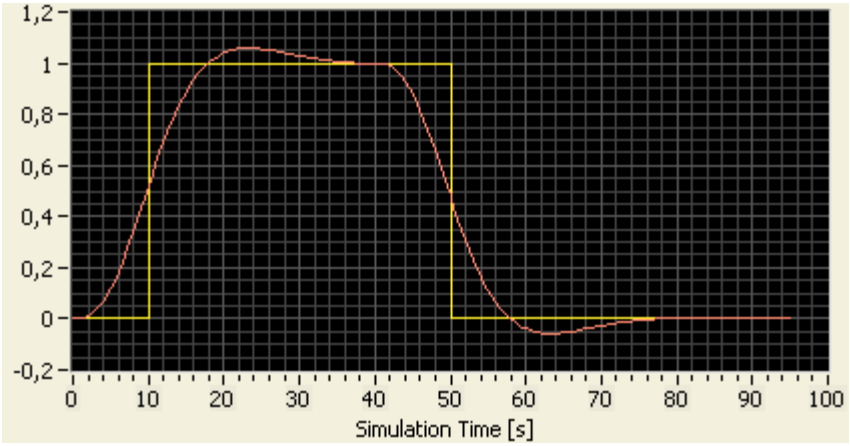
In this example we will use MPC on a MISO system with 2 inputs and 1 output.



We define the setpoint profile the system should follow in advance (the future setpoint is known) and see how the MPC controller works in order to follow the setpoint.



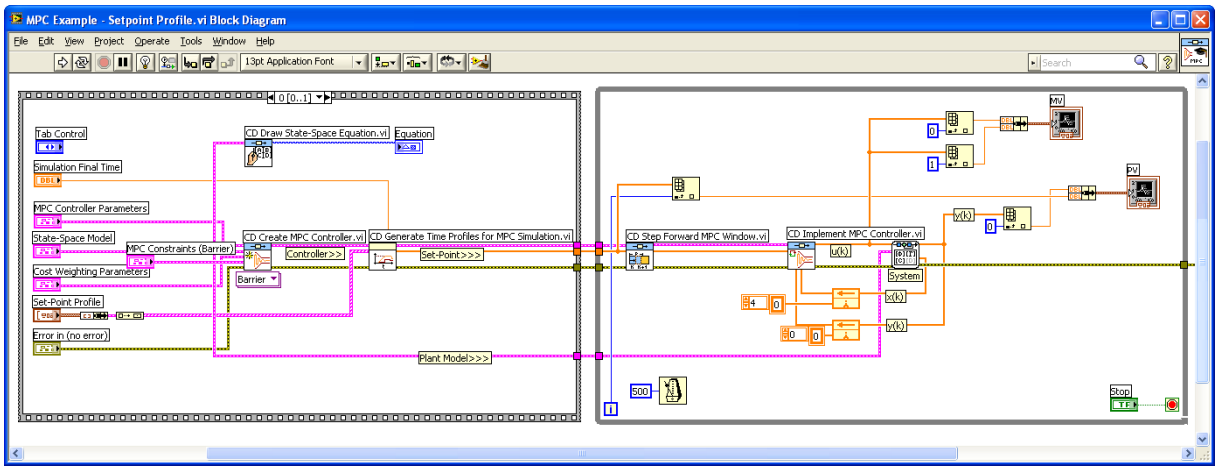
We see that the controller starts to react before the reference actually changes, which is a typically feature for the MPC controller.



Here we see the main difference between a MPC controller and a more traditional PID controller. Another main difference between MPC and PID is that MPC can handle MIMO (Multiple Inputs, Multiple Outputs) systems, while PID is used for SISO systems (Single Input, Single Output).

Block Diagram:

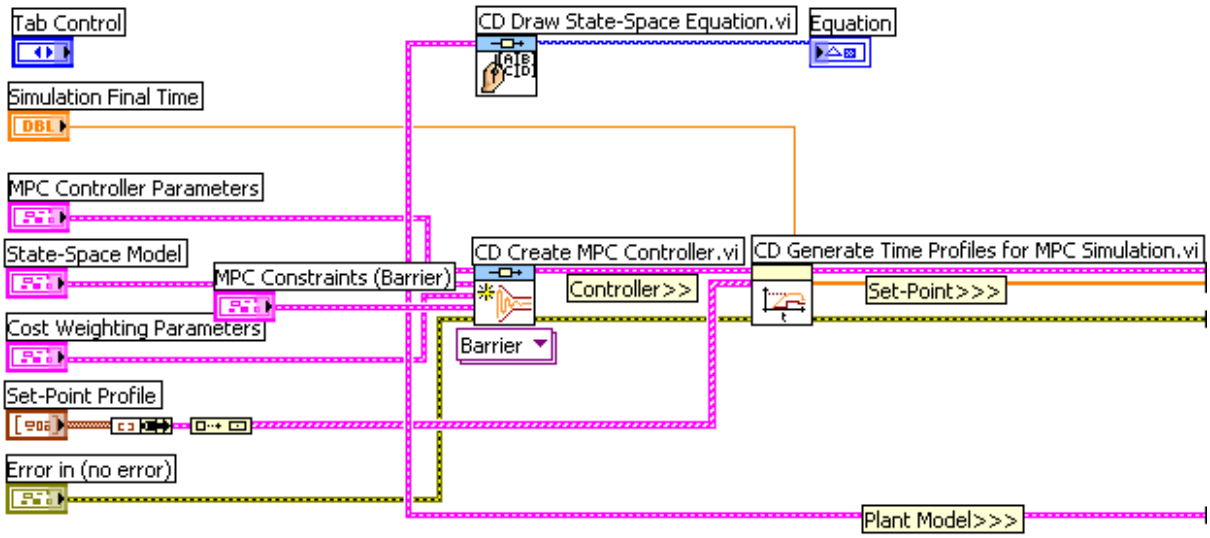
Below we see the block diagram for the program:



We can divide the solution into 2 different parts:

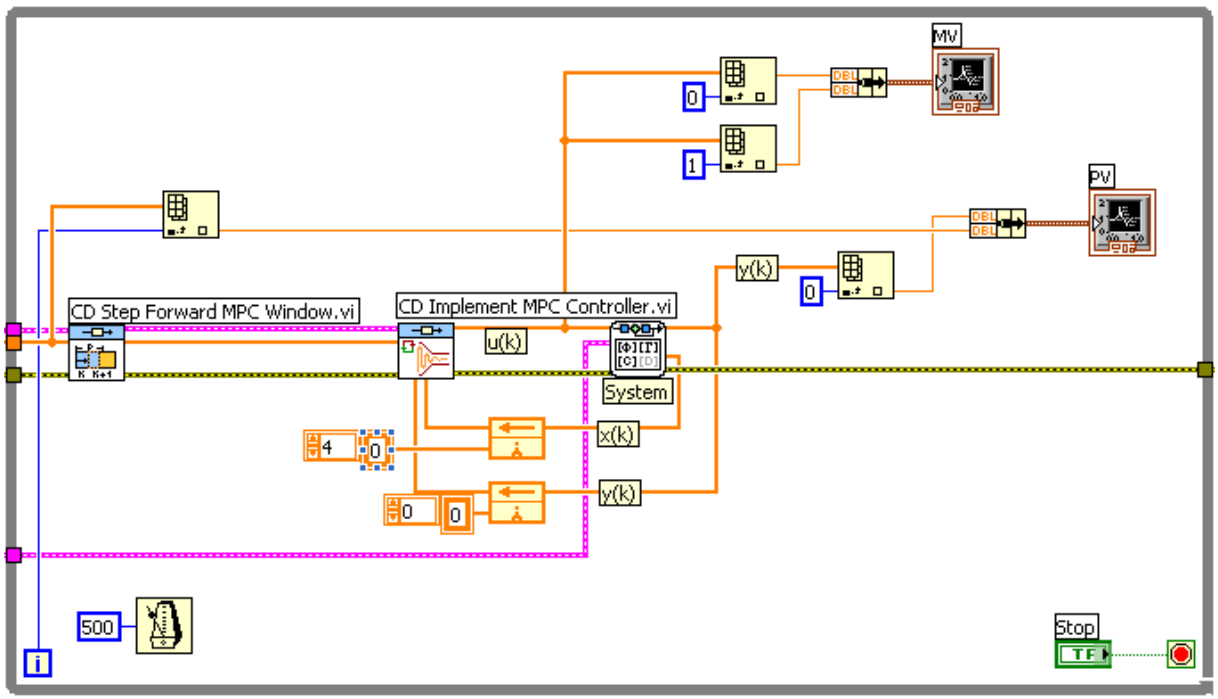
Initialization the MPC Controller:

This is something we do only once when we start the program. We use the “CD Create MPC Controller.vi”.



Run the Controller:

This operation is performed each sample, and this is normally executed inside a loop, e.g. a While Loop. We use the “CD Implement MPC Controller.vi”.



[End of Example]

Model Predictive Control in LabVIEW

Hans-Petter Halvorsen

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

Web: https://www.halvorsen.blog

https://www.halvorsen.blog