Advanced Control Functions
User Manual

MU214610 Rev. B

December 9, 2022

www.altus.com.br

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. This can even put human lives at risk. The user should consider the possible consequences of the defects and should provide additional external installations for safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use. The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

Altus guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers. The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request. These products can use EtherCAT® technology (www.ethercat.org).

COPYRIGHTS

OPEN SOURCE SOFTWARE NOTICE

To obtain the source code under GPL, LGPL, MPL and other open source licenses, that is contained in this product, please contact opensource@altus.com.br. In addition to the source code, all referred license terms, warranty disclaimers and copyright notices may be disclosed under request.

Contents

1. Introduction . . . 1
   1.1. Technical Support . . . 1
   1.2. Warning Messages Used in this Manual . . . 2
2. PIDA_REAL . . . 3
   2.1. Input Parameters . . . 3
      2.1.1. SP - Setpoint . . . 3
      2.1.2. SP_CASC - Setpoint in Cascade Configuration . . . 4
      2.1.3. PV - Process Variable . . . 4
      2.1.4. Kp - Proportional Gain . . . 4
      2.1.5. Td_Kd - Derivate Time or Derivative Gain . . . 4
      2.1.6. Ti_Ki - Integral Time or Integral Gain . . . 5
      2.1.7. Tfd - Filter Time for Derivative Action . . . 5
      2.1.8. BIAS - Offset for MV . . . 7
      2.1.9. ManualMV - Value for MV in Manual Mode . . . 7
      2.1.10. TRK_VAL - MV in Tracking or Override Mode . . . 7
      2.1.11. MaxVarMV - Maximum Variation of MV per Cycle . . . 8
      2.1.12. MaxMV - Maximum Value of MV . . . 9
      2.1.13. MinMV - Minimum Value of MV . . . 9
      2.1.14. DeadBand - Error Deadband . . . 9
      2.1.15. MaxPV - Maximum Value of PV . . . 9
      2.1.16. MinPV - Minimum Value of PV . . . 10
      2.1.17. Indep - Independent Gains . . . 10
      2.1.18. DisableP - Disable Proportional Action . . . 10
      2.1.19. DerivEr - Derivative Action Calculated on Error . . . 10
      2.1.20. CASC - Cascade Mode . . . 10
      2.1.21. TRK_IN - Tracking Mode Indication from a Cascade Slave . . . 11
      2.1.22. Windup_H_IN - Integral High Windup Indication from a Cascade Slave . . . 11
      2.1.23. Windup_L_IN - Integral Low Windup Indication from a Cascade Slave . . . 11
      2.1.24. OVERR - Override Mode . . . 11
      2.1.25. Manual - Manual Mode . . . 11
      2.1.26. Direct - Direct Mode . . . 12
   2.2. Output Parameters . . . 13
      2.2.1. MV - Manipulated Variable . . . 13
      2.2.2. TRK_OUT - Track Mode Indication to a Cascade Master . . . 13
      2.2.3. Windup_H_OUT - Integral High Windup Indication to a Cascade Master . . . 13
      2.2.4. Windup_L_OUT - Integral Low Windup Indication to a Cascade Master . . . 14
      2.2.5. ErrBits - Fatal Error Indication Bits . . . 14
      2.2.6. WarnBits - Warning Indication Bits . . . 15

2.2.7. ProportionalAction - Proportional Action Calculated by FB . . . 15
2.2.8. DerivativeAction - Derivative Action Calculated by FB . . . 15
2.2.9. IntegralActionIntegralActionCalculatedbyFB . . . 15
2.2.10. SampleTime - Cycle Used for Calling the FB . . . 15
3. PIDA_INT . . . 16
3.1. Input Parameters with Type INT . . . 16
3.2. Output Parameters with Type INT . . . 17
4. PIDA_TUNE_REAL . . . 18
4.1. Working Principle of Auto-Tuning Solution . . . 18
4.2. Constraints of Auto-Tuning . . . 19
4.3. Safety Recommendations . . . 20
4.4. Additional Input Parameters for Auto-Tuning . . . 20
4.4.1. AutoTune . . . 20
4.4.2. AutoTuneParam . . . 21
4.4.2.1. PercAmpMV . . . 21
4.4.2.2. PercHystPV . . . 21
4.4.2.3. PercMaxPeakPV . . . 21
4.4.2.4. NumCycles . . . 22
4.4.2.5. Mode . . . 22
4.5. Additional Output Parameters for Auto-Tuning . . . 22
4.5.1. AutoTuneDone . . . 22
4.5.2. AutoTuneError . . . 22
4.5.3. AutoTuneResult . . . 23
4.5.3.1. Kp . . . 23
4.5.3.2. Ti_Ki . . . 23
4.5.3.3. Td_Kd . . . 23
4.5.3.4. Bad_PercAmpMV . . . 23
4.5.3.5. Bad_PercHystPV . . . 23
4.5.3.6. Bad_PercMaxPeakPV . . . 24
4.5.3.7. Bad_NumCycles . . . 24
4.5.3.8. Bad_Mode . . . 24
4.5.3.9. MV_TOO_HIGH . . . 24
4.5.3.10. MV_TOO_LOW . . . 24
4.5.3.11. PV_TOO_HIGH . . . 24
4.5.3.12. PV_TOO_LOW . . . 24
4.5.3.13. PEAK_TOO_HIGH . . . 24
4.5.3.14. PEAK_TOO_LOW . . . 24
4.5.3.15. PV_NOT_STEADY . . . 25
4.5.3.16. CYCLE_TOO_FAST . . . 25
4.5.3.17. HIGH_DEAD_TIME . . . 25
4.6. Recommended SCADA Interface and Code for Copying the Calculated Tuning Parameters . . . 25
5. PIDA_TUNE_INT . . . 26
5.1. Input Parameters with Type INT . . . 26
5.2. Output Parameters with Type INT . . . 27
6. ControlON_OFF . . . 28
6.1. Input Parameters . . . 29
6.1.1. SP - Setpoint . . . 29
6.1.2. PV - Process Variable . . . 29

6.1.3. DeadBand - Dead Band for Error . . . 29
6.1.4. MaxPV – Maximum Value of Process Variable . . . 30
6.1.5. MinPV – Minimum Value of Process Variable . . . 30
6.1.6. Direct – Direct Control . . . 30
6.1.7. Manual – Manual Mode . . . 30
6.1.8. ManualMV – Output in Manual Mode . . . 30
6.2. Output Parameters . . . 30
6.2.1. MV – Manipulated Variable . . . 31
6.2.2. ErrBits – Fatal Error Bits . . . 31
6.2.3. WarnBits – Warning Bits . . . 31
7. ControlPWM . . . 32
7.1. Input Parameters . . . 32
7.1.1. MV – Manipulated Variable . . . 32
7.1.2. MaxMV – Maximum Value of MV . . . 33
7.1.3. MinMV – Minimum Value of MV . . . 33
7.1.4. Period – Period of PWM Output . . . 33
7.2. Output Parameters . . . 33
7.2.1. PWM – Pulse Width Modulated Output . . . 33
7.2.2. Error – Fatal Error Indication . . . 33
7.2.3. Warning – Warning Indication . . . 33
8. ControlLowFilter . . . 34
8.1. Input Parameters . . . 34
8.1.1. InVar – Input Variable . . . 34
8.1.2. Tc – Time Constant . . . 35
8.2. Output Parameters . . . 35
8.2.1. OutVar – Output Variable . . . 35
8.2.2. Error – Fatal Error Indication . . . 35
8.2.3. SampleTime – Cycle Used for Calling the FB . . . 35
9. ControlDelay . . . 36
9.1. Adjusting the Parameter MaxSamplesDelay . . . 36
9.2. Input Parameters . . . 37
9.2.1. InVar – Input Variable . . . 37
9.2.2. DelayTime – Delay Time . . . 37
9.2.3. InitDelayedVar – Initial Value of Delayed Variable . . . 37
9.3. Output Parameters . . . 37
9.3.1. OutVar – Output Variable . . . 37
9.3.2. Error – Fatal Error Indication . . . 38
9.3.3. SampleTime – Cycle Used for Calling the FB . . . 38
10. ControlLag . . . 39
10.1. Input Parameters . . . 39
10.1.1. InVar – Input Variable . . . 39
10.1.2. Gs – Gain . . . 40
10.1.3. Offset – Offset Added to Output . . . 40
10.1.4. Tc – Time Constant . . . 40
10.1.5. Init_OutVar – Initial Value of Output . . . 40
10.1.6. Min_OutVar – Minimum Value of Output . . . 40
10.1.7. Max_OutVar – Maximum Value of Output . . . 40
10.2. Output Parameters . . . 40

10.2.1. OutVar – Output Variable . . . 40
10.2.2. Error – Fatal Error Indication . . . 41
10.2.3. SampleTime – Cycle Used for Calling the FB . . . 41
11. ControlDelayLag . . . 42
11.1. Adjusting the Maximum Delay . . . 43
11.2. Input Parameters . . . 43
11.2.1. InVar – Input Variable . . . 43
11.2.2. DelayTime – Delay Time . . . 43
11.2.3. InitDelayedVar – Initial Value of Delayed Variable . . . 43
11.2.4. Gs – Gain . . . 43
11.2.5. Offset – Offset Added to Output . . . 43
11.2.6. Tc – Time Constant . . . 43
11.2.7. Init_OutVar – Initial Value of Output . . . 44
11.2.8. Min_OutVar – Minimum Value of Output . . . 44
11.2.9. Max_OutVar – Maximum Value of Output . . . 44
11.3. Output Parameters . . . 44
11.3.1. OutVar – Output Variable . . . 44
11.3.2. Error – Fatal Error Indication . . . 44
11.3.3. SampleTime – Cycle Used for Calling the FB . . . 44
12. ControlLeadLag . . . 45
12.1. Input Parameters . . . 46
12.1.1. InVar – Input Variable . . . 46
12.1.2. G – Gain . . . 46
12.1.3. Tlead – Lead Time . . . 46
12.1.4. Tlag – Lag Time . . . 46
12.1.5. Min_OutVar – Minimum Value of Output . . . 47
12.1.6. Max_OutVar – Maximum Value of Output . . . 47
12.1.7. Disable – Disable Output . . . 47
12.2. Output Parameters . . . 47
12.2.1. OutVar – Output Variable . . . 47
12.2.2. Error – Fatal Error Indication . . . 47
12.2.3. SampleTime – Cycle Used for Calling the FB . . . 47
13. ControlSelectMax . . . 48
13.1. Input Parameters . . . 48
13.1.1. MV1 – MV from First PID Control FB . . . 48
13.1.2. MV2 – MV from Second PID Control FB . . . 48
13.2. Output Parameters . . . 48
13.2.1. MV – Selected MV . . . 48
13.2.2. OVERR1 – MV1 is Overridden . . . 48
13.2.3. OVERR2 – MV2 is Overridden . . . 48
14. ControlSelectMin . . . 49
14.1. Input Parameters . . . 49
14.1.1. MV1 – MV from First PID Control FB . . . 49
14.1.2. MV2 – MV from Second PID Control FB . . . 49
14.2. Output Parameters . . . 49
14.2.1. MV – Selected MV . . . 49
14.2.2. OVERR1 – MV1 is Overridden . . . 49
14.2.3. OVERR2 – MV2 is Overridden . . . 49

15. ControlSplitRange . . . 50
15.1. Input Parameters . . . 53
15.1.1. MV – MV from PID . . . 53
15.1.2. SPLIT – Split Point . . . 53
15.1.3. Mode – Operation Mode . . . 54
15.2. Output Parameters . . . 54
15.2.1. MV1 – MV for Valve 1 . . . 54
15.2.2. MV2 – MV for Valve 2 . . . 54
15.2.3. Error – Fatal Error Indication . . . 54
16. Application Notes . . . 55
16.1. PID Simulation with Auto-Tuning . . . 55
16.1.1. Code Description . . . 55
16.1.2. Auto-Tuning Example . . . 56
16.1.3. Example using the Calculated Tuning Parameters in Automatic Mode . . . 57
16.2. Avoiding Bumps when Switching between Automatic and Manual . . . 58
16.3. I/O Diagnostics . . . 58
16.4. Cascade Configurations . . . 59
16.4.1. Special Connections . . . 59
16.4.2. Purpose of Special Connections . . . 60
16.4.3. Additional User Code in Cascade Configurations . . . 61
16.4.3.1. Copy PV to SP in Manual Mode . . . 61
16.4.3.2. Copy MV to ManualMV in Automatic Mode . . . 61
16.4.3.3. Copy PV to SP in Tracking Mode in Master Controllers . . . 61
16.4.3.4. Copy MV to ManualMV in Tracking Mode in Master Controllers . . . 61
16.4.3.5. Copy PV to SP in Automatic/Cascade Mode in Slave Controllers . . . 61
16.5. Override Configurations . . . 61
16.6. Ratio Control . . . 62
16.7. Feed-Forward . . . 63
16.7.1. Simulated Process for Illustrating Feed-Forward Control . . . 63
16.7.2. Implementation in FBD Language . . . 65
16.7.2.1. Network1 . . . 65
16.7.2.2. Network2 . . . 66
16.7.2.3. Network3 . . . 66
16.7.2.4. Network4 . . . 67
16.7.2.5. Network5 . . . 67
16.7.2.6. Network6 . . . 68
16.7.2.7. Network7 . . . 68
16.7.2.8. Network8 . . . 69
16.7.2.9. Network9 . . . 69
16.7.2.10. Network10 . . . 70
16.8. Simulating the Effects of Feed-Forward Controller . . . 70
16.8.1. Effects of Disturbances in Fi without Feed-Forward Controller . . . 70
16.8.2. Effects of Disturbances in Fi with Feed-Forward Controller . . . 71
16.8.3. Effects of Disturbances in Ti without Feed-Forward Controller . . . 72
16.8.4. Effects of Disturbances in Ti with Feed-Forward Controller . . . 72
16.8.5. Conclusions . . . 73
17. PID Tuning Tips . . . 74
17.1. Open Loop Process Characterization . . . 74

17.2. Field and Engineering Units 74
17.3. Controllability of Processes 75
17.4. Maximum Sample Time of Controller 75
17.5. Selection of PID Modes 75
17.6. Synthesis Method 76
17.7. Minimization of Integral of Absolute Error 79
17.8. Hints for Tuning PID in Cascade Configuration 80

1. Introduction

This manual describes several advanced control functions that are packed in library NextoPID, which can be installed in Nexto series PLCs and XTORM series RTUs.

This library contains the following function blocks (FBs) or functions (FUNs):

Function Block (FB) or Function (FUN)	Type	Description
PIDA_REAL	FB	PID control with most parameters with type REAL, with support to override and cascade configurations
PIDA_INT	FB	PID control with most parameters with type INT, with support to override and cascade configurations
PIDA_TUNE_REAL	FB	PID control with most parameters with type REAL, with support to override and cascade configurations, and with auto-tuning
PIDA_TUNE_INT	FB	PID control with most parameters with type INT, with support to override and cascade configurations, and with auto-tuning
ControlON_OFF	FB	ON-OFF control
ControlPWM	FB	Converts a type REAL analog output in a type BOOL digital output with pulse width modulation
ControlLowFilter	FB	Applies a low-pass filter to a type REAL variable
ControlDelay	FB	Applies a delay to a type REAL variable
ControlLag	FB	Applies a first order lag to a type REAL variable with gain and limits
ControlDelayLag	FB	Combination of function blocks ControlDelay and ControlLag
ControlLeadLag	FB	Implements lead and lag times with gain and limits (e.g.: for using in feed-forward controls)
ControlSelectMax	FUN	Selects maximum PID output for using in PID override configurations
ControlSelectMin	FUN	Selects minimum PID output for using in PID override configurations
ControlSplitRange	FUN	Splits a PID output in two outputs for different valves

Table 1: List of advanced control functions in libray NextoPID

ATTENTION

This manual is applicable for version 1.3.0.0 or newer of library NextoPID. Older versions of this library do not contain the functions shown in previous table. Furthermore, this library can be used only with Mastertool MT8500 3.40 or newer.

The following chapters describe these control functions in detail.

After description of these control functions, there are two additional chapters about application notes and PID tuning tips.

1.1. Technical Support

For Altus Technical Support contact in São Leopoldo, RS, call +55 51 3589-9500. For further information regarding the Altus Technical Support existent on other places, see https://www.altus.com.br/en/ or send an email to altus@altus.com.br.

If the equipment is already installed, you must have the following information at the moment of support requesting:

- The model from the used equipments and the installed system configuration
- The product serial number
- The equipment revision and the executive software version, written on the tag fixed on the product's side
- CPU operation mode information, acquired through MasterTool IEC XE
- The application software content, acquired through MasterTool IEC XE
- Used programmer version

1.2. Warning Messages Used in this Manual

In this manual, the warning messages will be presented in the following formats and meanings:

DANGER

Reports potential hazard that, if not detected, may be harmful to people, materials, environment and production.

CAUTION

Reports configuration, application or installation details that must be taken into consideration to avoid any instance that may cause system failure and consequent impact.

ATTENTION

Identifies configuration, application and installation details aimed at achieving maximum operational performance of the system.

2. PIDA_REAL

PIDA_REAL is used for PID control, with support to cascade and override configurations.

ATTENTION

Next chapters describe other FBs for PID control: PIDA_INT, PIDA_TUNE_REAL and PIDA_TUNE_INT. Read the first paragraph of these chapters to discover the special features of each of these FBs.

The following figure shows the input and output parameters of this FB.

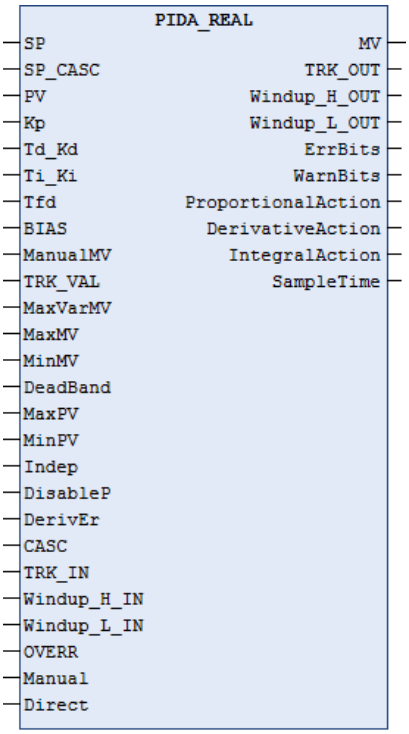

Figure 1: PIDA_REAL parameters

2.1. Input Parameters

The following subsections describe the input parameters.

2.1.1. SP - Setpoint

- Type: REAL
- Range: see parameters MaxPV - Maximum Value of PV and MinPV - Minimum Value of PV

SP informs the target value for PV (process variable).
Error is computed as:

$$Error = SP - PV \tag{1}$$

The main objective of PIDA_REAL control is to keep magnitude of error as low as possible.

2.1.2. SP_CASC - Setpoint in Cascade Configuration

- Type: REAL
- Range: see parameters MaxPV - Maximum Value of PV and MinPV - Minimum Value of PV

SP_CASC works as SP, when PIDA_REAL FB is in cascade mode (see input parameter CASC - Cascade Mode).

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.1.3. PV - Process Variable

- Type: REAL
- Range:see parameters MaxPV - Maximum Value of PV and MinPV - Minimum Value of PV

PV corresponds to the process variable controlled by PIDA_REAL FB. It is normally read from a transmitter, for instance, using an analog input.

The objective of a PIDA_REAL FB is to keep PV close to the selected setpoint (SP or SP_CASC), by controlling the MV output.

2.1.4. Kp - Proportional Gain

- Type: REAL
- Range (see input parameter Indep - Independent Gains):
  - Dependent form (Indep = FALSE): must be bigger than 0.
  - Independent form (Indep = TRUE): must be bigger or equal to 0.

This gain applies to the proportional term of PIDA_REAL control.

In dependent form (Indep = FALSE), it also affects integral and derivative terms.

In independent form (Indep = TRUE), it only affects proportional term.

2.1.5. Td_Kd - Derivate Time or Derivative Gain

- Type: REAL
- Range: must be bigger or equal to 0
- Unit (see input parameter Indep - Independent Gains):
  - Seconds in dependent form (Indep = FALSE)
  - Dimensionless in independent form (Indep = TRUE)

This time or gain applies to the derivative term of PIDA_REAL control.

In dependent form (Indep = FALSE), it means a derivative time, and the derivative gain is calculated as:

$$Kp * Td\_Kd \tag{2}$$

Note that proportional gain (Kp) affects the derivative gain, in dependent form.

In independent form (Indep = TRUE), Td_Kd expresses the derivative gain directly.

2.1.6. Ti_Ki - Integral Time or Integral Gain

- Type: REAL
- Range (see input parameter Indep - Independent Gains):
  - Must be bigger or equal than 0.
  - In dependent form (Indep = FALSE), integral action is disabled if smaller than 0.001 seconds
- Unit (see input parameter Indep - Independent Gains):
  - Seconds in dependent form (Indep = FALSE)
  - Dimensionless in independent form (Indep = TRUE)

This time or gain applies to the integral term of PIDA_REAL control.
In dependent form (Indep = FALSE), it means an integral time, and the integral gain is calculated as:

- If Ti_Ki >= 0.001 seconds:

$$Kp/Ti\_Ki \tag{3}$$

- Integral action disabled, if Ti_Ki < 0.001 seconds:

$$0 \tag{4}$$

Note that proportional gain (Kp) affects the integral gain, in dependent form.
In independent form (Indep = TRUE), Ti_Ki expresses the integral gain directly.

2.1.7. Tfd - Filter Time for Derivative Action

- Type: REAL
- Range: must be bigger or equal than 0 (when equal 0, no filtering occurs)
- Unit: seconds

This parameter can be used to filter the derivative action. It is useful for filtering noise and also for smoothing bumps in the derivative action.

The following figure shows what happens with derivative gain Td_Kd = 10 and independent gains (Indep = TRUE), when a ramp makes PV decrease with a slope of 1 unit per second, with no filtering (Tfd = 0). A kick of magnitude 10 occurs in the derivative action when the ramp starts and when the ramp stops.

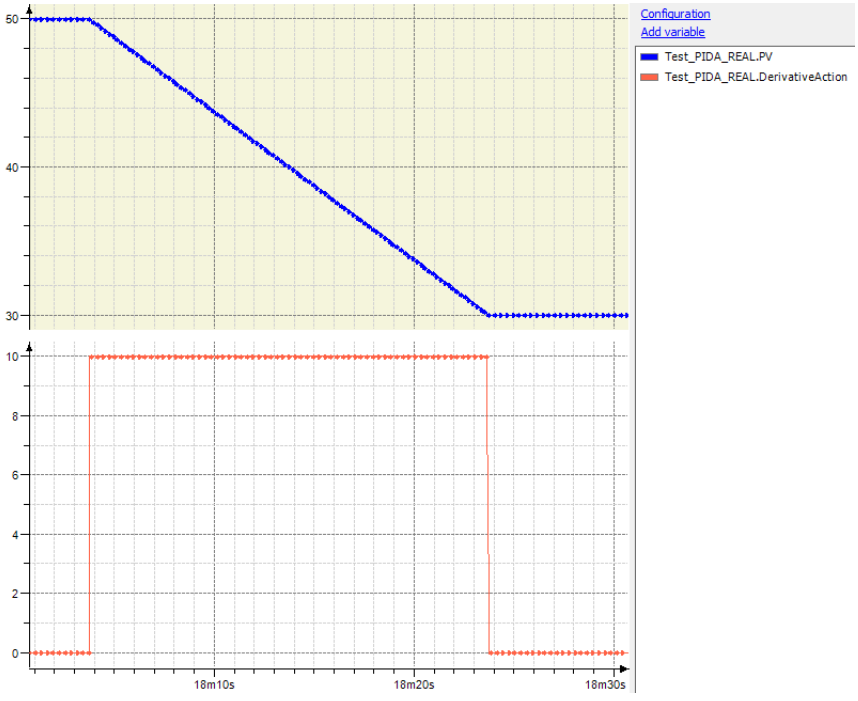

Figure 2: Derivative kick with Tfd = 0 seconds

The next figure shows the same situation as the previous one, this time with a Tfd = 2 seconds. Observe that it takes 2 seconds (time between vertical cursors) for the derivative action to reach 63% of its total span (10).

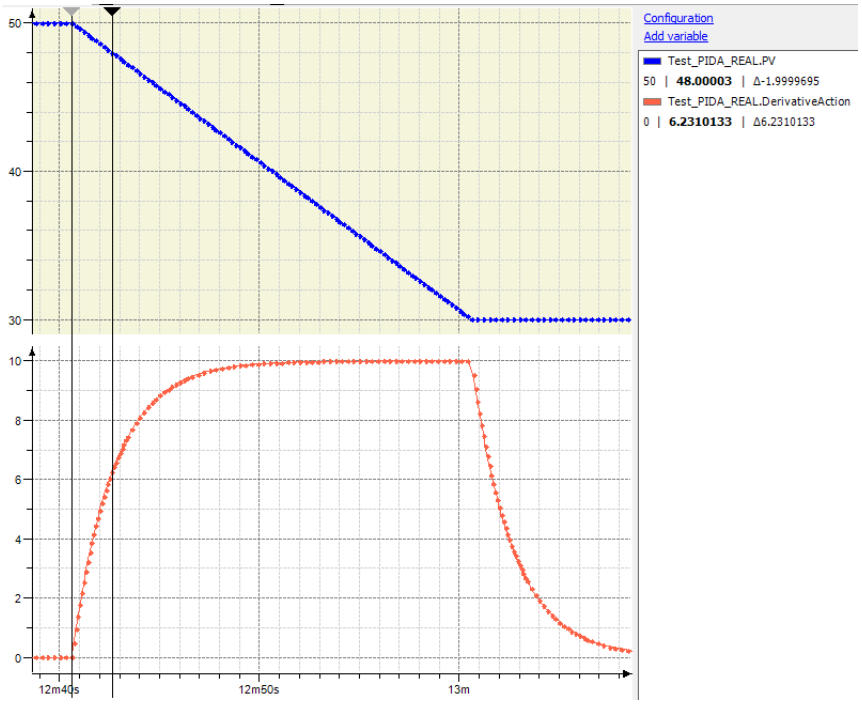

Figure 3: Derivative response with Tfd = 2 seconds

The next figure shows a triangular wave of noise (peak values of ±0.5, variation = 1 each 100 ms) superimposed to PV. The controller has derivative (Td_Kd = 1) and independent (Indep = TRUE) gains and no derivative filtering (Tfd = 0). Note that the derivative action oscillates between ±10.

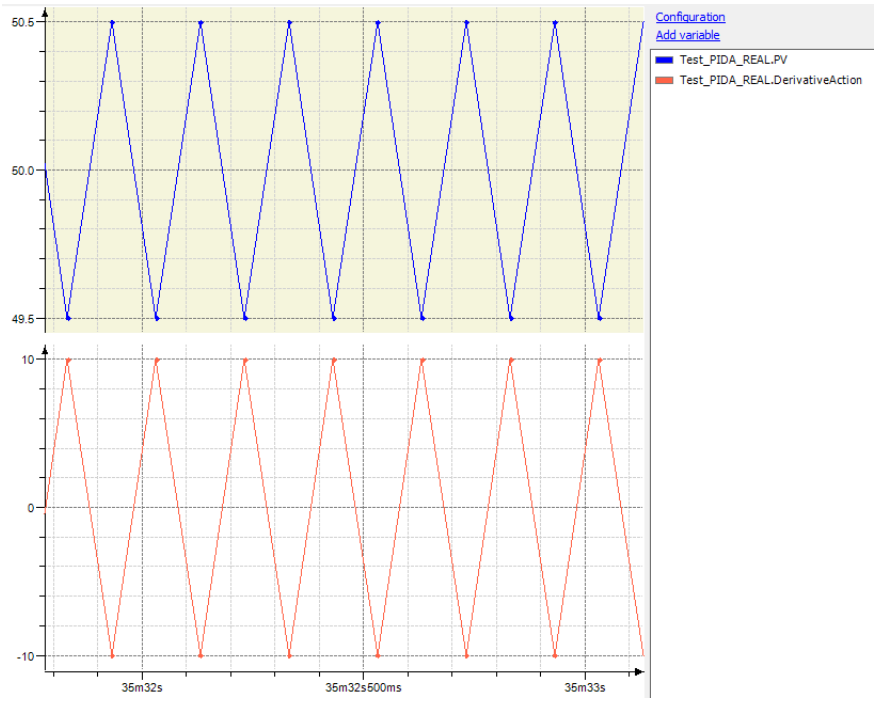

Figure 4: Effect of noise in derivative response with Tfd = 0 seconds

The next figure shows how Tfd = 2 seconds help on reducing the noise in the derivative action. Note that this time the derivative action oscillates between ±0.25, which is much smaller than ±10. The bigger the Tfd value is, the bigger will be the noise attenuation over the derivative filter. However, an excessively big value of Tfd may slow down the derivative action too much.

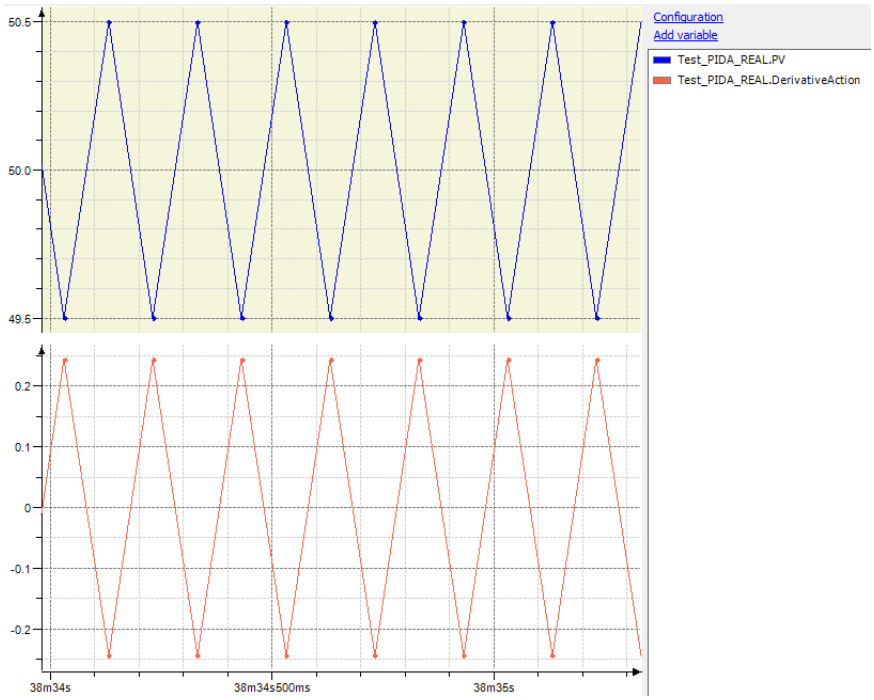

Figure 5: Effect of noise in derivative response with Tfd = 2 seconds

2.1.8. BIAS - Offset for MV

- Type: REAL

BIAS is an offset added to the output of PIDA_REAL FB (MV).

BIAS is typically used for adding a feed forward contribution to the PIDA_REAL control.

2.1.9. ManualMV - Value for MV in Manual Mode

- Type: REAL

This input parameter is copied to the output MV when the manual mode is selected (see input parameter Manual - Manual Mode).

In manual mode, the PIDA_REAL FB does not try to make PV approach the selected setpoint (SP or SP_CASC). The Operator or another system is responsible for the controlling in manual mode.

2.1.10. TRK_VAL - MV in Tracking or Override Mode

- Type: REAL

This input parameter is copied to the MV output when the tracking mode is selected (see input parameter TRK_IN - Tracking Mode Indication from a Cascade Slave). The tracking mode is used in cascade configurations, to indicate to a cascade master that its slave is not in automatic/cascade mode.

This input parameter is also used when the override mode is selected (see input parameter OVERR - Override Mode), for a different calculation of the integral action in the overridden PIDA_REAL FB.

ATTENTION

See section Cascade Configurations and section Override Configurations, inside chapter Application Notes.

2.1.11. MaxVarMV - Maximum Variation of MV per Cycle

- Type: REAL
- Range:
  - Must be bigger or equal than 0.
  - If equal to 0, maximum variation is not checked

This input parameter can be used to impose a limit in the variation of MV, between two consecutive cycles. It can be useful to avoid fast kicks in valves, for instance.

If MaxVarMV equals 0, this limitation is disabled.

If MaxVarMV is bigger than 0, the absolute value of MV variation is limited to be equal to MaxVarMV.

The next figure shows an example without limitation (MaxVarMV = 0) and other with limitation (MaxVarMV = 2). Note that limitation makes the control of the process to be slower. Excessively high values in MaxVarMV may cause problems in the process control as well.

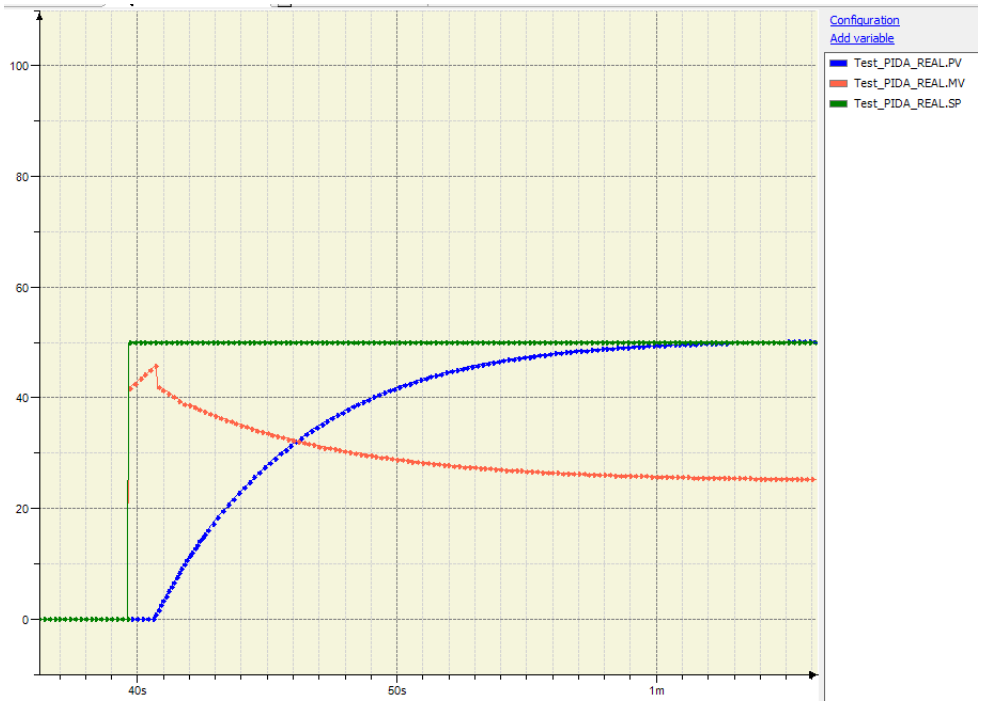

Figure 6: Response with MaxVarMV = 0 (no limitation, faster)

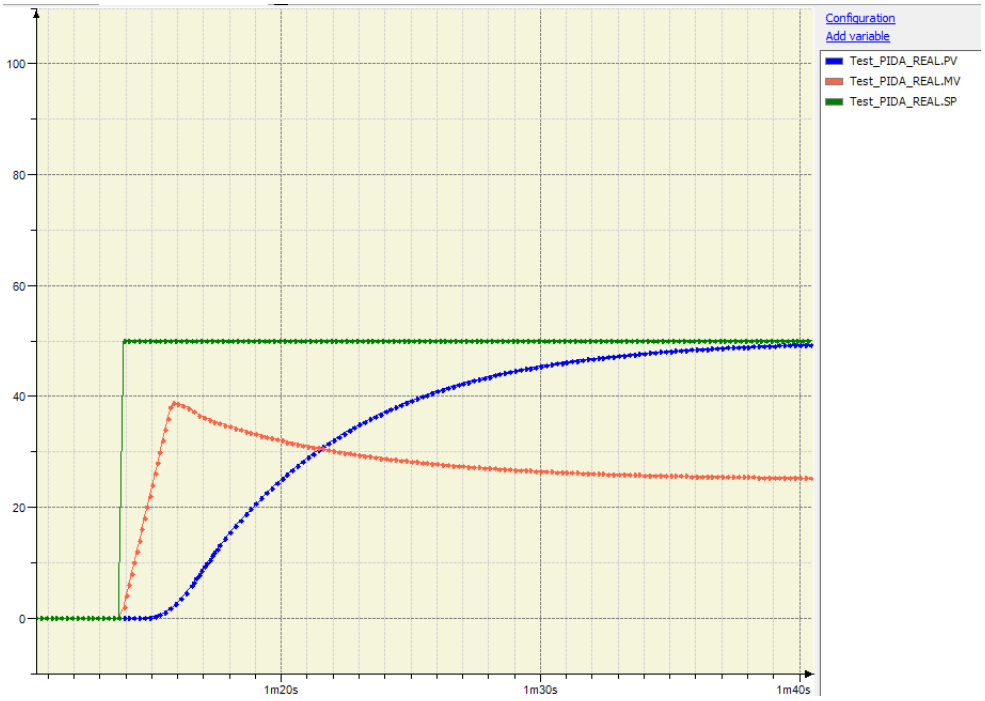

Figure 7: Response with MaxVarMV = 2 (with limitation, slower)

2.1.12. MaxMV - Maximum Value of MV

- Type: REAL
- Range: must be bigger than MinMV

This input parameter imposes a maximum limit to output MV. If MV is calculated and it ends up being bigger than MaxMV, then the value of MaxMV is assigned to MV.

2.1.13. MinMV - Minimum Value of MV

- Type: REAL
- Range: must be smaller than MaxMV

This input parameter imposes a minimum limit to the MV output. If MV is calculated and it ends up being smaller than MinMV, then the value of MinMV is assigned to MV.

2.1.14. DeadBand - Error Deadband

- Type: REAL
- Range: must be bigger or equal to 0

This input parameter defines a minimum absolute value for any error that occurs to cause a control action.

If the absolute value of error (setpoint – PV) is smaller than the DeadBand, the PIDA_REAL controller assumes as if it had an error that is equal zero and thus not changing the output (MV).

It also can be useful to avoid unnecessary valve moves when the error value is in an acceptable low range.

2.1.15. MaxPV - Maximum Value of PV

- Type: REAL
- Range: must be bigger than MinPV

This input parameter imposes a maximum limit to input parameters PV, SP and SP_CASC. If PV, SP or SP_CASC are bigger than MaxPV, then the value of MaxPV is assigned to the respective input parameter (PV, SP or SP_CASC) for internal usage in the PIDA_REAL FB.

2.1.16. MinPV - Minimum Value of PV

- Type: REAL
- Range: must be smaller than MaxMV

This input parameter imposes a minimum limit to the input parameters PV, SP and SP_CASC. If PV, SP or SP_CASC are smaller than MinPV, then the value of MinPV is assigned to the respective input parameter (PV, SP or SP_CASC) for internal usage in the PIDA_REAL FB.

2.1.17. Indep - Independent Gains

- Type: BOOL

When TRUE, this input parameter indicates that proportional, derivative and integral gains are independent; otherwise, they are dependent.

It is useful to set Indep as TRUE if the user wants to change Kp to affect only the proportional action, without affecting derivative or integral actions.

See previous descriptions of input parameters Kp - Proportional Gain, Td_Kd - Derivate Time or Derivative Gain and Ti_Ki - Integral Time or Integral Gain.

2.1.18. DisableP - Disable Proportional Action

- Type: BOOL

When TRUE, this input parameter disables the proportional action, but only with dependent gains (Indep = FALSE). This may be necessary, for instance, if the user wants a pure integral controller with a dependent gain (Kp / Ti_Ki). In this case, the user cannot assign Kp = 0 to disable the proportional action, because this would also disable the integral action.

When using independent gains (Indep = TRUE), the input parameter DisableP is ignored. In this case, it is possible to disable the proportional action by assigning Kp = 0, without affecting the derivative and integral actions.

2.1.19. DerivEr - Derivative Action Calculated on Error

- Type: BOOL

When TRUE, the derivative action is calculated on error; otherwise it is calculated on PV.

Note that:

$$Error = Setpoint - PV \tag{5}$$

So, when DerivEr is TRUE, a derivative action can be caused by a change in PV or by a change in Setpoint. When DerivEr is FALSE, a derivative action can only be caused by a change in PV.

Sometimes big changes occur in the Setpoint between two consecutive PLC cycles. In the other hand, big changes in PV are not expected between two consecutive PLC cycles.

Therefore most applications calculate the derivative action on PV (DerivEr = FALSE), to avoid a kick on MV when the Setpoint changes abruptly.

2.1.20. CASC - Cascade Mode

- Type: BOOL

When TRUE, it indicates that PIDA_REAL is executing in cascade mode. In this case, the input parameter SP_CASC corresponds to the Setpoint, instead of SP.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.1.21. TRK_IN - Tracking Mode Indication from a Cascade Slave

- Type: BOOL

When TRUE, indicates that the output MV must be copied from the input parameter TRK_VAL.
See description of TRK_VAL - MV in Tracking or Override Mode.

ATTENTION

See section Cascade Configurations and section Override Configurations, inside chapter Application Notes.

2.1.22. Windup_H_IN - Integral High Windup Indication from a Cascade Slave

- Type: BOOL

This input parameter must be used in a cascade master, and is connected to the output Windup_H_OUT of the corresponding cascade slave. It is necessary for avoiding windup problems in the integral action.
If the input is TRUE, the integral action of the cascade master is not increased, but it can be decreased.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.1.23. Windup_L_IN - Integral Low Windup Indication from a Cascade Slave

- Type: BOOL

This input parameter must be used in a cascade master, and is connected to the output Windup_L_OUT of the corresponding cascade slave. It is necessary for avoiding windup problems in the integral action.
If the input is TRUE, the integral action of cascade master is not decreased, but it can be increased.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.1.24. OVERR - Override Mode

- Type: BOOL

When TRUE, it indicates that this PIDA_REAL FB is overridden by another PIDA_REAL FB, in override configurations.
In this situation, the integral action of PIDA_REAL FB is calculated differently using the TRK_VAL input parameter.

ATTENTION

See section Override Configurations inside chapter Application Notes.

2.1.25. Manual - Manual Mode

- Type: BOOL

When TRUE, it indicates that this PIDA_REAL FB is in manual mode, and MV is copied from input parameter ManualMV.
See the description from the input parameter ManualMV - Value for MV in Manual Mode.

2.1.26. Direct - Direct Mode

- Type: BOOL

PIDA_REAL can be configured for direct control (Direct = TRUE) or reverse control (Direct = FALSE). Some controlled processes require direct control while others required reverse control.

If this parameter is not configured correctly, a positive feedback control will be established, and PIDA_REAL will not be able to control the process.

The following rules must be used to get a correct configuration with negative feedback:

- Direct control must be used when MV must be increased for controlling when PV increases. In other words, an increase in MV causes PV to decrease.
- Reverse control must be used when MV must be decreased for controlling when PV increases. In other words, a decrease in MV causes PV to decrease.

The following figures show examples with direct and reverse control:

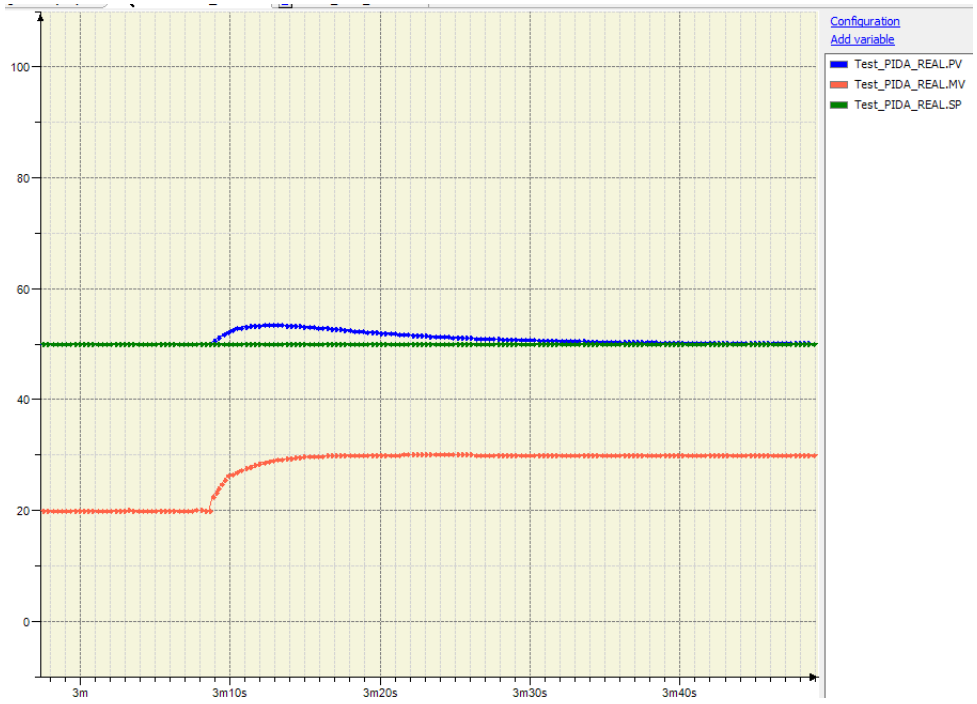

Figure 8: Example with direct control (Direct = TRUE)

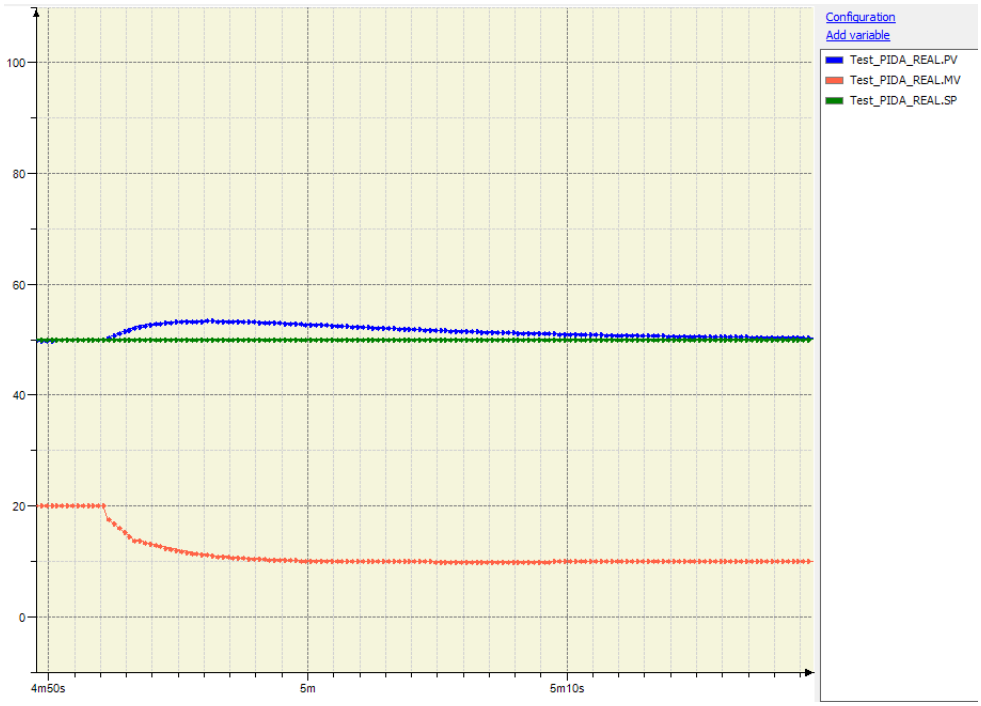

Figure 9: Example with reverse control (Direct = FALSE)

2.2. Output Parameters

The following subsections describe the output parameters.

2.2.1. MV - Manipulated Variable

- Type: REAL

MV corresponds to the manipulated variable for controlling the process. It is normally written to an actuator, for instance, using an analog output.

The objective of a PIDA_REAL FB is to keep PV close to the selected setpoint (SP or SP_CASC) by controlling the MV output.

2.2.2. TRK_OUT - Track Mode Indication to a Cascade Master

- Type: BOOL

In cascade configurations, this output is set by a cascade slave when this is slave is not in automatic or is not in cascade mode.

This output must be connected to input parameter TRK_IN of the corresponding cascade master, so that the master can know it is not controlling the cascade slave.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.2.3. Windup_H_OUT - Integral High Windup Indication to a Cascade Master

- Type: BOOL

This output parameter must be used in a cascade slave, and is connected to input Windup_H_IN of the corresponding cascade master. It is necessary for avoiding windup problems in the integral action of the cascade master.

See description of the input parameter Windup_H_IN - Integral High Windup Indication from a Cascade Slave.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.2.4. Windup_L_OUT - Integral Low Windup Indication to a Cascade Master

- Type: BOOL

This output parameter must be used in a cascade slave, and is connected to input Windup_L_IN of the corresponding cascade master. It is necessary for avoiding windup problems in the integral action of the cascade master.

See the description of the input parameter Windup_L_IN - Integral Low Windup Indication from a Cascade Slave.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.

2.2.5. ErrBits - Fatal Error Indication Bits

- Type: PIDA_ERROR_BITS

This output parameter is a structure with several fatal error bits that indicates bad input parameters or a bad sample time. These errors prevent the execution of the PIDA_REAL FB.

- MaxMinMV bit is set when MaxMV <= MinMV;
- MaxMinPV bit is set when MaxPV <= MinPV;
- Kp bit is set when Kp <= 0 with Indep = FALSE, or KP < 0 with Indep = TRUE;
- Td_Kd bit is set when Td_Kd < 0;
- Ti_Ki bit is set when Ti_Ki < 0;
- Tfd bit is set when Tfd < 0;
- MaxVarMV bit is set when MaxVarMV < 0;
- DeadBand bit is set when DeadBand < 0;
- SampleTime bit is set when the SampleTime output is bigger than 60 seconds or smaller than 0.001 seconds, or if the FB is called inside a non-cyclic task.

The following figure details the struct type PIDA_ERROR_BITS.

```
TYPE PIDA_ERROR_BITS :
STRUCT
    MaxMinMV : BIT;          // MaxMV <= MinMV
    MaxMinPV : BIT;          // MaxPV <= MinPV
    Kp : BIT;                // Kp <= 0 with Indep = FALSE, OR Kp < 0 with Indep = TRUE
    Td_Kd : BIT;             // Td_Kd < 0
    Ti_Ki : BIT;             // Ti_Ki < 0
    Tfd : BIT;               // Tfd < 0
    MaxVarMV : BIT;          // MaxVarMV < 0
    DeadBand : BIT;          // DeadBand < 0
    SampleTime : BIT;        // Sample time < 0.001 seconds or > 60 seconds
END_STRUCT
END_TYPE
```

Figure 10: Struct PIDA_ERROR_BITS

2.2.6. WarnBits - Warning Indication Bits

- Type: PIDA_WARNING_BITS

This output parameter is a structure with several warning bits that indicates limits reached. These warnings do not prevent execution of PIDA_REAL FB.

- PV_MaxPV bit is set when PV is being limited by MaxPV;
- PV_MinPV bit is set when PV is being limited by MinPV;
- SP_MaxPV bit is set when SP is being limited by MaxPV;
- SP_MinPV bit is set when SP is being limited by MinPV;
- SP_CASC_MaxPV bit is set when SP_CASC is being limited by MaxPV;
- SP_CASC_MinPV bit is set when SP_CASC is being limited by MinPV;
- MV_MaxMV bit is set when MV is being limited by MaxMV;
- MV_MinMV bit is set when MV is being limited by MinMV;
- MV_P_MaxVarMV bit is set when a positive variation of MV is being limited by MaxVarMV;
- MV_N_MaxVarMV bit is set when a negative variation of MV is being limited by MaxVarMV.

The following figure details the struct type PIDA_WARNING_BITS.

```
TYPE PIDA_WARNING_BITS :
STRUCT
    PV_MaxPV : BIT;         // PV is being limited by MaxPV
    PV_MinPV : BIT;         // PV is being limited by MinPV
    SP_MaxPV : BIT;         // SP is being limited by MaxPV
    SP_MinPV : BIT;         // SP is being limited by MinPV
    SP_CASC_MaxPV : BIT;    // SP_CASC is being limited by MaxPV
    SP_CASC_MinPV : BIT;    // SP_CASC is being limited by MinPV
    MV_MaxMV : BIT;         // MV is being limited by MaxMV
    MV_MinMV : BIT;         // MV is being limited by MinMV
    MV_P_MaxVarMV : BIT;    // Positive variation of MV is being limited by MaxVarMV
    MV_N_MaxVarMV : BIT;    // Negative variation of MV is being limited by MaxVarMV
END_STRUCT
END_TYPE
```

Figure 11: Struct PIDA_WARNING_BITS

2.2.7. ProportionalAction - Proportional Action Calculated by FB

- Type: REAL

This output parameter show the proportional action calculated by PIDA_REAL FB, and it is merely informative.

2.2.8. DerivativeAction - Derivative Action Calculated by FB

- Type: REAL

This output parameter show the derivative action calculated by PIDA_REAL FB, and it is merely informative.

2.2.9. IntegralActionIntegralActionCalculatedbyFB

- Type: REAL

This output parameter show the integral action calculated by PIDA_REAL FB, and it is merely informative.

2.2.10. SampleTime - Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

3. PIDA_INT

PIDA_INT works similarly to PIDA_REAL, described in the previous chapter. Both FBs are used for PID control and support features like cascade and override configurations.

The following figure shows the input and output parameters of this FB.

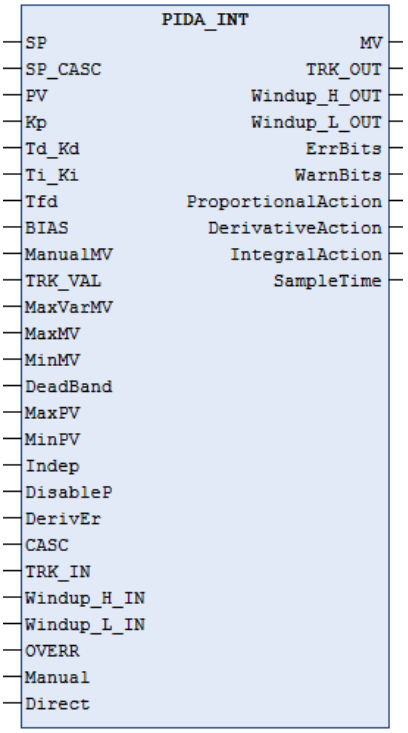

Figure 12: PIDA_INT parameters

In PIDA_REAL, all numeric parameters have the REAL type. In PIDA_INT, most numeric parameters have the INT type, and few parameters have the REAL type.

If the user selects PIDA_INT instead of PIDA_REAL, the following advantages can be achieved:

- Each instance consumes less memory. This may be important in redundant PLC applications with hundreds of PIDA_INT instances, to save memory synchronization time between the redundant PLCs.
- For some communication protocols, like MODBUS TCP, the number of bytes exchanged cyclically with the SCADA system can be reduced significantly, especially if the PLC application has hundreds of PIDA_INT instances.
- The application gets easier when the PLC manages field units, and conversions between engineering units and field units are executed by the SCADA system.

In the other hand, PIDA_INT should not be used if the PLC must manage engineering units internally.

This chapter will not describe again all the parameters of PIDA_INT. Instead, it will only present a list of those parameters which had a change of type from REAL in PIDA_REAL to INT in PIDA_INT. The meaning of these parameters is the same in PIDA_REAL and PIDA_INT. So, for using PIDA_INT, the user must read the chapter PIDA_REAL, and just consider that some parameters changed their type from REAL to INT.

3.1. Input Parameters with Type INT

The following input parameters have type REAL in PIDA_REAL and type INT in PIDA_INT:

- SP
- SP_CASC
- PV
- BIAS
- ManualMV

- TRK_VAL
- MaxVarMV
- MaxMV
- MinMV
- DeadBand
- MaxPV
- MinPV

3.2. Output Parameters with Type INT

The following output parameters have type REAL in PIDA_REAL and type INT in PIDA_INT:

- MV

4. PIDA_TUNE_REAL

PIDA_TUNE_REAL works similarly to PIDA_REAL for PID control, but has auto-tuning as an additional function. This function calculates automatically the tuning parameters for the PID controller (Kp, Td_Kd and Ti_Ki).

The following figure shows the input and output parameters of this FB.

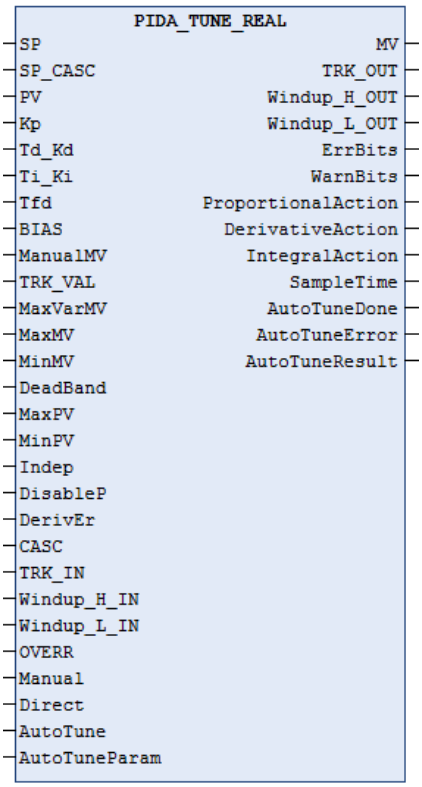

Figure 13: PIDA_TUNE_REAL parameters

All parameters that exist in PIDA_REAL also exist in PIDA_TUNE_REAL, with the same types and meanings. Furthermore, PIDA_TUNE_REAL has few additional parameters for managing the auto-tuning function.

This chapter will not repeat the descriptions of all parameters that are common for PIDA_REAL and PIDA_TUNE_REAL. For these parameters, the user must read the descriptions in the chapter PIDA_REAL. The current chapter will only describe those additional parameters related to auto-tuning.

Before describing the additional parameters for auto-tuning, this chapter will describe the working principle of the auto-tuning solution, the constraints that may limit its usage, and some safety recommendations.

4.1. Working Principle of Auto-Tuning Solution

This auto-tuning function is based in the relay method, and aims to calculate the tuning parameters of the PID controller (Kp, Td_Kd and Ti_Ki). This section explains the working principles of this function, with the help of Figure 14.

The function can be started in manual mode or automatic mode, but manual mode is safer (see section Safety Recommendations).

Before starting the function, the user should check if the process is steady (PV and MV steady). In the following figure, PV is assumed to be steady at RefPV and MV at RefMV. Of course, some noise may exist in PV, but the PV oscillations due to this noise must be smaller than the hysteresis parameter configured by the user for the function. In the following figure, the configured hysteresis is assumed to be H, defining a window of size 2 * H around RefPV.

The function is started at time t0 in the figure, and assumes that the current value of MV is RefMV. During 3 seconds (from t0 to t1), the function calculates RefPV as the average value of PV in this interval, and verifies if PV stays inside the range RefPV ± H during this interval. If PV gets out of this window, the auto-tuning is aborted.

The user must also configure a safety limit (S in the figure) around RefPV, to avoid big oscillations of PV. This safety limit defines a window of size 2 * S around RefPV. After t1 until the end of auto-tuning, PV must stay inside the range RefPV ± S. If PV gets out of this window, the auto-tuning is also aborted.

The auto-tuning function controls MV. During the execution of the function MV may assume three different values, as the following figure shows:

- RefMV (initial value)
- RefMV + M
- RefMV - M

The value M comes from another user configuration parameter.

From t1 to t2, the auto-tuning makes some initializations. After t2, the function makes several measurement cycles. In the example shown in figure, 3 cycles were configured by the user (first cycle from t2 to t3, second cycle from t3 to t4, third cycle from t4 to t5). The objective of having several cycles is to make averages of measurements, and therefore produce better tuning results. If the user wants to accelerate auto-tuning, one single cycle can be configured.

After the last cycle ends (time t5 in the figure), auto-tuning results are calculated. The results may be valid or invalid. Diagnostics indicate the validity of results, as well as the reason of possible failures. The tuning parameters calculated are not copied automatically over the current gains. The user must provide a small piece of code to make this copy. This is discussed in the section Recommended SCADA Interface and Code for Copying the Calculated Tuning Parameters.

The subsections that describe input and output parameters for auto-tuning give more details about configuration parameters, calculated results and how to use these results.

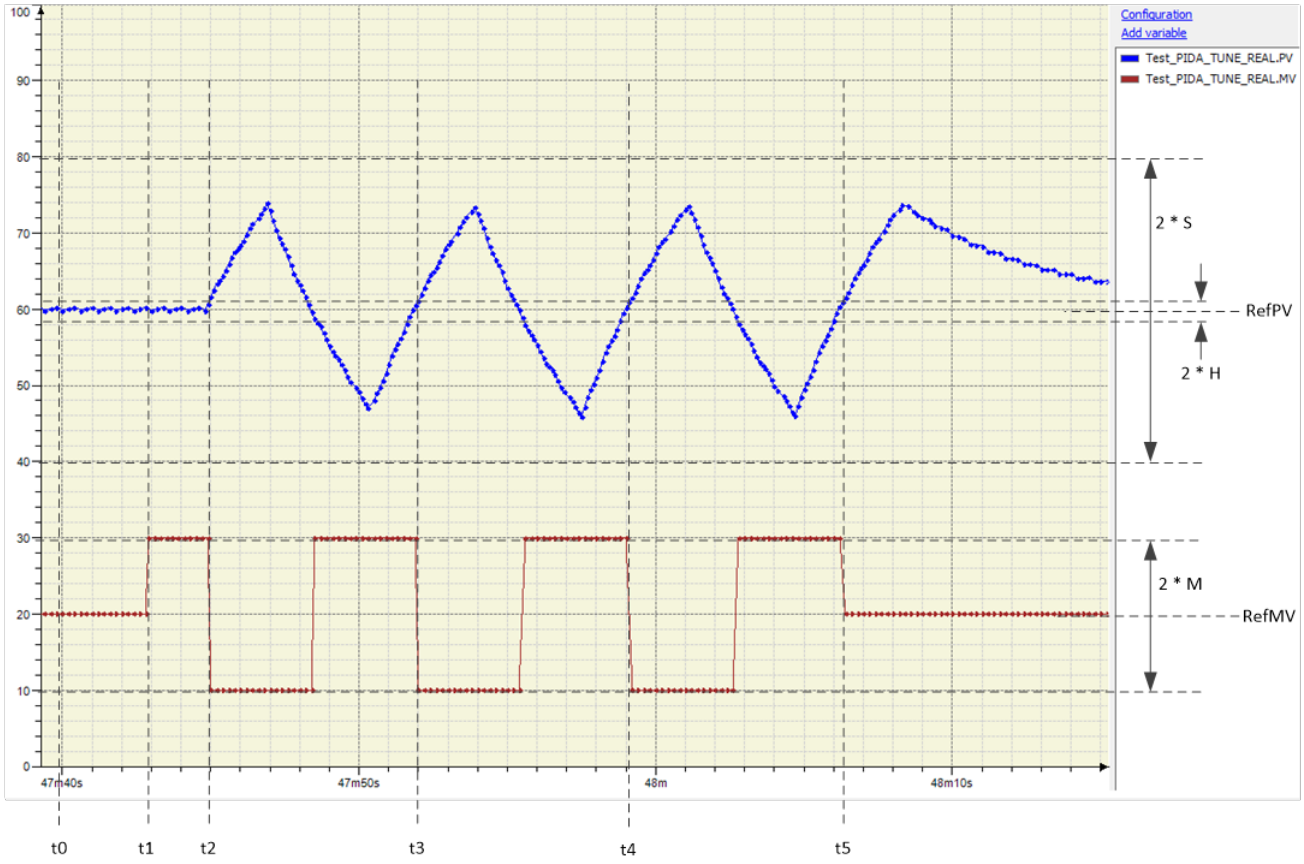

Figure 14: Working principle of auto-tuning

4.2. Constraints of Auto-Tuning

This auto-tuning function will not tune properly any kind of process. The main known constraints are the following;

- It works better for processes that can be approximated to a linear first order process with the dead time. Such a process is characterized by a static gain, a dead time and a time constant.

- It may produce bad tuning results if the dead time is too short when compared to the time constant, or when compared to PLC sample time (see output parameter SampleTime - Cycle Used for Calling the FB).
- It may also produce bad tuning results if the dead time is too long when compared to the time constant.
- Before starting auto-tuning, the process must be steady, that is, PV and MV must be steady.
- During the execution of auto-tuning, the process must not suffer significant disturbances, that is, only MV should influence it.

4.3. Safety Recommendations

The auto-tuning function will try to automatically detect conditions where the calculated tuning results are not good, according to the situations mentioned in the section Constraints of Auto-Tuning. However, some situations that produce bad tuning results may not be detected automatically. Therefore, it is important to follow some safety recommendations before using the tuning results calculated by the function.

- Although auto-tuning can be started both in automatic and manual mode, it is safer to start it in manual mode. If executed in automatic mode, a bad control could start as soon as the calculated tuning parameters are copied over to the active PIDA_TUNE_REAL tuning parameters.
- During auto-tuning, it is important to limit the oscillation of PV, using a configuration parameter (see S in Figure 14).
- Before running PIDA_TUNE_REAL in automatic mode for the first time, after using tuning parameters calculated by auto-tuning, it is recommended to create a command and a piece of code that returns to manual mode with a safe value in ManualMV. This piece of code can be started by a SCADA command or by a button connected to a digital input.

4.4. Additional Input Parameters for Auto-Tuning

This section presents only those input parameters related specifically to auto-tuning. See chapter PIDA_REAL for the additional input parameters.

4.4.1. AutoTune

- Type: BOOL

This input parameter is used for starting, ending and aborting the auto-tuning function. When AutoTune is FALSE, the auto-tuning function is not running and the following output parameters assume the reset default values:

- AutoTuneDone is FALSE;
- AutoTuneError is FALSE;
- All the auto-tuning results that compose AutoTuneResult are reset (0 for numeric values, FALSE for boolean values).

In the rising edge of AutoTune, the auto-tuning function is started.

While AutoTune is TRUE, the MV output of PIDA_TUNE_REAL is controlled by the auto-tuning function. MV is switched by the auto-tuning function between three possible values (see Figure 14):

- RefMV
- RefMV + M
- RefMV - M

If the user resets the input AutoTune to FALSE before the auto-tuning function ended the last cycle, the function is aborted, and MV returns to RefMV. After this, MV is controlled by PIDA_TUNE_REAL in manual or automatic mode, and so MV can possibly change.

When the auto-tuning function ends the last cycle, or aborts itself due to detection of an error:

- Output parameter AutoTuneDone changes to TRUE to indicate that the function has completed the calculation of results;
- Output parameter AutoTuneError is FALSE if auto-tuning ended normally without errors; otherwise it is to TRUE;
- Output parameter AutoTuneResult shows the calculated tuning parameters and the detected errors (if any);
- MV returns to RefMV.

It is important to note that the AutoTune input keeps the value as set to TRUE after the auto-tuning function ends (AutoTuneDone = TRUE). Normally a small user code must be created to copy the calculated tuning parameters over the current tuning parameters, if no errors were detected (AutoTuneDone = TRUE and AutoTuneError = FALSE). After this, the AutoTune input can be reset, and then PIDA_TUNE_REAL will take control of MV again in automatic or manual mode. An example of this small user code is shown in the section Recommended SCADA Interface and Code for Copying the Calculated Tuning Parameters.

4.4.2. AutoTuneParam

- Type: AUTOTUNE_PARAM

This input parameter is a structure that contains the configuration parameters for auto-tuning. The following figure shows the definition of this data structure:

```
TYPE AUTOTUNE_PARAM :
STRUCT
    PercAmpMV : REAL;       // Percentage of "MaxMV - MinMV" for amplitude of relay signal (range: 0.1% ... 50%)
    PercHystPV : REAL;      // Percentage of "MaxPV - MinPV" for hysteresis of PV (range: 0.01% ... 1%)
    PercMaxPeakPV : REAL;   // Percentage of "MaxPV - MinPV" for maximum peak of PV during auto-tuning for safety (range: 0.1% ... 50%)
    NumCycles : UINT;       // Number of measurement cycles during auto-tuning for averaging results (range: 1 ... 10)
    Mode : AUTOTUNE_MODE;   // Mode for autotuning (PID configuration)
END_STRUCT
END_TYPE
```

Figure 15: Struct AUTOTUNE_PARAM

4.4.2.1. PercAmpMV

- Type: REAL
- Range: 0.1% to 50%

This configuration parameter indicates the amplitude of the MV oscillation during the auto-tuning, expressed as a percentage of the MV range (MaxMV - MinMV). This parameter is used to calculate the value M in Figure 14:

$$M = PercAmpMV * (MaxMV - MinMV)/100 \tag{6}$$

If this parameter is too small, the corresponding peaks in PV may also be too small. In this case, the auto-tuning function may detect an error (see section PEAK_TOO_LOW).

If this parameter is too big, the corresponding peaks in PV may also be too high. In this case, the auto-tuning function may also detect an error (see section PEAK_TOO_HIGH).

4.4.2.2. PercHystPV

- Type: REAL
- Range: 0.01% to 1%

This configuration parameter indicates the PV hysteresis during the auto-tuning, expressed as a percentage of the PV range (MaxPV - MinPV). This parameter is used to calculate the value H in the Figure 14:

$$H = PercHystPV * (MaxPV - MinPV)/100 \tag{7}$$

This parameter must be bigger than the noise level in PV, otherwise the auto-tuning function may detect an error (see section PV_NOT_STEADY).

In the other hand, it must be as small as possible, to produce more accurate tuning results, and also for avoiding other errors that can be detected by the function (see section PEAK_TOO_LOW).

4.4.2.3. PercMaxPeakPV

- Type: REAL
- Range: 0.1% to 50%

This configuration parameter indicates the maximum peak of PV during the auto-tuning for safety, expressed as a percentage of PV range (MaxPV - MinPV). This parameter is used to calculate value S in Figure 14:

$$S = PercMaxPeakPV * (MaxPV - MinPV)/100 \tag{8}$$

If a peak of PV gets out of range RefPV ± S, the auto-tuning is aborted and MV returns to the initial value RefMV (see Figure 14). A good configuration of this parameter is important for safety because it limits the oscillations of PV during auto-tuning.

4.4.2.4. NumCycles

- Type: UINT
- Range: 1 to 10

This configuration parameter indicates the number of measurement cycles used by the auto-tuning function for averaging the results. In the example shown in Figure 14, its value is 3.

Bigger values should produce more accurate results, but take more time.

4.4.2.5. Mode

- Type: AUTOTUNE_MODE

This configuration parameter indicates the PID configuration (P, PI or PID). Other configurations (D, PD, I, etc) are not supported by the auto-tuning function.

The following figure shows the enumeration AUTOTUNE_MODE.

```
TYPE AUTOTUNE_MODE :
(
    AUTOTUNE_PID := 0,  // PID configuration
    AUTOTUNE_PI := 1,   // PI configuration
    AUTOTUNE_P := 2     // P configuration
);
END_TYPE
```

Figure 16: Enumeration AUTOTUNE_MODE

ATTENTION

See section Selection of PID Modes inside chapter PID Tuning Tips, if you are not sure about which configuration is better for controlling your process.

4.5. Additional Output Parameters for Auto-Tuning

This section presents only those output parameters related specifically to auto-tuning. See the chapter PIDA_REAL for the additional output parameters.

4.5.1. AutoTuneDone

- Type: BOOL

This output parameter indicates that auto-tuning function has ended and the results were computed in the output parameter AutoTuneResult. It can be ended with or without errors (see output parameter AutoTuneError).

4.5.2. AutoTuneError

- Type: BOOL

This output parameter indicates that auto-tuning function has ended (or aborted) due to an error detected by the function. In this case, the output parameter AutoTuneResult gives more information about the detected error(s).

4.5.3. AutoTuneResult

- Type: AUTOTUNE_RESULT

This output parameter is a structure that contains the calculated tuning parameters and error indications. The following figure shows the definition of this data structure:

```
TYPE AUTOTUNE_RESULT :
STRUCT
    Kp : REAL;                  // Kp calculated by auto-tunning
    Ti_Ki : REAL;               // Ti_Ki calculated by auto-tunning
    Td_Kd : REAL;               // Td_Kd calculated by auto-tunning
    Bad_PercAmpMV : BIT;        // Parameter AutoTuneParam.PercAmpMV out of range
    Bad_PercHystPV : BIT;       // Parameter AutoTuneParam.PercHystPV out of range
    Bad_PercMaxPeakPV : BIT;    // Parameter AutoTuneParam.PercMaxPeakPV out of range
    Bad_NumCycles : BIT;        // Parameter AutoTuneParam.NumCycles out of range
    Bad_Mode : BIT;             // Auto-tuning mode not suported
    MV_TOO_HIGH : BIT;          // Current value of MV is too high for auto-tunning
    MV_TOO_LOW : BIT;           // Current value of MV is too low for auto-tunning
    PV_TOO_HIGH : BIT;          // Current value of PV is too high for auto-tunning
    PV_TOO_LOW : BIT;           // Current value of PV is too low for auto-tunning
    PEAK_TOO_HIGH : BIT;        // Peak of PV during auto-tuning was too high - aborted for safety
    PEAK_TOO_LOW : BIT;         // Peak of PV during auto-tuning was too low
    PV_NOT_STEADY : BIT;        // PV not in steady state or big noise during initialization
    CYCLE_TOO_FAST : BIT;       // Cycle period too fast compared with PLC sample time
    HIGH_DEAD_TIME : BIT;       // Relation between dead time and time constant too high
END_STRUCT
END_TYPE
```

Figure 17: Struct AUTOTUNE_RESULT

4.5.3.1. Kp

- Type: REAL

This is the proportional gain Kp calculated by auto-tuning.

4.5.3.2. Ti_Ki

- Type: REAL

This is the integral time or integral gain Ti_Ki calculated by the auto-tuning. If the input parameter Indep is TRUE during auto-tuning, a gain is calculated; otherwise a time is calculated.

4.5.3.3. Td_Kd

- Type: REAL

This is the derivative time or derivative gain Td_Kd calculated by the auto-tuning. If the input parameter Indep is TRUE during auto-tuning, a gain is calculated; otherwise a time is calculated.

4.5.3.4. Bad_PercAmpMV

- Type: BIT(BOOL)

This bit returns TRUE if the input parameter AutoTuneParam.PercAmpMV is out of the allowed range.

4.5.3.5. Bad_PercHystPV

- Type: BIT (BOOL)

This bit returns TRUE if the input parameter AutoTuneParam.PercHystPV is out of the allowed range.

4.5.3.6. Bad_PercMaxPeakPV

- Type: BIT (BOOL)

This bit returns TRUE if the input parameter AutoTuneParam.PercMaxPeakPV is out of the allowed range.

4.5.3.7. Bad_NumCycles

- Type: BIT (BOOL)

This bit returns TRUE if the input parameter AutoTuneParam.NumCycles is out of the allowed range.

4.5.3.8. Bad_Mode

- Type: BIT (BOOL)

This bit returns TRUE if the input parameter AutoTuneParam.Mode is out of the allowed range.

4.5.3.9. MV_TOO_HIGH

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if (RefMV + M) is bigger than MaxMV.

4.5.3.10. MV_TOO_LOW

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if (RefMV - M) is smaller than MinMV.

4.5.3.11. PV_TOO_HIGH

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if (RefPV + H) is bigger than MaxPV.

4.5.3.12. PV_TOO_LOW

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if (RefPV - H) is smaller than MinPV.

4.5.3.13. PEAK_TOO_HIGH

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if PV gets outside the range (RefPV ± S) during auto-tuning measurement cycles. The function is aborted for safety, and MV returns to RefMV.

4.5.3.14. PEAK_TOO_LOW

- Type: BIT (BOOL)

This bit returns TRUE if the PV peaks during measurement cycles are too low compared with hysteresis H (see Figure 14).

In this case, the calculated tuning parameters cannot be trusted.

This can happen, for instance, when the dead time is very low compared to the time constant of the process, or when the static gain of the process is too low. Sometimes it is possible to avoid this error by increasing the value of the AutoTuneParam.PercAmpMV parameter or by decreasing the value of the AutoTuneParam.PercHystPV parameter.

4.5.3.15. PV_NOT_STEADY

- Type: BIT (BOOL)

See Figure 14. This bit returns TRUE if PV gets outside the range (RefPV ± H) in the first 3 seconds after auto-tuning was started. During this interval, the function checks if PV is steady. It is possible to avoid this error by increasing the value of the AutoTuneParam.PercHystPV parameter.

4.5.3.16. CYCLE_TOO_FAST

- Type: BIT (BOOL)

This bit returns TRUE if the duration of a measurement cycle is too small compared with the PLC cycle time (output parameter SampleTime).

In this case, the calculated tuning parameters cannot be trusted.

This normally happens when the dead time is very low compared to the time constant of the process.

4.5.3.17. HIGH_DEAD_TIME

- Type: BIT (BOOL)

This bit returns TRUE if the duration of a measurement cycle is too small compared with the measured dead time.

In this case, the calculated tuning parameters cannot be trusted.

This can happen when the dead time is much higher compared to the time constant of the process.

4.6. Recommended SCADA Interface and Code for Copying the Calculated Tuning Parameters

After auto-tuning ends, the calculated tuning parameters are not copied automatically over the active tuning parameters of PIDA_TUNE_REAL.

The following method is recommended for copying the tuning parameters:

- Create a command like CopyParam (boolean variable) that must be set to TRUE by the user (e.g.: in SCADA system) to start copying the calculated tuning parameters.
- Use a piece of code similar to that shown in the following figure (in this example, ST language).

```
IF CopyParam THEN
    CopyParam := FALSE;
    AutoTune := FALSE;
    IF AutoTuneDone AND NOT(AutoTuneError) THEN
        Kp := AutoTuneResult.Kp;
        Ti_Ki := AutoTuneResult.Ti_Ki;
        Td_Kd := AutoTuneResult.Td_Kd;
    END_IF
END_IF
```

Figure 18: Code for copying tuning parameters

ATTENTION

For more details, see section PID Simulation with Auto-Tuning inside the Application Notes chapter.

5. PIDA_TUNE_INT

PIDA_TUNE_INT works similarly to PIDA_TUNE_REAL, described in the previous chapter. Both FBs are used for PID control with the auto-tuning function. The following figure shows the input and output parameters of this FB.

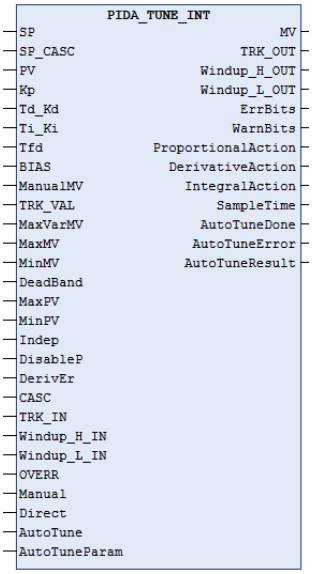

Figure 19: PIDA_TUNE_INT parameters

In PIDA_TUNE_REAL, all numeric parameters have the REAL type. In PIDA_TUNE_INT, most numeric parameters have the INT type and a few of them have the REAL type.

If the user selects PIDA_TUNE_INT instead of PIDA_TUNE_REAL, the following advantages can be achieved:

- Each instance consumes less memory. This may be important in redundant PLC applications with hundreds of PIDA_TUNE_INT instances, for saving memory synchronization time between the redundant PLCs.)
- For some communication protocols like MODBUS TCP, the number of bytes exchanged cyclically with the SCADA system can be reduced significantly, especially if the PLC application has hundreds of PIDA_TUNE_INT instances.
- The application gets easier when the PLC manages field units, and conversions between engineering units and field units are executed by the SCADA system.

In the other hand, PIDA_TUNE_INT should not be used if the PLC must manage engineering units internally.

This chapter will not describe again all the parameters of PIDA_TUNE_INT. Instead, it will only present a list of those parameters which type changed from REAL in PIDA_TUNE_REAL to INT in PIDA_TUNE_INT. The meaning from each of the parameters, if compared between PIDA_TUNE_REAL and PIDA_TUNE_INT is the same. So, to use PIDA_TUNE_INT, the user must read the chapter PIDA_TUNE_REAL, and just consider that some parameters changed their type from REAL to INT.

5.1. Input Parameters with Type INT

The following input parameters have the REAL type in PIDA_TUNE_REAL and the INT type in PIDA_TUNE_INT:

- SP
- SP_CASC
- PV
- BIAS
- ManualMV
- TRK_VAL
- MaxVarMV
- MaxMV
- MinMV
- DeadBand
- MaxPV
- MinPV

5.2. Output Parameters with Type INT

The following output parameters have type REAL in PIDA_REAL and type INT in PIDA_INT:

- MV

6. ControlON_OFF

ControlON_OFF is used for ON-OFF control.

The following figure shows the input and output parameters of this FB.

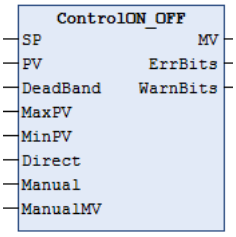

Figure 20: ControlON_OFF parameters

Like the PID controllers described in previous chapters, this FB is also a feedback controller (measures PV, computes the error relative to SP, and tries to eliminate the error using its MV output).

The main difference is that the MV output has the BOOL type instead of REAL or INT, so it can only assume two states (ON = TRUE, OFF = FALSE).

Another difference is that the ControlON_OFF FB is quite simpler than the FBs for PID control.

The working principle is very simple.

With reverse control (Direct = FALSE):

1. The MV output is turned ON when the error (SP – PV) is bigger than the input parameter DeadBand.
2. The MV output is turned OFF when the error (SP – PV) is smaller than the negative of the input parameter DeadBand.
3. The MV output keeps its previous value if the conditions 1 and 2 are false.

With direct control (Direct = TRUE):

1. The MV output is turned OFF when the error (SP – PV) is bigger than the input parameter DeadBand.
2. The MV output is turned ON when the error (SP – PV) is smaller than the negative of the input parameter DeadBand.
3. The MV output keeps its previous value if conditions 1 and 2 are false.

ON-OFF controllers are typically used when:

- The tolerated steady state error is relatively big. The maximum tolerated error is configured in the parameter DeadBand.
- When the time constant of the process is high and the dead time of the process is low, thus making the process easy to control.

The next figure shows an example of control with Direct = FALSE, DeadBand = 3 and SP = 20.

At the left side cursor, MV turns on when PV becomes smaller than (SP - DeadBand). At the right side cursor, MV turns off when PV becomes bigger than (SP + DeadBand).

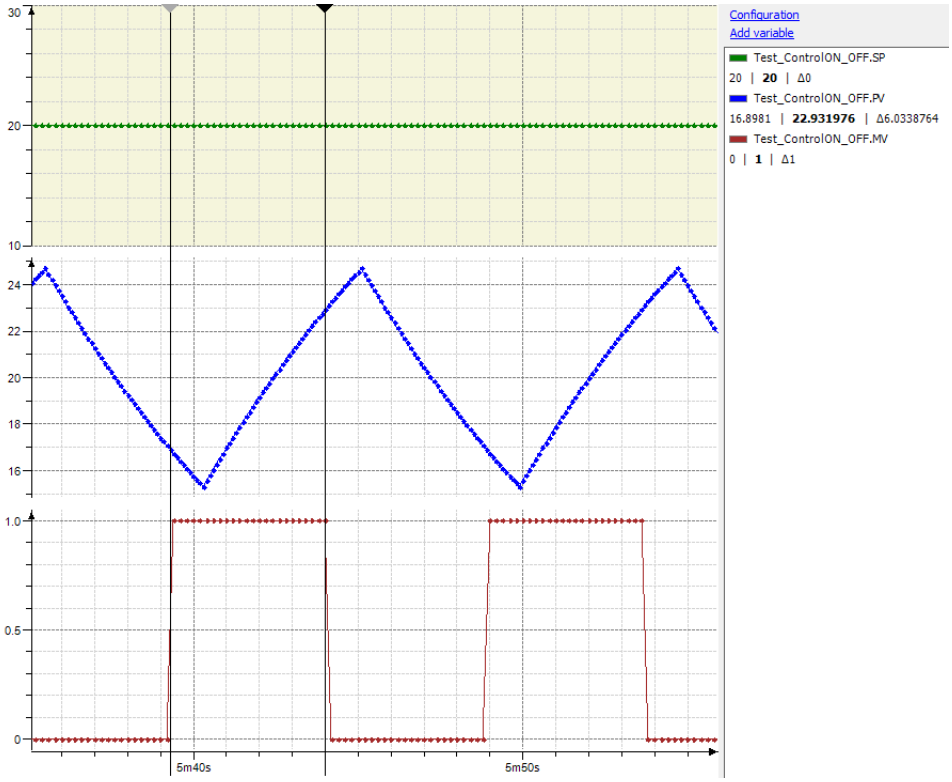

Figure 21: Example of ON OFF control

6.1. Input Parameters

The following subsections describe the input parameters.

6.1.1. SP - Setpoint

- Type: REAL
- Range: see parameters MaxPV – Maximum Value of Process Variable and MinPV – Minimum Value of Process Variable

SP informs the target value for PV (process variable).

Error is computed as:

$$Error = SP - PV \tag{9}$$

The main objective of ControlON_OFF is to keep the absolute value from the error smaller than the input parameter DeadBand.

6.1.2. PV - Process Variable

- Type: REAL
- Range: see parameters MaxPV – Maximum Value of Process Variable and MinPV – Minimum Value of Process Variable

PV corresponds to the process variable controlled by ControlON_OFF FB. It is normally read from a transmitter, for instance, using an analog input.

6.1.3. DeadBand - Dead Band for Error

- Type: REAL
- Range: must be bigger than 0

This input parameter defines a minimum absolute value for the error to cause a control action.

If the absolute value from the error (SP – PV) is smaller than the DeadBand, the ControlON_OFF FB assumes as if the error were zero, and does not change the output (MV).

If the error is bigger than the DeadBand:

- MV is turned ON, with Direct = FALSE
- MV is turned OFF, with Direct = TRUE

If the error is smaller than the negative of DeadBand:

- MV is turned OFF, with Direct = FALSE
- MV is turned ON, with Direct = TRUE

6.1.4. MaxPV – Maximum Value of Process Variable

- Type: REAL
- Range: must be bigger than MinPV

This input parameter imposes a maximum limit to the input parameters PV and SP. If PV or SP is bigger than MaxPV, then the value of MaxPV is assigned to PV or SP for internal usage in the FB.

6.1.5. MinPV – Minimum Value of Process Variable

- Type: REAL
- Range: must be smaller than MaxPV

This input parameter imposes a minimum limit to the input parameters PV and SP. If PV or SP is smaller than MinPV, then the value of MinPV is assigned to PV or SP for internal usage in the FB.

6.1.6. Direct – Direct Control

- Type: BOOL

ControlON_OFF can be configured for direct control (Direct = TRUE) or reverse control (Direct = FALSE). Some controlled processes require a direct control while others required a reverse control.

If this parameter is not configured correctly, a positive feedback control will be established, and ControlON_OFF will not be able to control the process.

The following rules must be used to get a correct configuration with negative feedback:

- If MV must be turned ON to control the process when PV increases, then direct control must be selected.
- If MV must be turned OFF to control the process when PV increases, then reverse control must be selected.

6.1.7. Manual – Manual Mode

- Type: BOOL

When TRUE, it indicates that the ControlON_OFF FB is in manual mode, and MV is copied from the input parameter ManualMV. See the description from the input parameter ManualMV – Output in Manual Mode.

6.1.8. ManualMV – Output in Manual Mode

- Type: BOOL

This input parameter is copied to output MV when the manual mode is selected (see input parameter Manual – Manual Mode).

In manual mode, the ControlON_OFF FB does not try to make PV approach to SP. Operator or another system is responsible for control, in manual mode.

6.2. Output Parameters

The following subsections describe the output parameters.

6.2.1. MV – Manipulated Variable

- Type: BOOL

MV corresponds to the manipulated variable for controlling the process. It is normally written to a digital actuator, for instance, using a digital output.

The objective of a ControlON_OFF FB is to keep PV close to SP, by controlling the MV output.

6.2.2. ErrBits – Fatal Error Bits

- Type: ON_OFF_ERROR_BITS

This output parameter is a structure of bits that indicates fatal errors that abort the execution of the ControlON_OFF FB.

- MaxMinPV bit is set when MaxPV <= MinPV;
- DeadBand bit is set when DeadBand <= 0.

The following figure details the struct type ON_OFF_ERROR_BITS.

```
TYPE ON_OFF_ERROR_BITS :
STRUCT
    MaxMinPV : BIT;          // MaxPV <= MinPV
    DeadBand : BIT;          // DeadBand <= 0
END_STRUCT
END_TYPE
```

Figure 22: Struct ON_OFF_ERROR_BITS

6.2.3. WarnBits – Warning Bits

- Type: ON_OFF_WARNING_BITS

This output parameter is a structure of bits that indicate warnings about the execution of the ControlON_OFF FB.

- PV_MaxPV bit is set when PV is being limited by MaxPV;
- PV_MinPV bit is set when PV is being limited by MinPV;
- SP_MaxPV bit is set when SP is being limited by MaxPV;
- SP_MinPV bit is set when SP is being limited by MinPV.

The following figure details the struct type ON_OFF_WARNING_BITS.

```
TYPE ON_OFF_WARNING_BITS :
STRUCT
    PV_MaxPV : BIT;          // PV is being limited by MaxPV
    PV_MinPV : BIT;          // PV is being limited by MinPV
    SP_MaxPV : BIT;          // SP is being limited by MaxPV
    SP_MinPV : BIT;          // SP is being limited by MinPV
END_STRUCT
END_TYPE
```

Figure 23: Struct ON_OFF_WARNING_BITS

7. ControlPWM

The ControlPWM FB is used to convert a typical analog output (MV, type REAL) to a pulse width modulated output (PWM, type BOOL).

The following figure shows the input and output parameters of this FB.

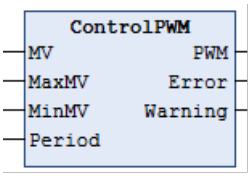

Figure 24: ControlPWM parameters

The MV range is defined by parameters MinMV and MaxMV, and the Period of the PWM output is an integer number of cycles of cyclic task where the FB is called. By the way, FB must be called in a cyclic task for working properly.

The following figure shows an example with Period equal 10 cycles of 100 ms, MinMV = 0 and MaxMV = 100.

- With MV = 36, PWM output stays 4 cycles ON and 6 cycles OFF (10 * (36 / (100 - 0)) = 3.6 (rounded up to 4).
- With MV = 34, PWM output stays 3 cycles ON and 7 cycles OFF ((34 / (100 - 0) * 10) = 3.4 (rounded down to 3).

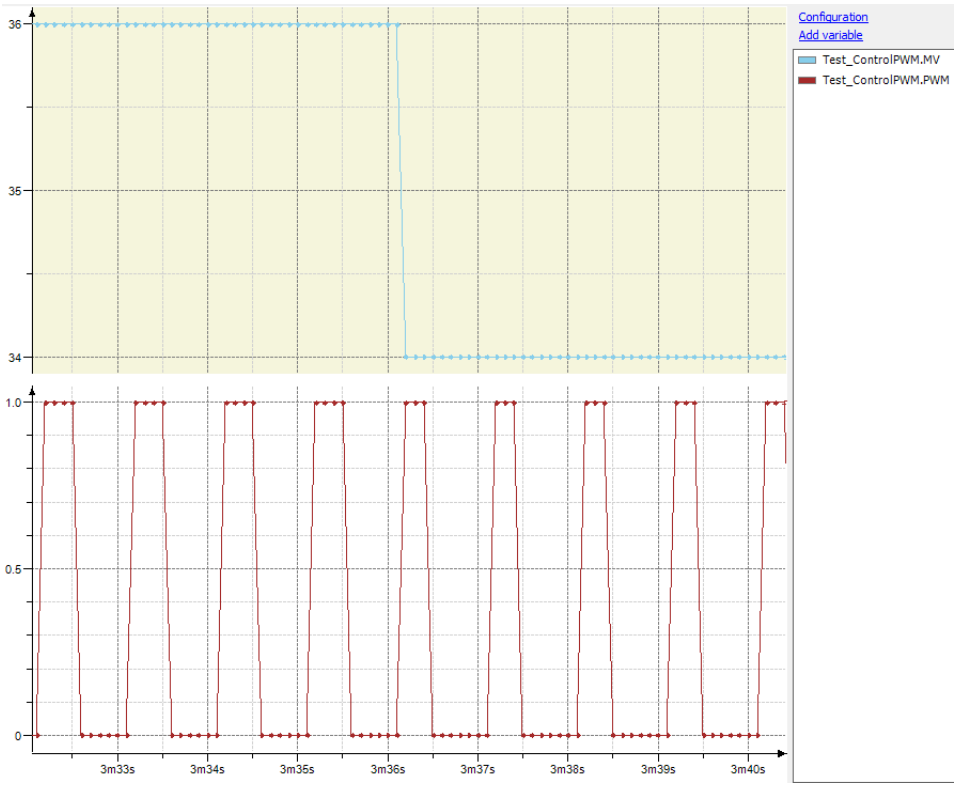

Figure 25: Example of ControlPWM conversion

7.1. Input Parameters

7.1.1. MV – Manipulated Variable

- Type: REAL
- Range: see parameters MaxMV – Maximum Value of MV and MinMV – Minimum Value of MV

MV is the REAL input variable that will be converted to a pulse width modulation output.

7.1.2. MaxMV – Maximum Value of MV

- Type: REAL
- Range: must be bigger than MinMV

This input parameter imposes a maximum limit to input parameter MV, and defines the range of MV together with parameter MinMV.

7.1.3. MinMV – Minimum Value of MV

- Type: REAL
- Range: must be smaller than MaxMV

This input parameter imposes a minimum limit to input parameter MV, and defines the range of MV together with parameter MaxMV.

7.1.4. Period – Period of PWM Output

- Type: UINT
- Range: 1 . . . 65535

This input parameter defines the period of PWM output as a number of cycles of the cyclic task where the FB is called.

For instance, if Period = 10, and the interval of the cyclic task is 100 ms, then the PWM output will have a period equal to one second (10 * 100 ms).

7.2. Output Parameters

7.2.1. PWM – Pulse Width Modulated Output

- Type: BOOL

PWM is the pulse width modulated output.

7.2.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is true if a fatal error is aborting the FB execution. It is set when any of the following conditions is true:

- MaxMV <= MinMV
- Period = 0

7.2.3. Warning – Warning Indication

- Type: BOOL

This output parameter is true when MV is being limited by MinMV or MaxMV.

8. ControlLowFilter

This FB implements a first order filter with constant time Tc. It can be used, for instance, to filter noise over PV input before connecting it to a PID control FB.

The following figure shows the input and output parameters of this FB.

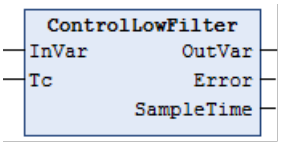

Figure 26: ControlLowFilter parameters

Note that filtering a signal helps eliminating the noise, but also causes some lag between unfiltered signal and filtered signal.

The following figure shows an example where this FB received as input an unfiltered signal (PV_Noisy) and produced as output a filtered signal (PV_Filtered). In this example a triangular noise with 5 Hz was added to the unfiltered signal, and a filter with Tc = 0.5 seconds was employed, with sample time of 0.1 seconds.

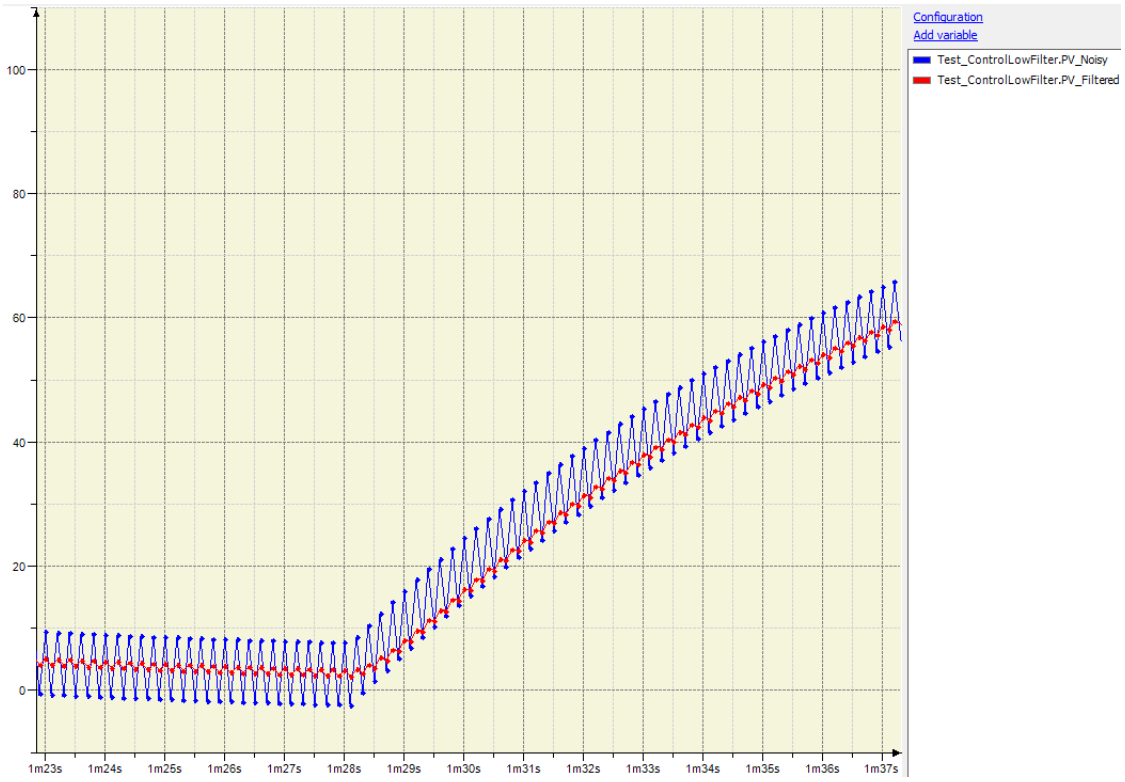

Figure 27: Example of filtering with ControlLowFilter

8.1. Input Parameters

8.1.1. InVar – Input Variable

- Type: REAL

This is the input variable to be filtered.

8.1.2. Tc – Time Constant

- Type: REAL
- Unit: seconds
- Range: > 0

This parameter is the time constant of the first order filter.

8.2. Output Parameters

8.2.1. OutVar – Output Variable

- Type: REAL

This is the first order filtered copy of input variable InVar.

8.2.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FB. It is set when one of the following errors occurs:

- The FB is being called inside a non-cyclic task
- SampleTime < 0.001 seconds
- SampleTime > 60 seconds
- Tc <= 0

8.2.3. SampleTime – Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

9. ControlDelay

ControlDelay FB creates a delay, specified in seconds, for an input variable of type REAL.

The FB must be called in a cyclic task.

The delay is limited to MaxSamplesDelay sample times of this FB, due to memory allocation limits. MaxSamplesDelay is a parameter that can be adjusted in the NextoPID library.

The following figure shows the input and output parameters of this FB.

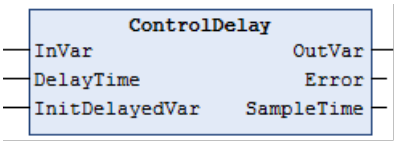

Figure 28: ControlDelay parameters

This FB can be useful for process simulation and for feed-forward controllers (see section Feed-Forward inside chapter Application Notes).

The following figure shows an example where the input variable (InVar) is delayed producing the output variable (OutVar).

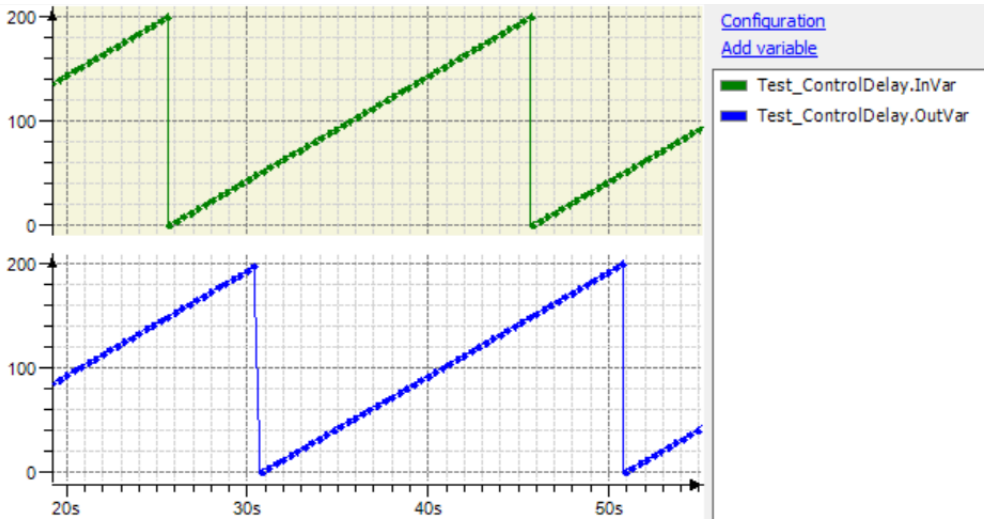

Figure 29: Example of delay created by ControlDelay

9.1. Adjusting the Parameter MaxSamplesDelay

This parameter is a constant which default value is 100 when the NextoPID library is installed in Library Manager. The maximum delay provided by FB ControlDelay is MaxSamplesDelay multiplied by the sample time. Sample time is the interval of the cyclic task where the FB is called (see output parameter SampleTime – Cycle Used for Calling the FB).

The user can change the value of this parameter according to his needs. The bigger is the parameter, more memory will be allocated for each instance of FB ControlDelay. So, avoid to configure unnecessarily big values.

For changing the parameter, open the Library Manager, select the NextoPID library, open the "Auxiliary Control Blocks" folder, and select "Delay_Parameters". In the right pane, select tab "Library Parameters" and edit the value of MaxSamples-Delay.

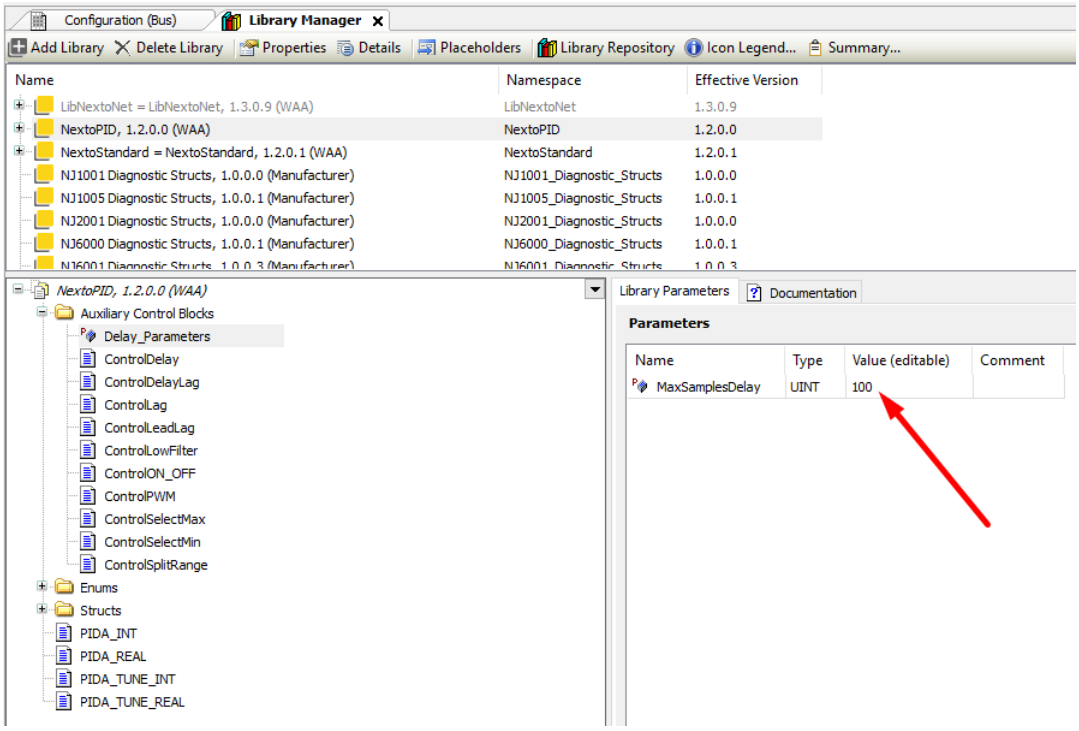

Figure 30: Adjusting parameter MaxSamplesDelay

9.2. Input Parameters

9.2.1. InVar – Input Variable

- Type: REAL

This is the input variable where delay will be applied.

9.2.2. DelayTime – Delay Time

- Type: REAL
- Unit: seconds
- Range: 0 to MaxSamplesDelay * SampleTime

This is the applied delay. It must not exceed MaxSamplesDelay * SampleTime (see output parameter SampleTime – Cycle Used for Calling the FB).

In case of change of DelayTime, OutVar will stay unchanged during DelayTime seconds. After this, it will again receive the delayed copy of Invar.

9.2.3. InitDelayedVar – Initial Value of Delayed Variable

- Type: REAL

During the first DelayTime seconds after initialization, the delayed output of the ControlDelay FB is unknown.

During this initialization interval, the delayed output of the ControlDelay FB will be InitDelayedVar.

The user normally should assign a constant value like zero to InitDelayedVar.

9.3. Output Parameters

9.3.1. OutVar – Output Variable

- Type: REAL

This is the output variable with delay applied.

9.3.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FB. It is set when one of the following errors occurs:

- The FB is being called inside a non-cyclic task
- SampleTime < 0.001 seconds
- SampleTime > 60 seconds
- DelayTime < 0
- DelayTime > MaxSamplesDelay* SampleTime

9.3.3. SampleTime – Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

10. ControlLag

ControlLag FB applies a first order lag over an input. It can be used, for instance, to simulate a first order process without dead time (for simulating a first order process with dead time, see FB ControlDelayLag).

The following figure shows the input and output parameters of this FB.

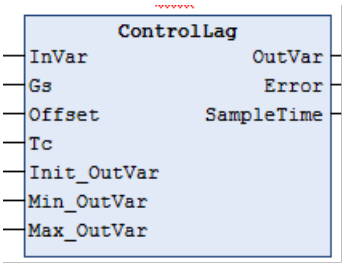

Figure 31: ControlLag parameters

It is possible to configure time constant, gain, offset, initial value, maximum value and minimum value for calculating the output.

The following figure shows an example of step response, with Tc = 10 seconds, Gs = 2 and Offset = 20.

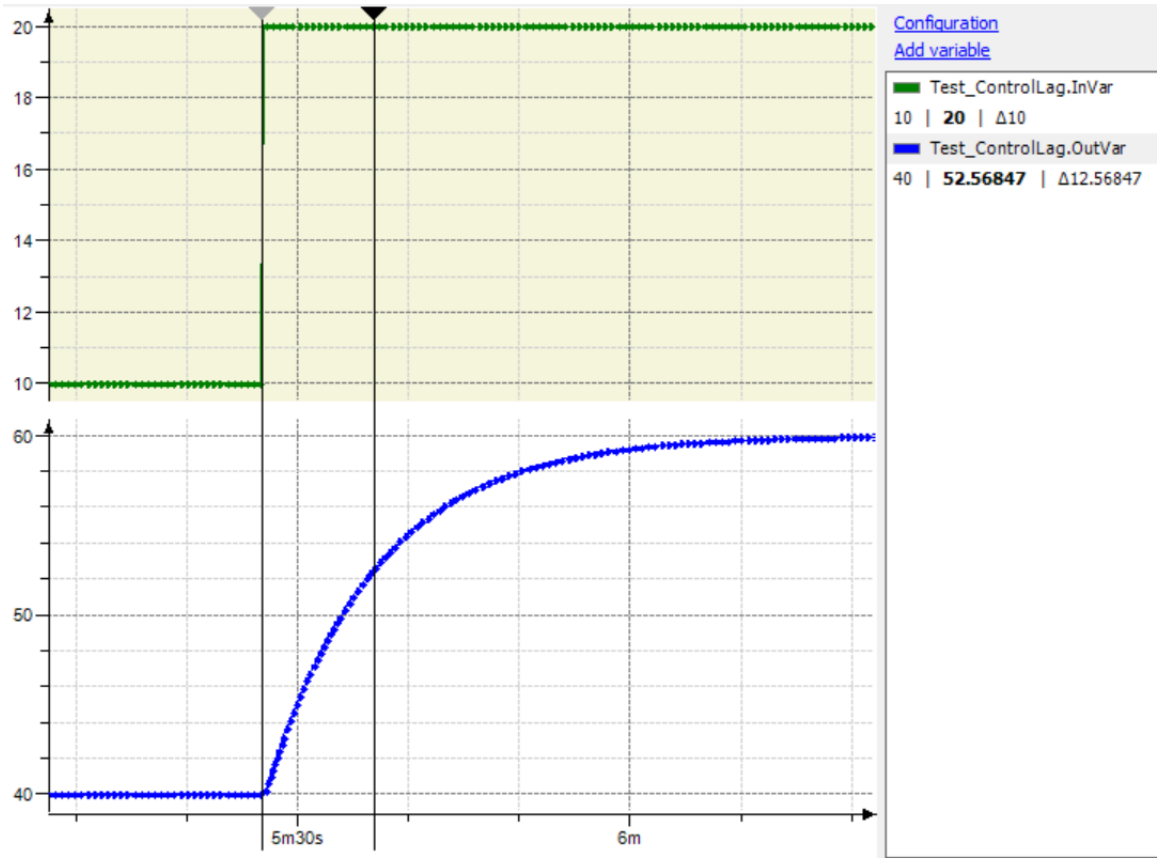

Figure 32: Example of lag created by ControlLag

10.1. Input Parameters

10.1.1. InVar – Input Variable

- Type: REAL

This is the input variable where lag will be applied.

10.1.2. Gs – Gain

- Type: REAL

This is the gain used to calculate the steady state value of the output.
In a specific moment, the steady state value of output is:

$$Gs * InVar + Offset \tag{10}$$

Notes:

- A process simulated with Gs > 0 must be controlled by a PID controller with reverse control.
- A process simulated with Gs < 0 must be controlled by a PID controller with direct control.

10.1.3. Offset – Offset Added to Output

- Type: REAL

This is an offset used to calculate the steady state value of the output.
In a specific moment, the steady state value of output is:

$$Gs * InVar + Offset \tag{11}$$

10.1.4. Tc – Time Constant

- Type: REAL
- Unit: seconds
- Range: > 0

This parameter is the time constant of the first order lag.

10.1.5. Init_OutVar – Initial Value of Output

- Type: REAL

This is the initial value for the output.

10.1.6. Min_OutVar – Minimum Value of Output

- Type: REAL

This is a minimum limit for the output value. If the calculated output value gets smaller than this limit, Min_OutVar is copied to output value.

10.1.7. Max_OutVar – Maximum Value of Output

- Type: REAL

This is a maximum limit for the output value. If the calculated output value gets bigger than this limit, Max_OutVar is copied to output value.

10.2. Output Parameters

10.2.1. OutVar – Output Variable

- Type: REAL

This is the output variable with lag applied.

10.2.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FB. It is set when one of the following errors occurs:

- The FB is being called inside a non-cyclic task
- SampleTime < 0.001 seconds
- SampleTime > 60 seconds
- Tc <= 0
- Min_OutVar > Max_OutVar
- Init_OutVar < Min_OutVar
- Init_OutVar > Max_OutVar

10.2.3. SampleTime – Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

11. ControlDelayLag

ControlDelayLag FB combines one ControlDelay FB with one ControlLag FB. It can be used, for instance, to simulate a first order process with dead time. The input variable is first delayed and then a lag is applied over the delayed signal.

The FB must be called in a cyclic task.

The following figure shows the input and output parameters of this FB.

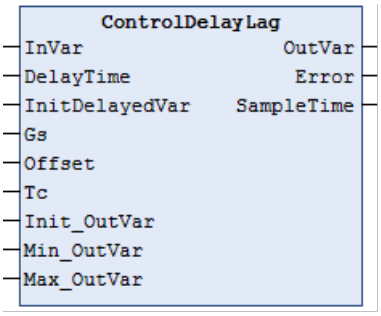

Figure 33: ControlDelayLag parameters

It is possible to configure delay, time constant, gain, offset, initial value, maximum value and minimum value for calculating the output.

The following figure shows an example of step response, with DelayTime = 2 seconds, Tc = 10 seconds, Gs = 2 and Offset = 20.

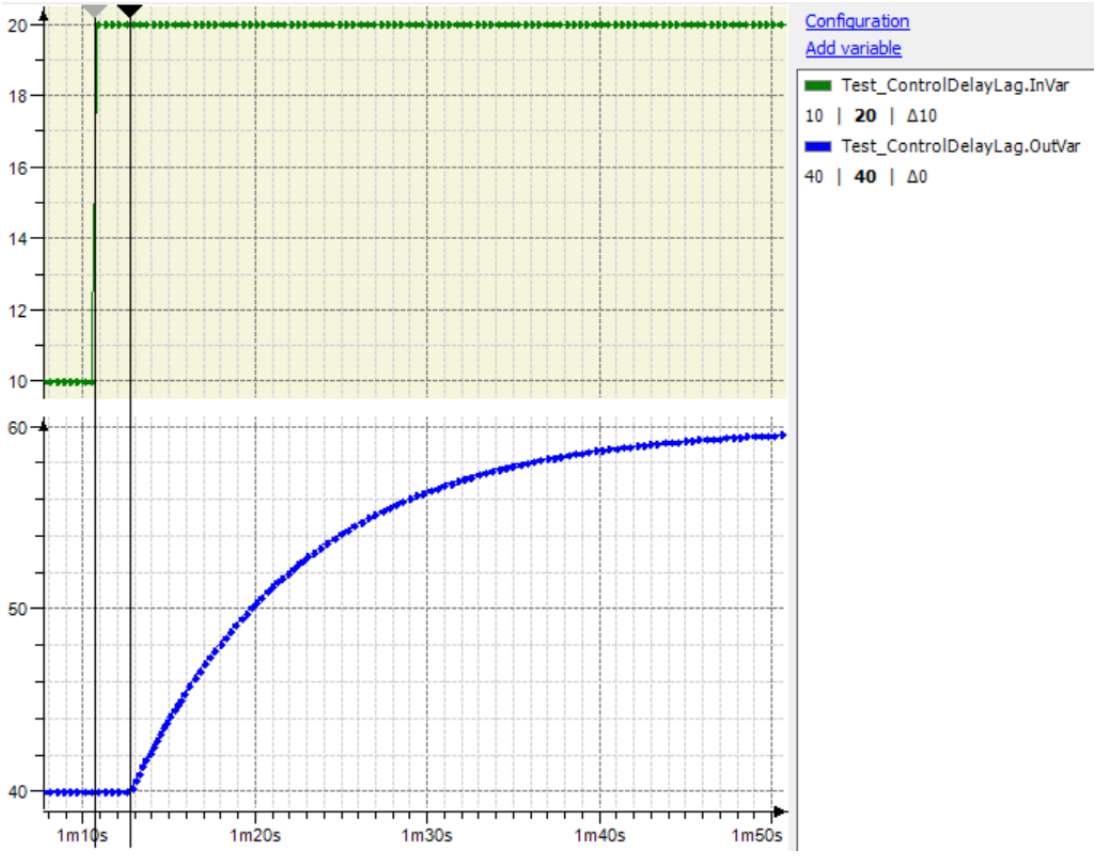

Figure 34: Example of delay plus lag created by ControlDelayLag

11.1. Adjusting the Maximum Delay

Note that ControlDelayLag is a combination between the previously described FBs ControlDelay and ControlLag.

The maximum delay provided by ControlDelay and ControlDelayLag depends on parameter MaxSamplesDelay. For adjusting this parameter, read section Adjusting the Parameter MaxSamplesDelay, inside the section ControlDelay.

11.2. Input Parameters

11.2.1. InVar – Input Variable

- Type: REAL

This is the input variable where delay and lag will be applied.

11.2.2. DelayTime – Delay Time

- Type: REAL
- Unit: seconds
- Range: 0 to MaxSamplesDelay * SampleTime

This is the applied delay. It must not exceed MaxSamplesDelay * SampleTime (see output parameter SampleTime – Cycle Used for Calling the FB).

In case of change of DelayTime, the delayed input signal (delayed InVar) will stay unchanged during DelayTime seconds.

11.2.3. InitDelayedVar – Initial Value of Delayed Variable

- Type: REAL

During the first DelayTime seconds after initialization, the delayed input signal (delayed InVar) is unknown and assumes the value InitDelayedVar.

The user normally should assign a constant value like zero to InitDelayedVar.

11.2.4. Gs – Gain

- Type: REAL

This is the gain used to calculate the steady state value of the output. In a specific moment, the steady state value of output is:

$$Gs * delayed(InVar) + Offset \tag{12}$$

Notes:

- A process simulated with Gs > 0 must be controlled by a PID controller with reverse control.
- A process simulated with Gs < 0 must be controlled by a PID controller with direct control.

11.2.5. Offset – Offset Added to Output

- Type: REAL

This is an offset used to calculate the steady state value of the output.

In a specific moment, the steady state value of output is:

$$Gs * delayed(InVar) + Offset \tag{13}$$

11.2.6. Tc – Time Constant

- Type: REAL
- Unit: seconds
- Range: > 0

This parameter is the time constant of the first order lag.

11.2.7. Init_OutVar – Initial Value of Output

- Type: REAL

This is the initial value for the output.

11.2.8. Min_OutVar – Minimum Value of Output

- Type: REAL

This is a minimum limit for the output value. If the calculated output value gets smaller than this limit, Min_OutVar is copied to output value.

11.2.9. Max_OutVar – Maximum Value of Output

- Type: REAL

This is a maximum limit for the output value. If the calculated output value gets bigger than this limit, Max_OutVar is copied to output value.

11.3. Output Parameters

11.3.1. OutVar – Output Variable

- Type: REAL

This is the output variable with delay and lag applied.

11.3.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FB. It is set when one of the following errors occurs:

- The FB is being called inside a non-cyclic task
- SampleTime < 0.001 seconds
- SampleTime > 60 seconds
- DelayTime < 0
- DelayTime > MaxSamplesDelay* SampleTime
- Tc <= 0
- Min_OutVar > Max_OutVar
- Init_OutVar < Min_OutVar
- Init_OutVar > Max_OutVar

11.3.3. SampleTime – Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

12. ControlLeadLag

The ControlLeadLag FB can be used as a component for some advanced control strategies, like feed-forward (see section Feed-Forward inside chapter Application Notes).

The following figure shows the input and output parameters of this FB.

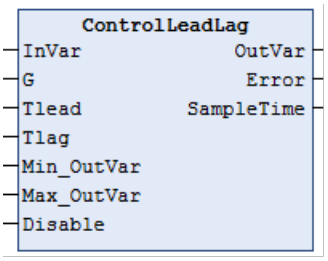

Figure 35: ControlLeadLag parameters

The output parameter OutVar is calculated applying a lead time (Tlead) and a lag time (Tlag), and gain G, over the input parameter InVar.

Next figure shows an example of step response with Tlead = 4 seconds, Tlag = 2 seconds and G = 2.

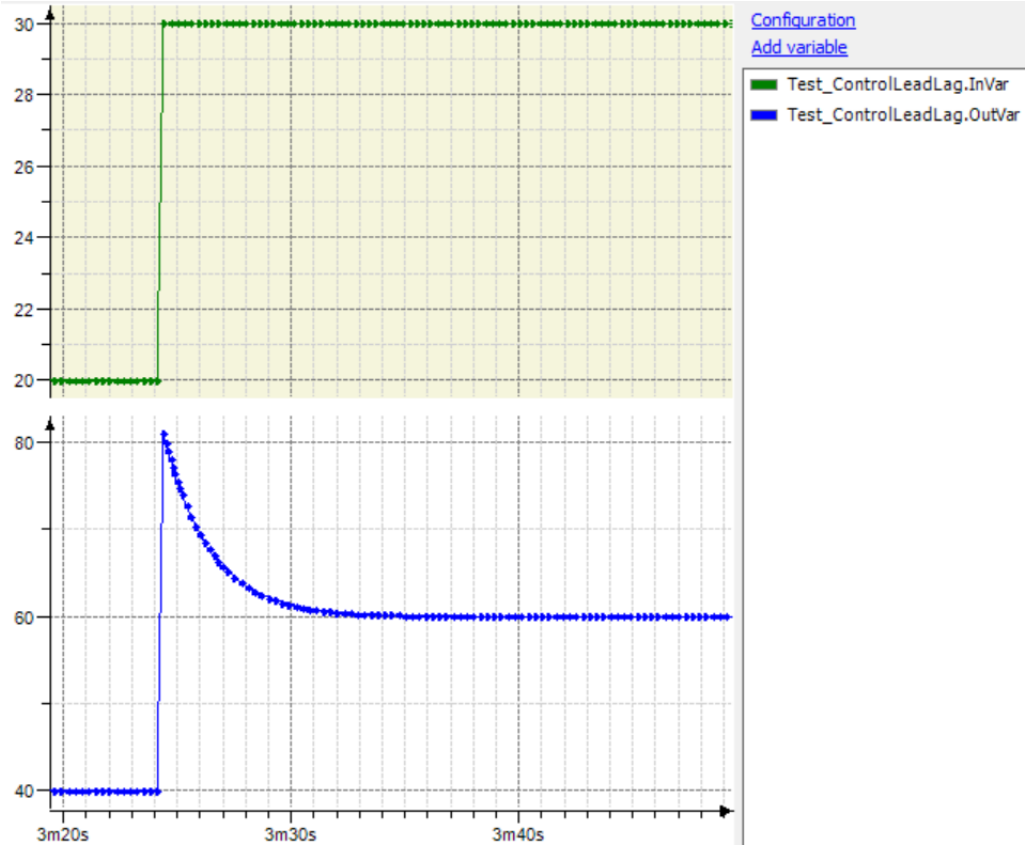

Figure 36: Example of ControlLeadLag with Tlead = 4s and Tlag = 2s

Next figure shows an example of step response with Tlead = 2 seconds, Tlag = 4 seconds and G = 2.

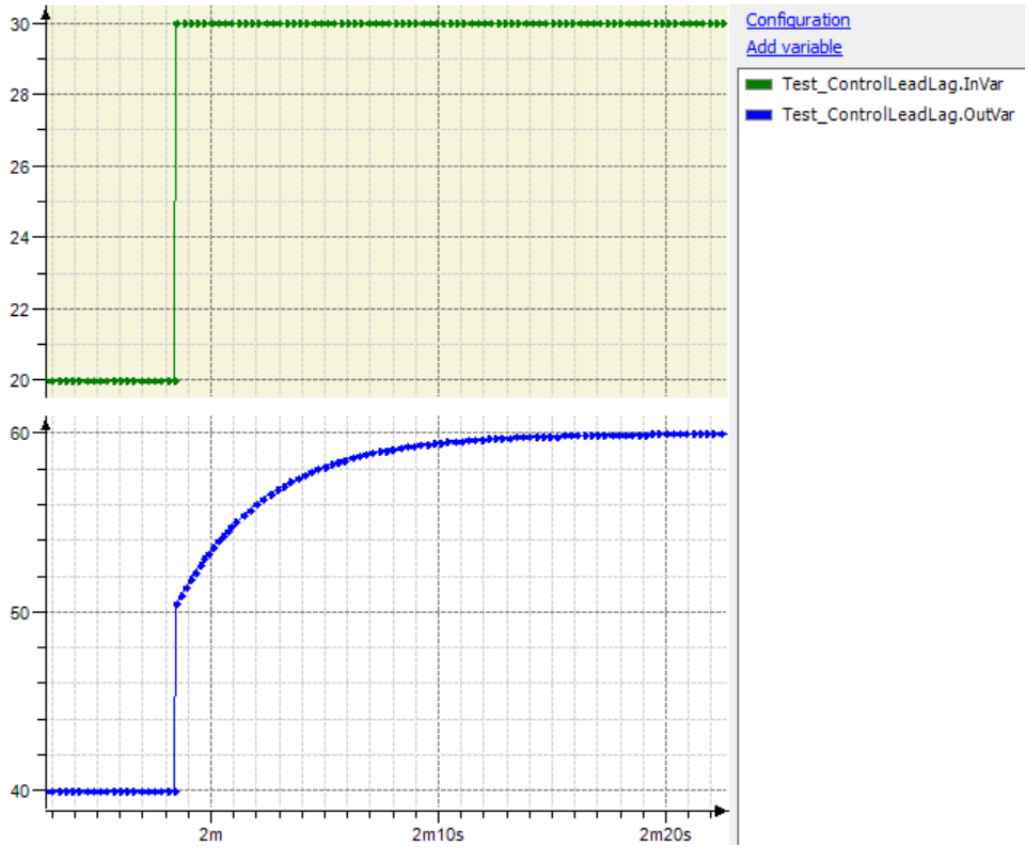

Figure 37: Example of ControlLeadLag with Tlead = 2s and Tlag = 4s

12.1. Input Parameters

12.1.1. InVar – Input Variable

- Type: REAL

This is the input variable where the lead/lag processing will be applied to produce output OutVar.

12.1.2. G – Gain

- Type: REAL

This is the gain. The steady state of output OutVar is equal to G multiplied by input InVar.

12.1.3. Tlead – Lead Time

- Type: REAL
- Unit: seconds
- Range: >= 0

This is the lead time in seconds.

12.1.4. Tlag – Lag Time

- Type: REAL
- Unit: seconds
- Range: > 0

This is the lag time in seconds.

12.1.5. Min_OutVar – Minimum Value of Output

- Type: REAL

This is a minimum limit for the output OutVar. If the calculated output gets smaller than this limit, Min_OutVar is copied to output OutVar.

12.1.6. Max_OutVar – Maximum Value of Output

- Type: REAL

This is a maximum limit for the output OutVar. If the calculated output gets bigger than this limit, Max_OutVar is copied to output OutVar.

12.1.7. Disable – Disable Output

- Type: BOOL

When this parameter is TRUE, output OutVar is cleared to zero.

12.2. Output Parameters

12.2.1. OutVar – Output Variable

- Type: REAL

This is the result of lead/lad processing over input InVar.

12.2.2. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FB. It is set when one of the following errors occurs:

- The FB is being called inside a non-cyclic task
- SampleTime < 0.001 seconds
- SampleTime > 60 seconds
- Tlead < 0
- Tlag <= 0

12.2.3. SampleTime – Cycle Used for Calling the FB

- Type: REAL
- Unit: seconds

The FB calculates the SampleTime automatically by reading the task interval, and shows its value in this output variable just for information.

13. ControlSelectMax

This simple FUN is intended to be used together with two PID control FBs (PIDA_REAL, PIDA_INT, PIDA_TUNE_REAL, PIDA_TUNE_INT) in override configuration, where the maximum output of two PID controls (MV1 and MV2) must be selected.

ATTENTION

See section Override Configurations inside chapter Application Notes.
This FUN is very easy to implement. If the override configuration involves more than two PID controls, a variation of this FUN can be easily developed.

The following figure shows the input and output parameters of this FUN.

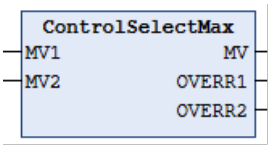

Figure 38: ControlSelectMax parameters

The output MV selects the biggest value among MV1 and MV2. OVERR1 and OVERR2 indicates which output is overridden (not selected).

13.1. Input Parameters

13.1.1. MV1 – MV from First PID Control FB

- Type: REAL

This input variable corresponds to MV output of first PID control FB used in override configuration.

13.1.2. MV2 – MV from Second PID Control FB

- Type: REAL

This input variable corresponds to MV output of second PID control FB used in override configuration.

13.2. Output Parameters

13.2.1. MV – Selected MV

- Type: REAL

This output is the selected MV, that is, the maximum value between inputs MV1 and MV2.

13.2.2. OVERR1 – MV1 is Overridden

- Type: BOOL

This output indicates that MV1 is not selected for output MV. This output must be connected to input parameter OVERR of first PID control FB.

13.2.3. OVERR2 – MV2 is Overridden

- Type: BOOL

This output indicates that MV2 is not selected for output MV. This output must be connected to input parameter OVERR of second PID control FB.

14. ControlSelectMin

This simple FUN is intended to be used together with two PID control FBs (PIDA_REAL, PIDA_INT, PIDA_TUNE_REAL, PIDA_TUNE_INT) in override configuration, where the minimum output of two PID controls (MV1 and MV2) must be selected.

ATTENTION

See section Override Configurations inside chapter Application Notes.
This FUN is very easy to implement. If the override configuration involves more than two PID controls, a variation of this FUN can be easily developed.

The following figure shows the input and output parameters of this FUN.

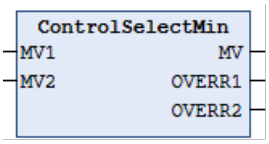

Figure 39: ControlSelectMin parameters

The output MV selects the smallest value among MV1 and MV2. OVERR1 and OVERR2 indicates which output is overridden (not selected).

14.1. Input Parameters

14.1.1. MV1 – MV from First PID Control FB

- Type: REAL

This input variable corresponds to MV output of first PID control FB used in override configuration.

14.1.2. MV2 – MV from Second PID Control FB

- Type: REAL

This input variable corresponds to MV output of second PID control FB used in override configuration.

14.2. Output Parameters

14.2.1. MV – Selected MV

- Type: REAL

This output is the selected MV, that is, the minimum value between inputs MV1 and MV2.

14.2.2. OVERR1 – MV1 is Overridden

- Type: BOOL

This output indicates that MV1 is not selected for output MV. This output must be connected to input parameter OVERR of first PID control FB.

14.2.3. OVERR2 – MV2 is Overridden

- Type: BOOL

This output indicates that MV2 is not selected for output MV. This output must be connected to input parameter OVERR of second PID control FB.

15. ControlSplitRange

This FUN is used when the MV output of a PID control FB (PIDA_REAL, PIDA_INT, PIDA_TUNE_REAL, PIDA_TUNE_INT) must be split to command two different valves, commanded by outputs MV1 and MV2.

The following figure shows the input and output parameters of this FUN.

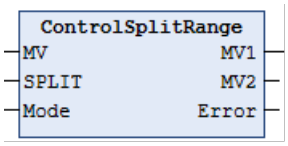

Figure 40: ControlSplitRange parameters

The MV output of a PID control FB is the input of ControlSplitRange FUN, while MV1 and MV2 outputs of ControlSplitRange FUN are sent to the two valves. The input SPLIT of ControlSplitRange FUN defines the split point.

Values of MV, SPLIT, MV1 and MV2 must be previously normalized in the range 0% ... 100% for using ControlSplitRange FUN. In the four next figures, a ramp from 0% to 100% is forced in input MV, and SPLIT is 40%. The figures show the response in MV1 and MV2 for each possible value of Mode input (1, 2, 3, and 4).

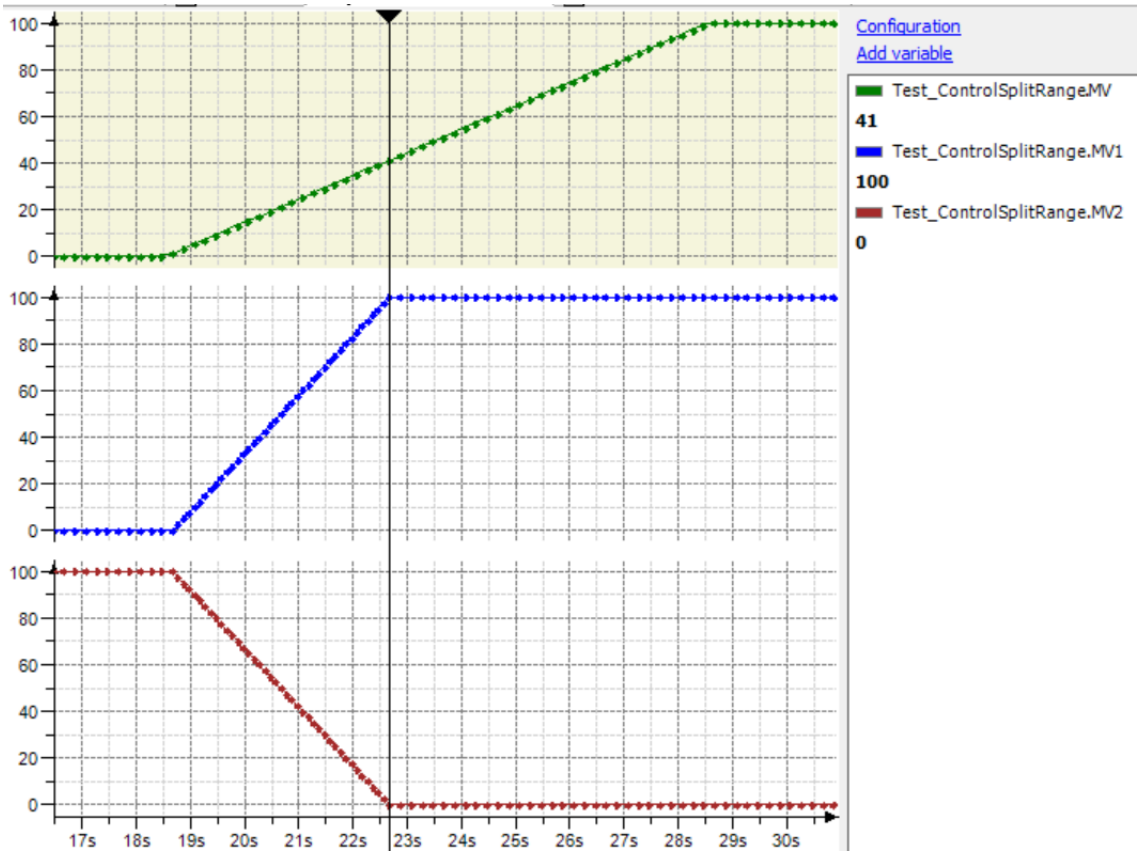

Figure 41: ControlSplitRange with Mode = 1 and SPLIT = 40%

Mode 1 works in the following way:

- MV in 0% ... SPLIT%:
  - MV1 in 0% ... 100%
  - MV2 in 100% ... 0%
- MV in SPLIT% ... 100%:

- MV1 = 100%
- MV2 = 0%

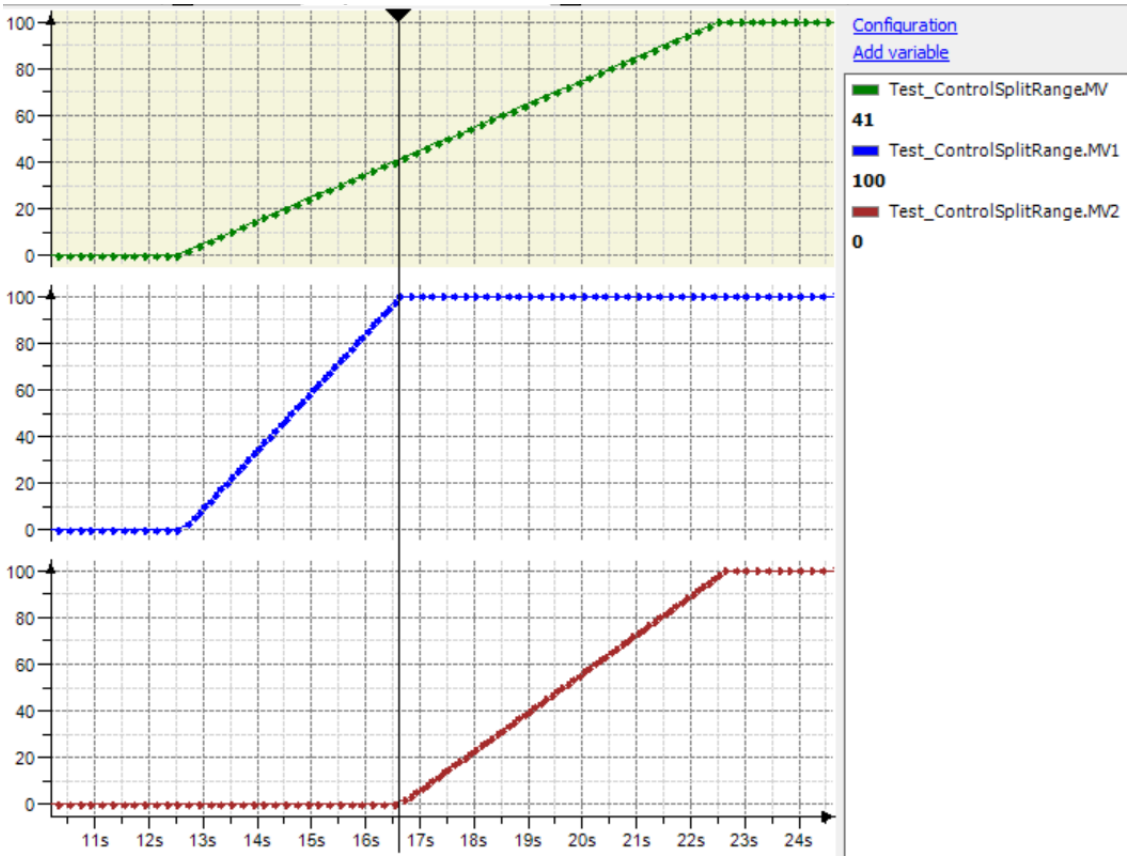

Figure 42: ControlSplitRange with Mode = 2 and SPLIT = 40%

Mode 2 works in the following way:

- MV in 0% ... SPLIT%:
  - MV1 in 0% ... 100%
  - MV2 = 0%
- MV in SPLIT% ... 100%:
  - MV1 = 100%
  - MV2 in 0% ... 100%

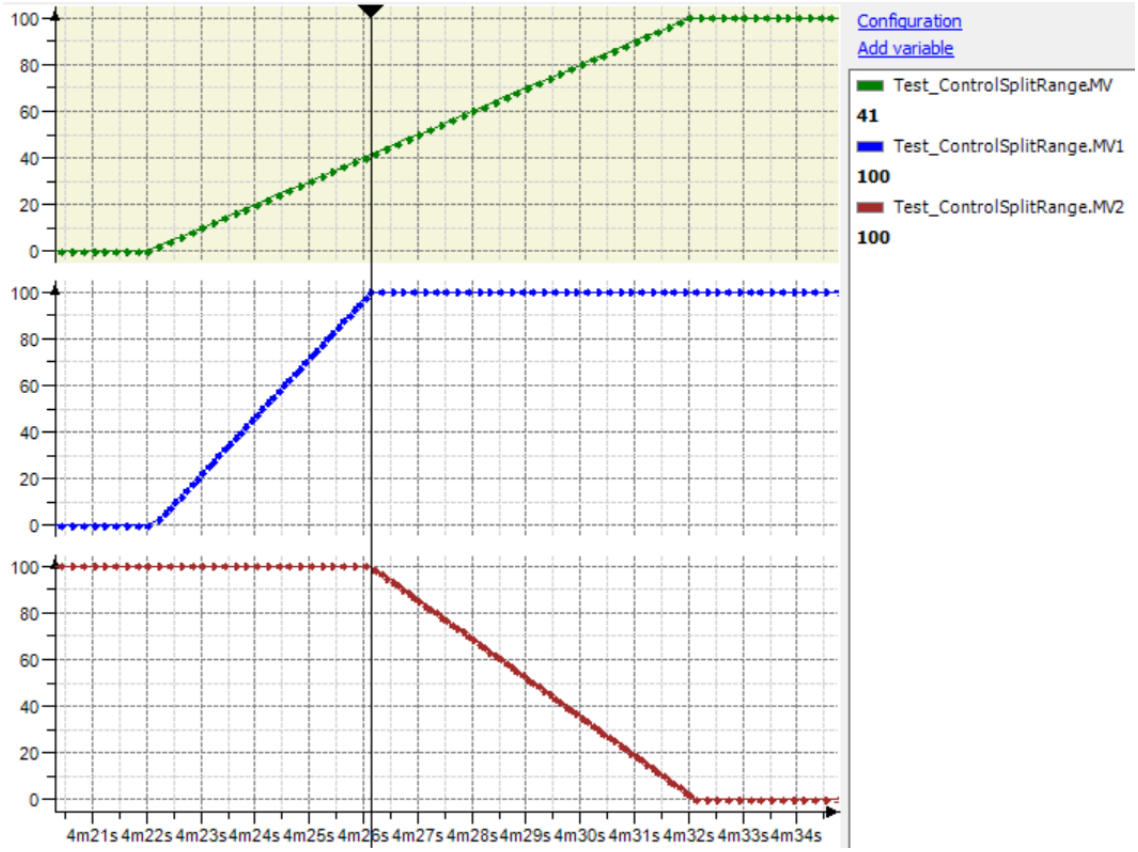

Figure 43: ControlSplitRange with Mode = 3 and SPLIT = 40%

Mode 3 works in the following way:

- MV in 0% ... SPLIT%:
  - MV1 in 0% ... 100%
  - MV2 = 100%
- MV in SPLIT% ... 100%:
  - MV1 = 100%
  - MV2 in 100% ... 0%

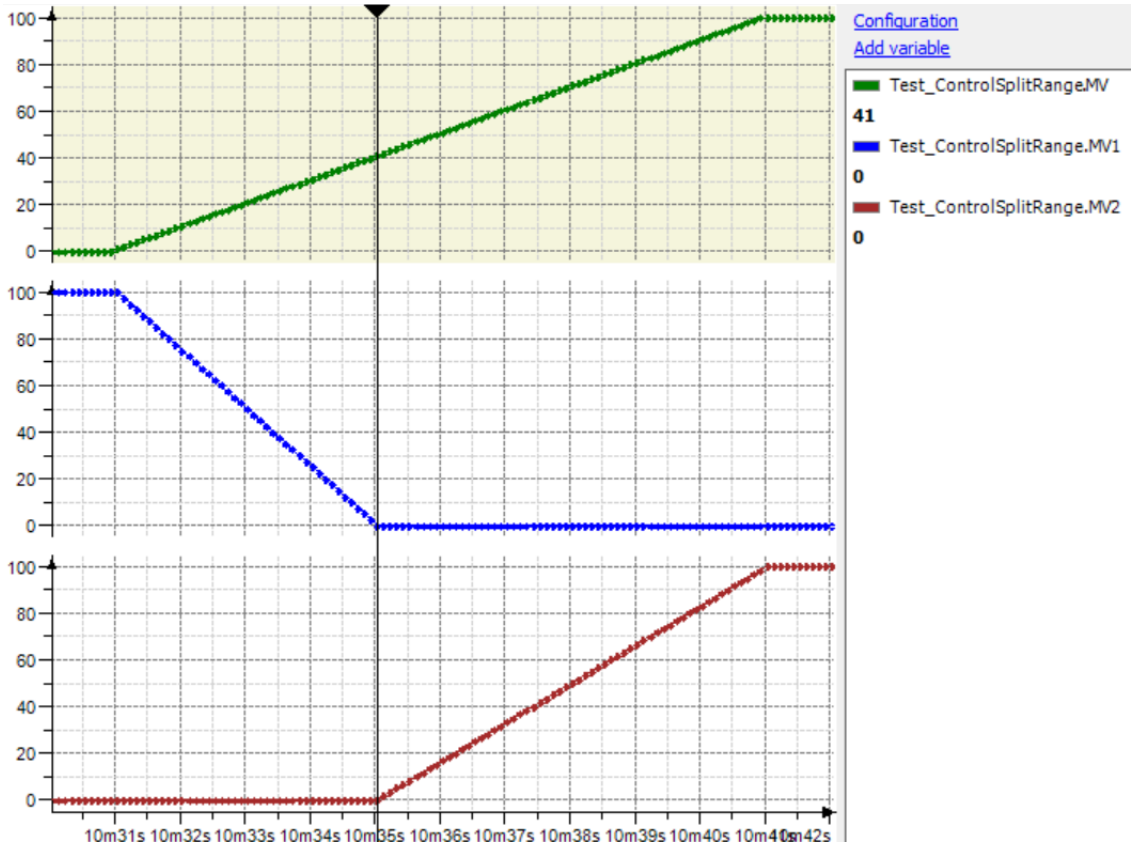

Figure 44: ControlSplitRange with Mode = 4 and SPLIT = 40%

Mode 4 works in the following way:

- MV in 0% ... SPLIT%:
  - MV1 in 100% ... 0%
  - MV2 = 0%
- MV in SPLIT% ... 100%:
  - MV1 = 0%
  - MV2 in 0% ... 100%

15.1. Input Parameters

15.1.1. MV – MV from PID

- Type: REAL
- Range: 0% ... 100%

This input variable corresponds to MV output of PID control FB, used to produce the two split values (MV1 and MV2).

15.1.2. SPLIT – Split Point

- Type: REAL
- Range: > 0% and < 100% (must neither be equal to 0% nor to 100%)

This input variable corresponds to the split point. See previous figures for a better understanding.

15.1.3. Mode – Operation Mode

- Type: USINT
- Range: 1 ... 4

Four operation modes are defined. See previous figures for a better understanding.

15.2. Output Parameters

15.2.1. MV1 – MV for Valve 1

- Type: REAL

Output connected to valve 1.

15.2.2. MV2 – MV for Valve 2

- Type: REAL

Output connected to valve 2.

15.2.3. Error – Fatal Error Indication

- Type: BOOL

This output parameter is TRUE when a fatal error is aborting execution of this FUN. It is set when one of the following errors occurs:

- Mode < 1
- Mode > 4
- MV < 0
- MV > 100
- SPLIT <= 0
- SPLIT >= 100

16. Application Notes

16.1. PID Simulation with Auto-Tuning

The following example, using CFC language, shows a simple PID simulation, where a PIDA_TUNE_REAL FB is connected to a simulated process (first order with dead time, using ControlDelayLag FB). Additional instructions were added for bumpless transfer between automatic and manual modes, and for auto-tuning.

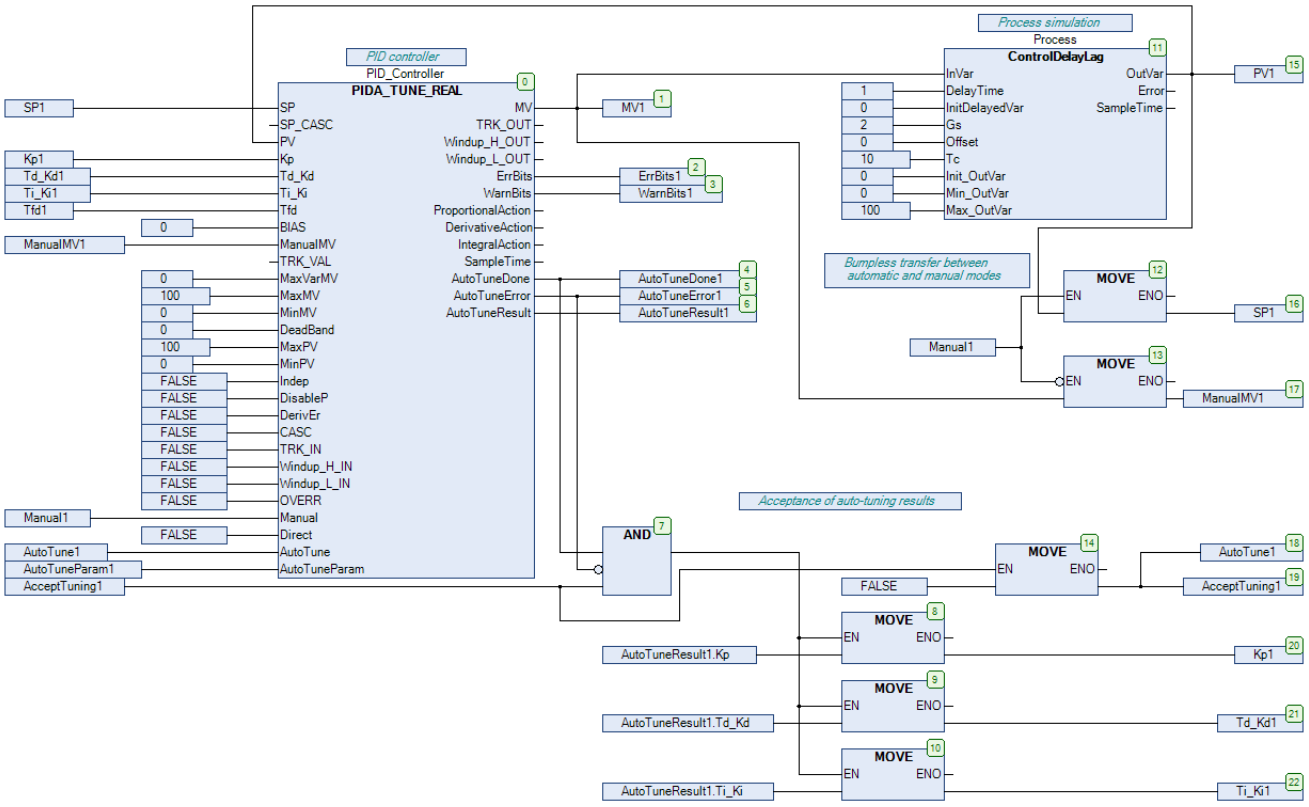

Figure 45: Example of PID simulation with auto-tuning

16.1.1. Code Description

PID_Controller (instance of PIDA_TUNE_REAL FB) is connected to the simulated Process (instance of ControlDelayLag FB). The simulated Process has the following parameters:

- Static gain (Gs): 2
- Time constant (Tc): 10 seconds
- Dead time (DelayTime): 1 second

Note the following connections between these two FB instances:

- Output MV of PID_Controller is connected to input InVar of Process;
- Output OutVar of Process is connected to input PV of PID_Controller.

In the leftmost column of the figure, several variables typically received from the SCADA system are shown:

- SP1: setpoint for PID_Controller
- Kp1, Td_Kd1, Ti_Ki1: tuning parameters for the PID_Controller
- ManualMV1: value for PID_Controller output in manual mode
- Manual1: command for selecting manual or automatic mode for PID_Controller
- AutoTune1: command for switching PID_Controller to auto-tuning mode
- AutoTuneParam1: parameters for auto-tuning mode

- AcceptTuning1: command for accepting auto-tuning results

Bellow the Process, there are two MOVE instructions intended for bumpless transfer between automatic and manual modes. This kind of code is optional, but it is used in many applications. The idea is to copy MV to ManualMV1 in automatic mode, and to copy PV over SP1 in manual mode. So, when switching from automatic to manual, ManualMV1 is already with the same value that MV was right before in automatic mode, avoiding bumps in MV. And when switching from manual to automatic mode, SP1 is already with the same value that PV was right before in manual mode, also avoiding bumps in MV.

Finally, in the lower right corner of the figure, there are one AND and four MOVE instructions intended for acceptance of auto-tuning results. The acceptance occurs when the command AcceptTuning1 is received from the SCADA system. The tuning parameters calculated (inside AutoTuneResult1) are copied over the current tuning parameters (Kp1, Td_Kd1, Ti_Ki1) only if AutoTuneDone is TRUE and AutoTuneError is FALSE. In addition, the commands AutoTune1 and AcceptTuning1 are reset by command AcceptTuning1.

16.1.2. Auto-Tuning Example

The following figure shows an example of auto-tuning, considering the code shown in Figure 45. Example of PID Simulation with Auto-Tuning. The SCADA operator executed the following procedure in this case:

1. He switched controller to manual mode, with ManualMV = 20. It would also be possible to execute auto-tuning in automatic mode, but it is safer in manual mode (see section Safety Recommendations inside chapter PIDA_TUNE_REAL).
2. He adjusted the auto-tuning parameters inside AutoTuneParam1:
   - PercAmpMV = 5% (step of MV is 5, because MaxMV = 100 and MinMV = 0)
   - PercHystPV = 0.1% (PV hysteresis is 0.1, because MaxPV = 100 and MinPV = 0)
   - PercMaxPeakPV = 10% (maximum peaks of PV is 10, because MaxPV = 100 and MinPV = 0)
   - NumCycles = 3 (3 measurement cycles)
   - Mode = AUTOTUNE_PID (PID configuration)
3. He waited until PV is almost steady. In this case, the steady state is 40 (MV = 20, Gs = 2, Offset = 0).
4. He sent a command to set AutoTune1, causing the auto-tuning process to start.
5. He waited until AutoTuneDone1 is TRUE, indicating that auto-tuning has finished.
6. He sent a command to set AcceptTuning1. This command will copy the tuning parameters, if AutoTuneError1 is FALSE.

The following figure shows what happened with MV and PV outputs during the auto-tuning process.

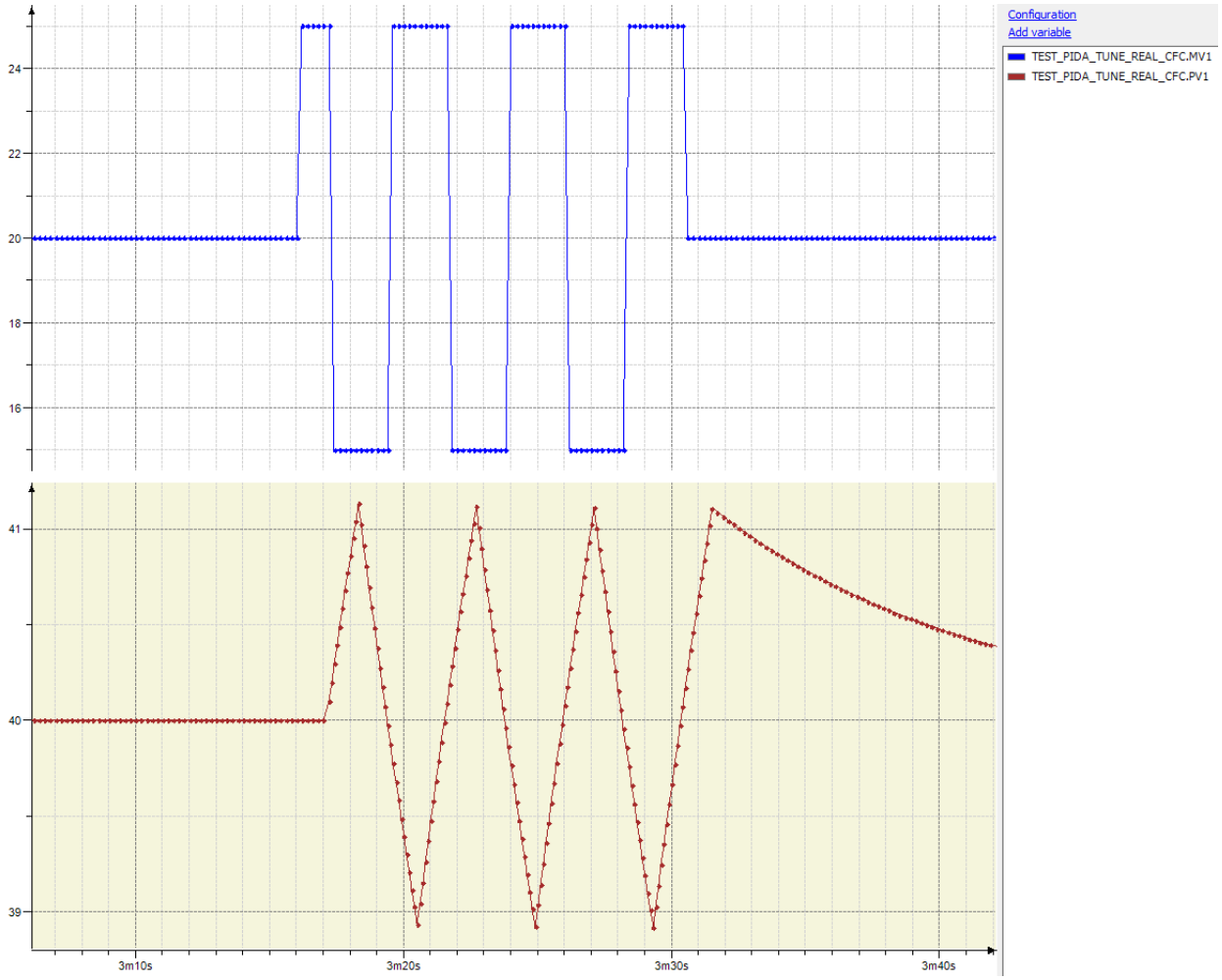

Figure 46: Example of PID simulation with auto-tuning

The tuning parameters calculated in this example were:

- Kp = 1.819
- Ti_Ki = 9.680 seconds
- Td_Kd = 0.698 seconds

16.1.3. Example using the Calculated Tuning Parameters in Automatic Mode

Righ after the auto-tuning was completed, as described in previous subsection, the SCADA operator switched the PID controller from manual to automatic mode. SP and PV were about 40, and SP was changed to 50.

The following figure shows the response.

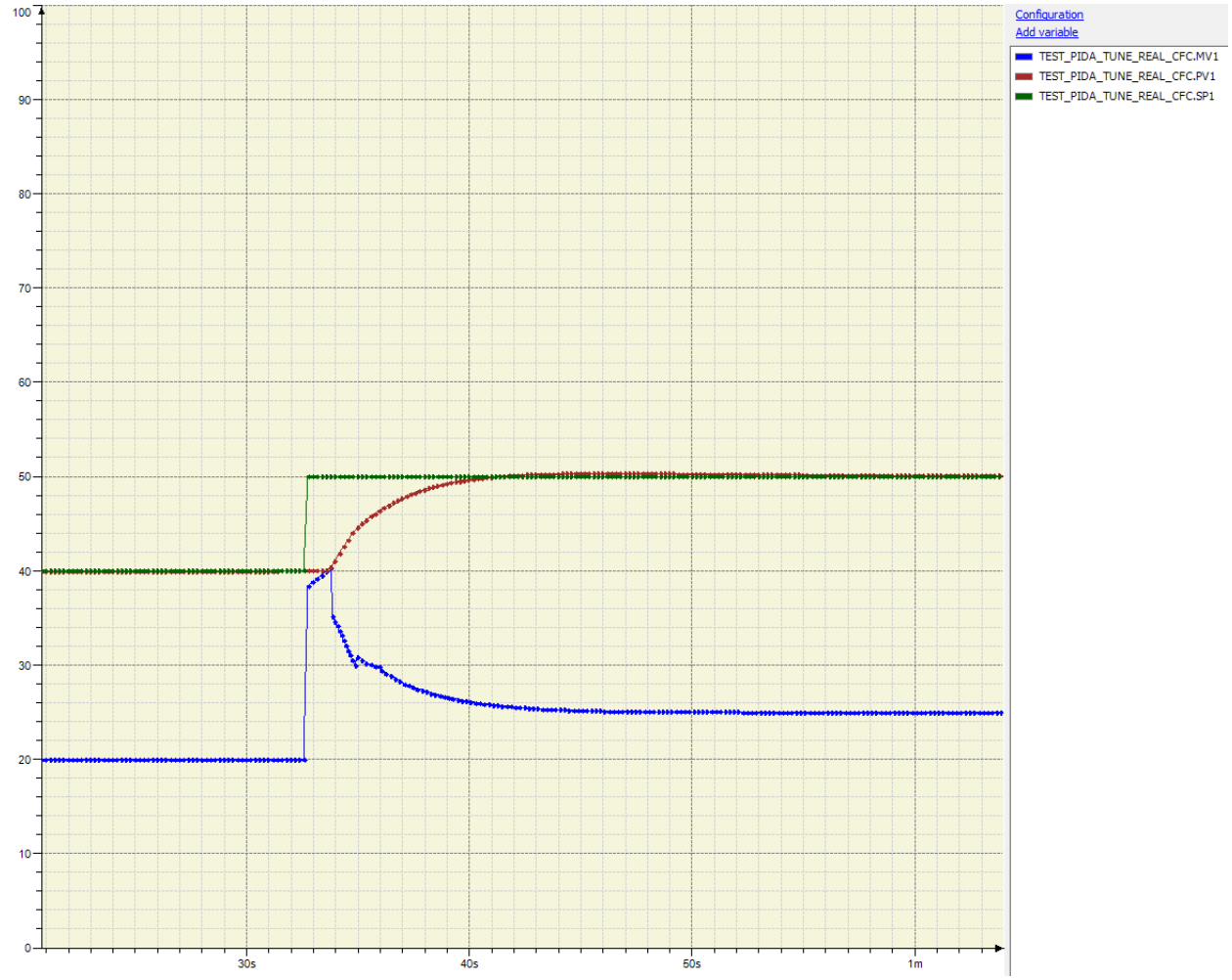

Figure 47: Example of PID simulation in automatic mode

16.2. Avoiding Bumps when Switching between Automatic and Manual

In the example shown in previous section PID Simulation with Auto-Tuning, two MOVE instructions were added for bumpless transfer between automatic and manual modes. These instructions execute the following actions:

- In manual mode, PV1 is copied over SP1. The goal is to provide a bumpless transfer when PID controller switches to automatic mode.
- In automatic mode, MV1 is copied over ManualMV1. The goal is to provide a bumpless transfer when PID controller switches to manual mode.

This additional piece of code is optional. It can be used according to each application, but normally is a recommended practice.

16.3. I/O Diagnostics

It is also recommended to evaluate I/O diagnostics, like the following:

- Broken wire (4-20 mA);
- I/O module failure;
- I/O channel failure;
- Remote I/O communication failure.

If any of such failures avoid reading correctly the analog input, or writing correctly to the analog output, it is recommended to:

- Switch the controller mode to manual;
- Eventually, copy a safe value ManualMV;
- Alarm the failure for the user, so he can start repairing the failure.

16.4. Cascade Configurations

This section provides guidelines on how to use cascade configurations.

16.4.1. Special Connections

Cascade configurations involve some special input and outputs parameters of PID FBs (PIDA_REAL, PIDA_INT, PIDA_TUNE_REAL, PIDA_TUNE_INT) that must be connected carefully:

- Inputs:
  - SP_CASC
  - CASC
  - TRK_VAL
  - TRK_IN
  - Windup_H_IN
  - Windup_L_IN
- Outputs:
  - TRK_OUT
  - Windup_H_OUT
  - Windup_L_OUT

The following figure shows an example of cascade configuration with 3 PID FBs and 3 processes and shows the main connections. In most situations, a cascade is composed by 2 PID controllers and 2 processes. However, this example with 3 PID controllers and 3 processes was used because it allows showing the 3 roles that a PID controller can assume in a cascade configuration:

- Only master (PID1 in example)
- Only slave (PID3 in example)
- Slave and master at the same time (intermediary PIDs - PID2 in example)

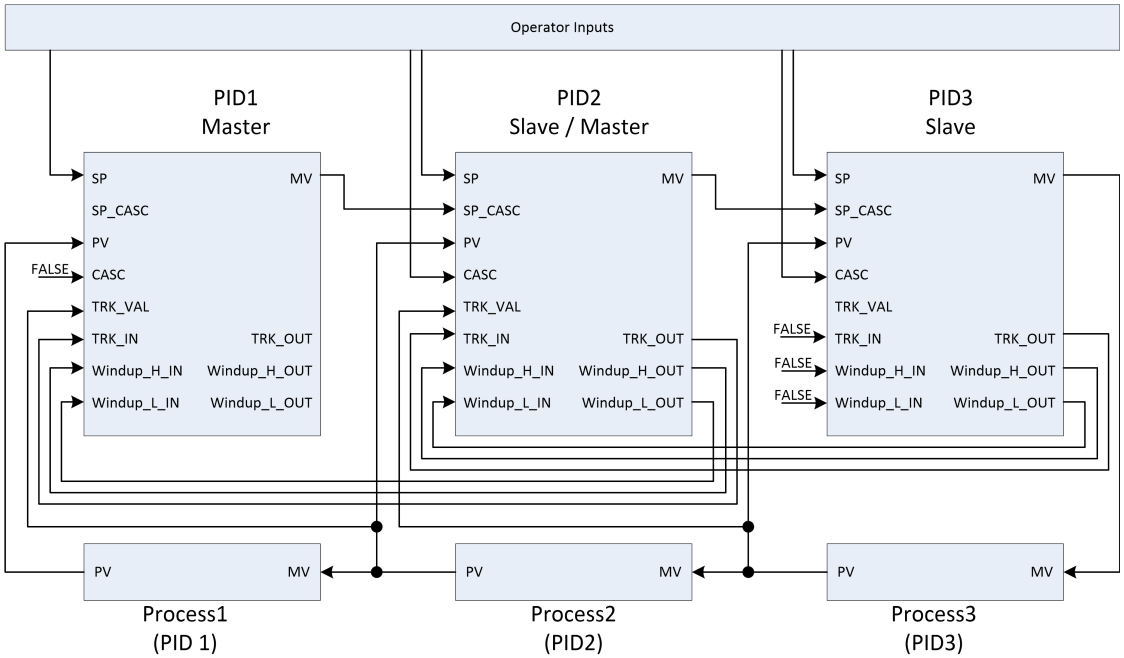

Figure 48: Special connection in cascade configurations

The connections of the special inputs and outputs for cascade configurations are the following:

- PIDs that assume only the master role (PID1):
  - Input CASC must be FALSE;
  - Input SP_CASC does not need to be connected;
  - Input TRK_VAL must be connected to the PV of process 2, controlled by its slave (PID2);
  - Input TRK_IN must be connected to output TRK_OUT of its slave (PID2);
  - Input Windup_H_IN must be connected to output Windup_H_OUT of its slave (PID2);
  - Input Windup_L_IN must be connected to output Windup_L_OUT of its slave (PID2);
  - Output TRK_OUT must not be connected;
  - Output Windup_H_OUT must not be connected;
  - Output Windup_L_OUT must not be connected.
- In PIDs that assume the slave and master role (PID2):
  - Input CASC normally comes from operator input, like HMI;
  - Input SP_CASC must be connected to output MV of its master (PID1);
  - Input TRK_VAL must be connected to the PV of process 3, controlled by its slave (PID3);
  - Input TRK_IN must be connected to output TRK_OUT of its slave (PID3);
  - Input Windup_H_IN must be connected to output Windup_H_OUT of its slave (PID3);
  - Input Windup_L_IN must be connected to output Windup_L_OUT of its slave (PID3);
  - Output TRK_OUT must be connected to input TRK_IN of its master (PID1);
  - Output Windup_H_OUT must be connected to input Windup_H_IN of its master (PID1);
  - Output Windup_L_OUT must be connected to input Windup_L_IN of its master (PID1).
- In PIDs that assume only the slave role (PID3):
  - Input CASC normally comes from operator input, like HMI;
  - Input SP_CASC must be connected to output MV of its master (PID2);
  - Input TRK_VAL doesn't need to be connected;
  - Input TRK_IN must be FALSE;
  - Input Windup_H_IN must be FALSE;
  - Input Windup_L_IN must be FALSE;
  - Output TRK_OUT must be connected to input TRK_IN of its master (PID2);
  - Output Windup_H_OUT must be connected to input Windup_H_IN of its master (PID2);
  - Output Windup_L_OUT must be connected to input Windup_L_IN of its master (PID2).

16.4.2. Purpose of Special Connections

The following paragraphs give some additional information about the inputs and outputs specially created for cascading.

SP_CASC and CASC inputs were created to provide an easy way to switch between the operator setpoint (SP) and the cascaded setpoint (SP_CASC) in a slave. SP_CASC comes from the MV output of the corresponding master. If CASC input is TRUE, the setpoint of the slave is SP_CASC; otherwise, it comes from the SP input, typically provided by the operator.

TRK_OUT output means that a slave (or master/slave) is not in cascade/automatic mode (CASC is false, or Manual is true). TRK_OUT is also set to true when TRK_IN input is true.

The TRK_OUT output is connected to TRK_IN input of the corresponding master, as explained before.

When TRK_IN input is TRUE in a master, the master is in tracking mode. In this mode, the master knows that the slave is not following the setpoint provided by its MV output. Therefore, the master will take some actions:

- Copy TRK_VAL input to its MV output. Note that TRK_VAL is equal to PV of its slave process. So, SP_CASC of the slave controller becomes equal to PV of the slave process. Therefore, when the slave controller switches back to automatic/cascade, no bump will occur.
- The integral action is calculated in a special way in the master (like in manual mode), to avoid reset windup and bumps when switching back to automatic without tracking mode.

Windup_H_OUT means that a slave (or master/slave) has saturated its MV output, due to limits specified in parameters MaxMV, MinMV or MaxVarMV. This signal must be connected to input Windup_H_IN of the corresponding master. When the master observes this input is TRUE, it must not increase its integral action to avoid windup problems, but decreasing the integral action is still allowed.

Windup_L_OUT means that a slave (or master/slave) has saturated its MV output, due to limits specified in parameters MaxMV, MinMV or MaxVarMV. This signal must be connected to input Windup_L_IN of the corresponding master. When the master observes this input is TRUE, it must not decrease its integral action to avoid windup problems, but increasing the integral action is still allowed.

16.4.3. Additional User Code in Cascade Configurations

Note that all the following codes are optional (suggestions), aiming to provide bumpless transfer between operating modes:

- Automatic (Manual = FALSE);
- Manual (Manual = TRUE);
- Cascade (CASC = TRUE);
- Tracking (TRK_IN = TRUE).

16.4.3.1. Copy PV to SP in Manual Mode

A controller can copy its PV input to its SP input in manual mode. This avoids bumps when switching to automatic mode.

16.4.3.2. Copy MV to ManualMV in Automatic Mode

A controller can copy its MV output to its ManualMV input in automatic mode. This avoids bumps when switching to manual mode.

16.4.3.3. Copy PV to SP in Tracking Mode in Master Controllers

A master controller can copy its PV input to its SP input in tracking mode. This avoids bumps when switching to automatic/non-tracking mode.

16.4.3.4. Copy MV to ManualMV in Tracking Mode in Master Controllers

A master controller can copy its MV output to its ManualMV input in tracking mode. This avoids bumps when switching to manual mode.

16.4.3.5. Copy PV to SP in Automatic/Cascade Mode in Slave Controllers

A slave controller can copy its PV input to its SP input in automatic/cascade mode. This avoids bumps when switching to automatic/non-cascade mode.

16.5. Override Configurations

Override configurations are necessary when a single control valve affects two or more processes. In this case, the two or more PID controllers connected to these processes must use the override mode. This is necessary because only one of the controllers will be controlling the valve, while the other controllers are overridden.

For instance, suppose that the same valve controls temperature and pressure of a system, and that increasing MV causes both temperature and pressure to increase. So, two controllers with reverse control are allocated for controlling temperature and pressure. But there is an additional requirement: the temperature and pressure must not exceed their corresponding setpoints. So, it is acceptable to have temperature in its setpoint and pressure bellow its setpoint, and to have pressure in its setpoint and temperature bellow its setpoint. The solution is to use the minimum value of MV outputs calculated by the two controllers. But this is not enough. Note that one controller is effectively controlling the valve (that one with PV close to SP) while the other controller is not controlling the valve (that one with PV bellow SP). It is necessary to inform that controller not controlling the valve that it is overridden. Otherwise, reset windup problems will arise in the integral action.

Two special inputs of PID FBs (PIDA_REAL, PIDA_INT, PIDA_TUNE_REAL, PIDA_TUNE_INT) are used for override configurations:

- OVERR: informs that controller is overridden (not controlling the valve);
- TRK_VAL: informs the MV output of the non-overridden controller (controlling the valve), used for a back-calculation of integral action in the overridden controller when OVERR is TRUE.

Additional functions (selectors) must be used in conjunction with PID FBs for implementing an override configuration. This is necessary for selecting which PID FB will be controlling its process at given time, and which is overridden. Two selector FUNs are available (ControlSelectMin and ControlSelectMax), and they can manage only two PIDs/processes. However, selector functions can be easily developed for other situations, like those involving more than two PIDs/processes.

A PID FB knows it is overridden (not selected) when its OVERR input is TRUE. In this case, it makes a different calculation of the integral action to avoid windup problems. For this purpose, the TRK_VAL input must be connected to the selected output (output MV of the selector FUN).

The following figure shows an example of override configuration with two PID FBs, in conjunction with selector ControlSelectMin. Only the main input/output parameters related to override configuration are shown in the figure.

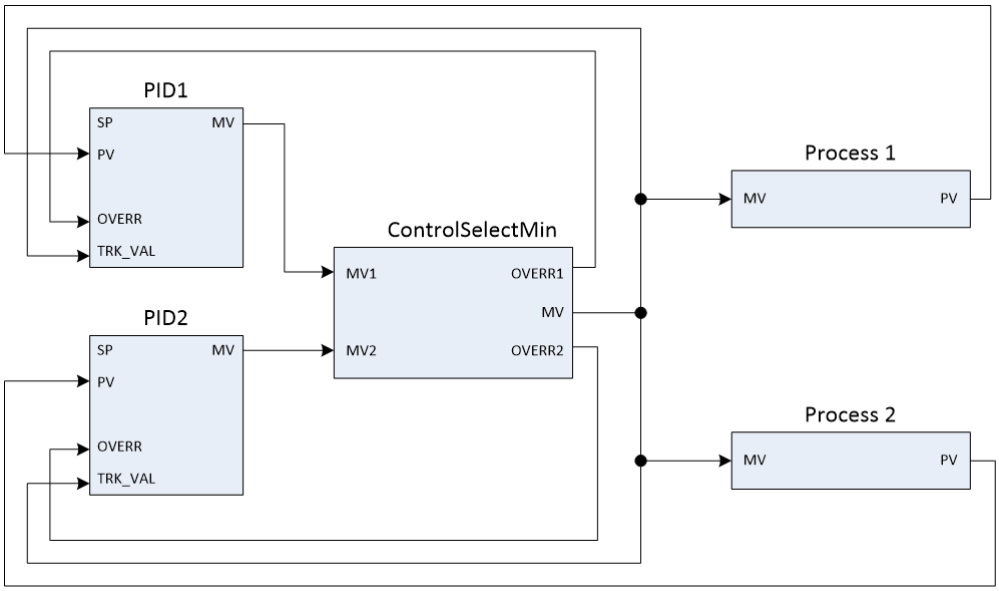

Figure 49: Special connections in override configuration

FUN ControlSelectMin takes as inputs the MVs (MV1 and MV2) calculated by the two PID FBs (PID1 and PID2). Then, ControlSelectMin calculates the following outputs:

- MV: selects the minimum value between MV1 and MV2;
- OVERR1: this output is TRUE if MV1 is not selected (if MV2 <= MV1, so that MV = MV2);
- OVERR2: this output is TRUE if MV2 is not selected (if MV1 < MV2, so that MV = MV1);

16.6. Ratio Control

The objective of ratio control is to maintain the ratio of two process variables as a specified value. The two variables are usually flow rates: a manipulated variable PVm, and a disturbance variable PVd. The controlled ratio is R = PVm / PVd. The objective is to control PVm so that it gets equal to R * PVd.

The following figure shows a typical P&ID for ratio control between two flows.

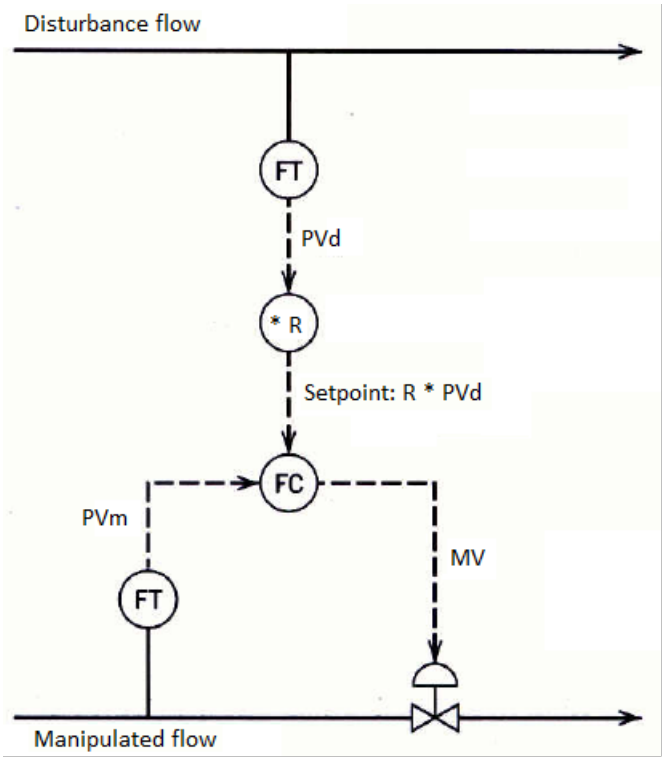

Figure 50: P&ID for ratio control

Two flow transmitters (FT) are used to measure PVm and PVd. A simple multiplication instruction is used to calculate the setpoint (R * PVd) for the flow controller (FC) which objective is to make PVm = R * PVd (R is calculated as PVm / PVd).

Usually the setpoint "R * PVd" is connected to SP_CASC, so that the main setpoint (SP) can be used to control the manipulated flow (PVm) independently. So, with CASC = TRUE, the FC is a ratio controller. With CASC = FALSE, FC controls only PVm.

16.7. Feed-Forward

This section describes how a feed-forward controller, combined with a PID feedback controller, can improve the response against disturbances. It also shows how the feed-forward controller can be implemented using FBs ControlDelay and ControlLeadLag.

16.7.1. Simulated Process for Illustrating Feed-Forward Control

This section uses a simulated process for illustrating an example of feed-forward control combined with feedback control. The simulated process is shown in the following figure.

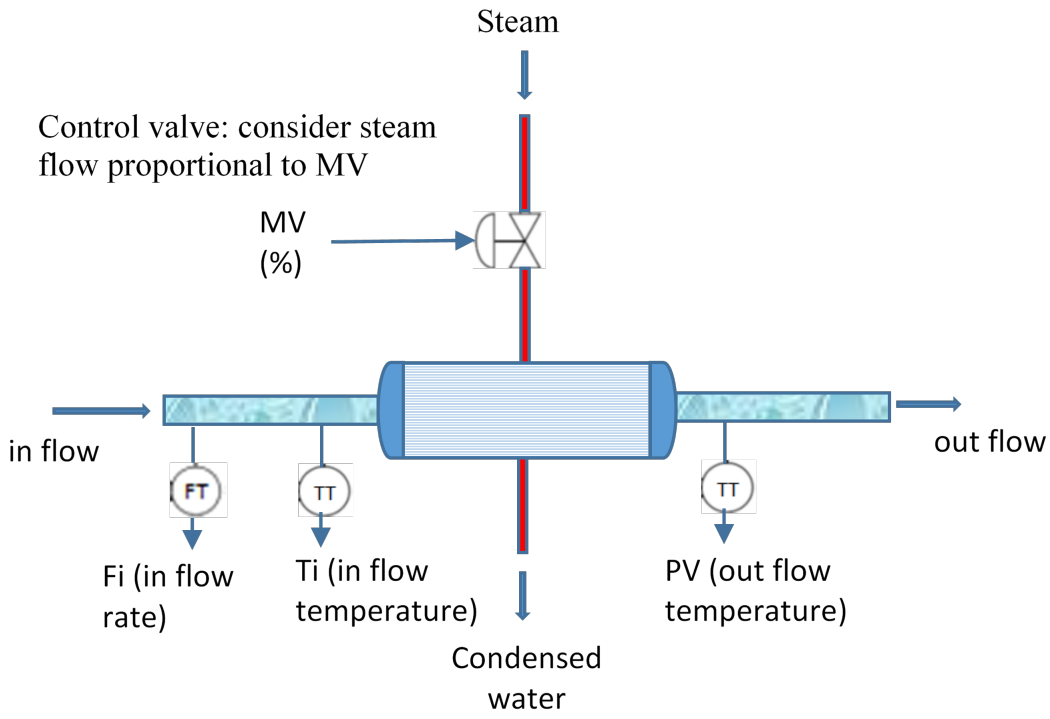

Figure 51: Simulated process for feed-forward control

This process is intended to control out flow temperature (PV) by adjusting the steam flow in the heat exchanger (steam flow is proportional to MV).

Two disturbances affect the process:

- Fi: the inflow rate
- Ti: the inflow temperature

The following energy balance equation is a good approximation for describing the steady state relationship between PV, MV, Ti and Fi:

$$K1 * MV = K2 * Fi * (PV - Ti) \tag{14}$$

K1 and K2 are constants.

The left side of equation (K1 * MV) corresponds to heat coming from steam that is proportional to valve position.

The right side of equation (K2 * Fi * (PV – Ti)) corresponds to heat necessary for increasing temperature of inflow Fi from temperature Ti to temperature PV.

The previous equation can be rewritten as:

$$PV = MV * K/Fi + Ti(whereK = K1/K2) \tag{15}$$

Note that this process is not linear, because its static gain is not constant. The static gain (ΔPV/ΔMV) is K / Fi, and Fi is a changing disturbance. For tuning such a process without risk of instability, one should consider the maximum static gain (K / Fi) that occurs for the minimum value of Fi allowed for the process. This will result in the minimum proportional gain of PID controller, because it is inversely proportional to process static gain, as explained in chapter PID Tuning Tips.

It is possible to calculate the constant K from this process, by adjusting a fixed value in MV, and measuring PV, Fi and Ti using the corresponding transmitters shown in previous figure. For better precision, it is advisable to make several calculations and averaging them, using different values of MV, PV, Fi and Ti.

In our example, consider that K = 2, so that:

$$PV = MV * 2/Fi + Ti \tag{16}$$

Equation 16. Equation for PV in process simulated for feed-forward control

This equation can also be written as:

$$MV = (PV - Ti) * Fi/2 \tag{17}$$

Equation 17. Equation for MV in process simulated for feed-forward control

Equation 16: represents how MV, Fi and Ti affect the steady state process variable (PV).

Equation 17: is the same equation written in another way. This equation is useful for implementing the feed-forward controller.

Besides the equation for calculating steady state value of PV, it is also necessary to consider some dynamic parameters for modelling how MV, Fi and Ti affect PV during transients. These dynamic parameters, and time values assumed for them, are the following:

- Tde_MV = 1 s: dead time between steam flow (MV) and effect in PV
- Tc_MV = 10 s: lag time (time constant) between steam flow (MV) and PV
- Tde_Ti = 2 s: dead time between Ti and effect in PV
- Tc_Ti = 5 s: lag time (time constant) between Ti and PV
- Tde_InvFi = 1.5 s: dead time between 1/Fi and effect in PV
- Tc_InvFi = 4 s: lag time (time constant) between 1/Fi and PV

The inverse of Fi (1/Fi) was used for measuring dynamic parameters between Fi and PV, because Fi is inversely proportional to PV, as shown in **Equation 16**.

These dynamic parameters can be measured using open loop tests, applying step inputs in MV, Ti and Fi and observing response of PV (see section Open Loop Process Characterization inside chapter PID Tuning Tips).

16.7.2. Implementation in FBD Language

The strategy for implementing a feed-forward controller combined with feedback controller for this example of process is shown in the following networks, using FBD language.

Initially this example includes 5 networks just for simulating the process (networks 1 to 5). They are unnecessary in a real system.

After this, 3 networks (6 to 8) implement the feed-forward controller.

Network 9 was only included for disabling the feed-forward controller, just for comparing results with and without feed-forward controller. It is unnecessary in a real system.

Finally network 10 implements the feedback controller (PID).

16.7.2.1. Network1

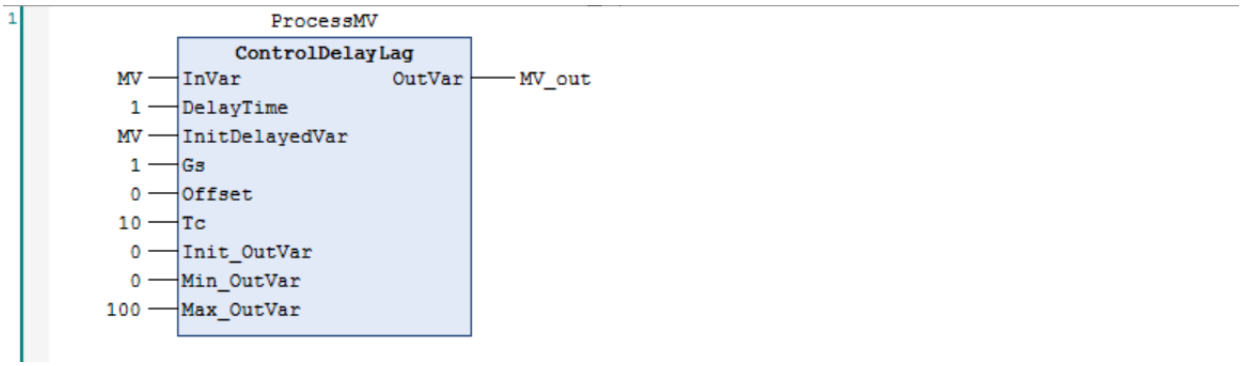

Figure 52: Network 1 of feed-forward example

This network implements the previously discussed dynamic parameters between MV and PV:

- Tde_MV = 1 s: dead time between steam flow (MV) and effect in PV, connected to input DelayTime of FB ControlDelayLag;
- Tc_MV = 10 s: lag time (time constant) between steam flow (MV) and PV, connected to input Tc of FB ControlDelayLag.

The output MV_out includes these dynamic parameters, and is used for calculating the final value of PV in network 5.

16.7.2.2. Network2

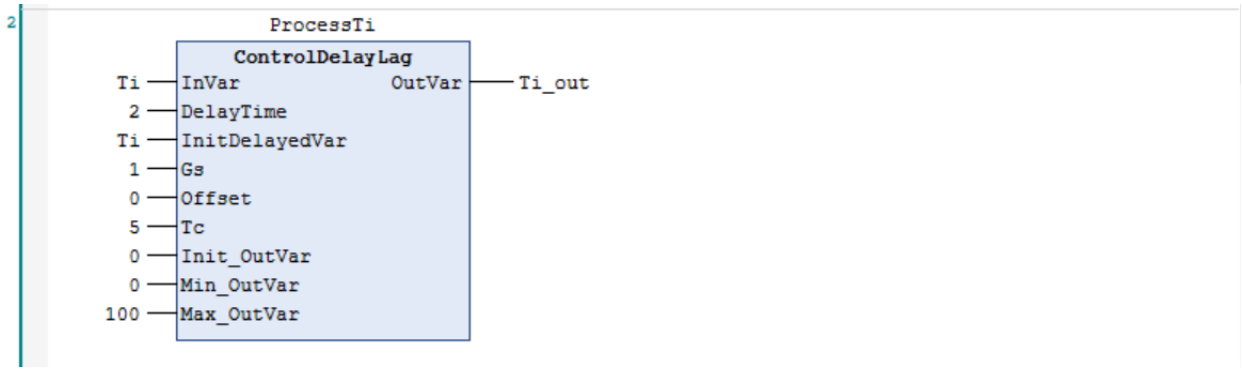

Figure 53: Network 2 of feed-forward example

This network implements the previously discussed dynamic parameters between Ti and PV:

- Tde_Ti = 2 s: dead time between inflow temperature (Ti) and effect in PV, connected to input DelayTime of FB ControlDelayLag;
- Tc_Ti= 5 s: lag time (time constant) between inflow temperature (Ti) and PV, connected to input Tc of FB ControlDelayLag.

The output Ti_out includes these dynamic parameters, and is used for calculating the final value of PV in network 5.

16.7.2.3. Network3

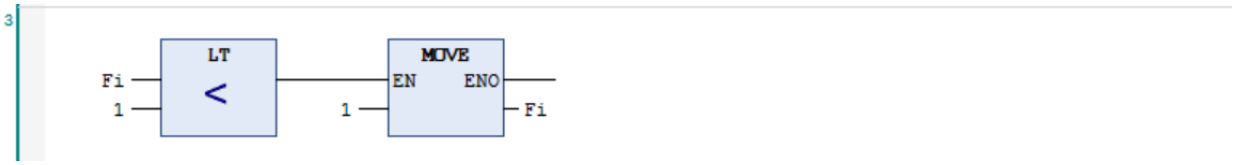

Figure 54: Network 3 of feed-forward example

This network limits minimum value of Fi to 1, with the following purposes:

- Avoid divisions by zero in network 4, when calculating InvFi;
- Limit the maximum value of process static gain (2 / Fi, see equation 16). So, the maximum static gain is 2, and a conservative tuning can use this value (proportional gain Kp of PID controller in network 1 is inversely proportional to static gain).

16.7.2.4. Network4

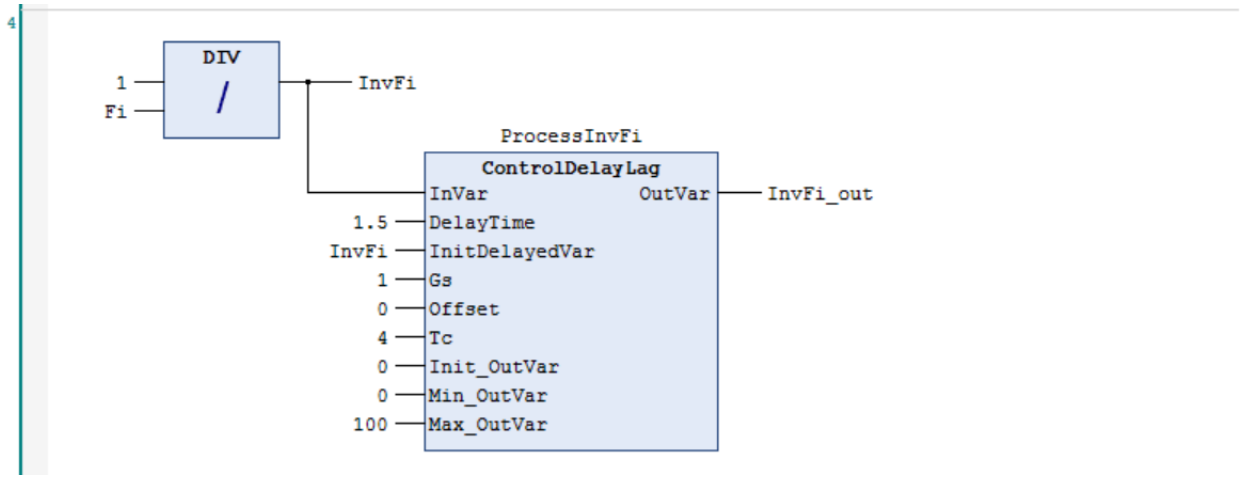

Figure 55: Network 4 of feed-forward example

This network first calculates InvFi as the inverse of Fi.

After this, it implements the previously discussed dynamic parameters between InvFi and PV:

- Tde_InvFi = 1.5 s: dead time between inverse of in flow (InvFi) and effect in PV, connected to input DelayTime of FB ControlDelayLag;
- Tc_InvFi = 4 s: lag time (time constant) between inverse of in flow (InvFi) and PV, connected to input Tc of FB ControlDelayLag.

The output InvFi_out includes these dynamic parameters, and is used for calculating the final value of PV in network 5.

16.7.2.5. Network5

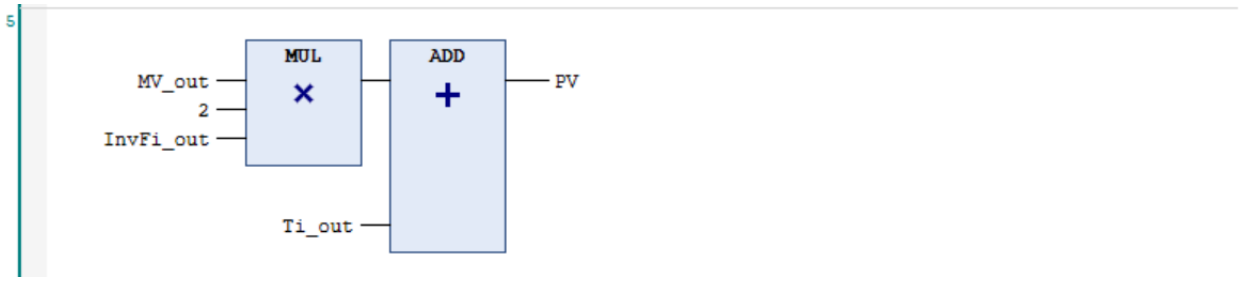

Figure 56: Network 5 of feed-forward example

This network completes the process simulation.

It calculates PV according **Equation 16**, using the values that include dynamic parameters (dead times and time constants).

16.7.2.6. Network6

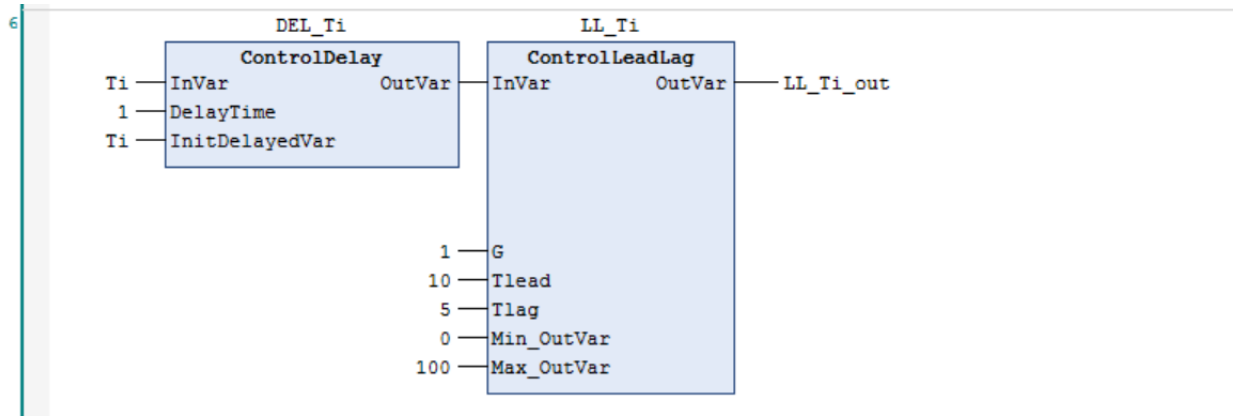

Figure 57: Network 6 of feed-forward example

This is the first network used for the feed-forward controller.
It manages the dynamic effects of Ti over MV.
The ControlDelay FB delays input Ti by 1 second. This delay time is calculated as:

$$Tde\_Ti - Tde\_MV = 2 - 1 = 1second \tag{18}$$

If "Tde_Ti – Tde_MV" was a negative value, then the delay would be zero, and the ControlDelay FB would be unnecessary.
The ControlLeadLag FB must take the following inputs:

- Tlead = Tc_MV = 10 seconds
- Tlag = Tc_Ti = 5 seconds

The final output LL_Ti_out combines the effect of delay and lead-lag.

16.7.2.7. Network7

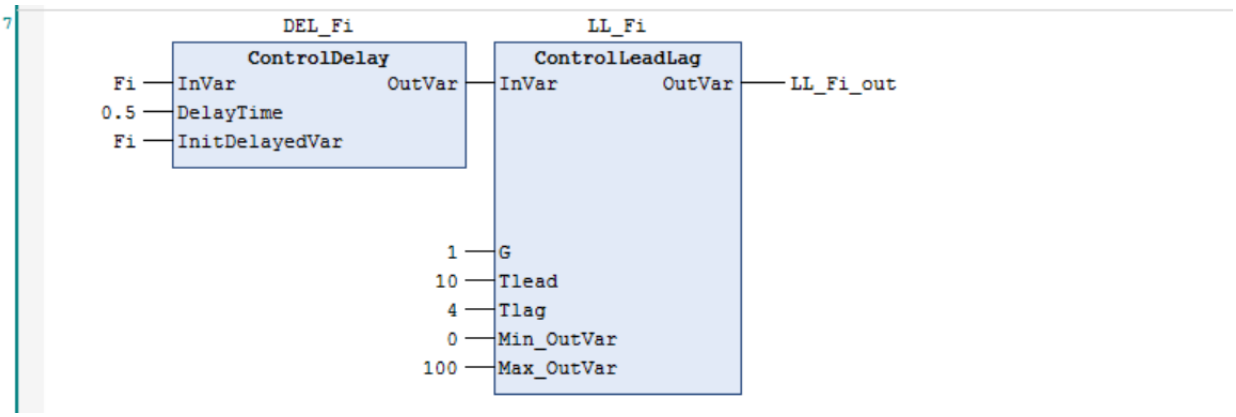

Figure 58: Network 7 of feed-forward example

This is the second network used for the feed-forward controller.
It manages the dynamic effects of Fi over MV.
The ControlDelay FB delays input Fi by 0.5 seconds. This delay time is calculated as:

$$Tde\_InvFi - Tde\_MV = 1.5 - 1 = 0.5 seconds \tag{19}$$

If "Tde_InvFi – Tde_MV" was a negative value, then the delay would be zero, and the ControlDelay FB would be unnecessary.

The ControlLeadLag FB must take the following inputs:

- Tlead = Tc_MV = 10 seconds
- Tlag = Tc_InvFi = 4 seconds

The final output LL_Fi_out combines the effect of delay and lead-lag.

16.7.2.8. Network8

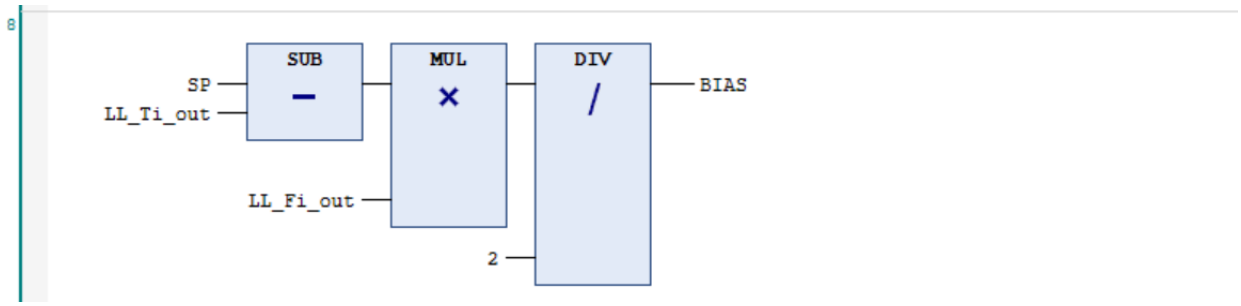

Figure 59: Network 8 of feed-forward example

This network is the last of feed-forward controller. It calculates the BIAS input for PID controller in network 10.

For this purpose, it uses previous **Equation 17**, with the following changes:

- PV is replaced by SP (desired PV);
- Ti is replaced by LL_Ti_out;
- Fi is replaced by LL_Fi_out.

16.7.2.9. Network9

Figure 60: Network 9 of feed-forward example

This network resets BIAS, so disabling the feed-forward controller.

This makes possible to compare the response to disturbances in Ti and Fi with and without the feed-forward controllers (see section Simulating the Effects of Feed-Forward Controller).

16.7.2.10. Network10

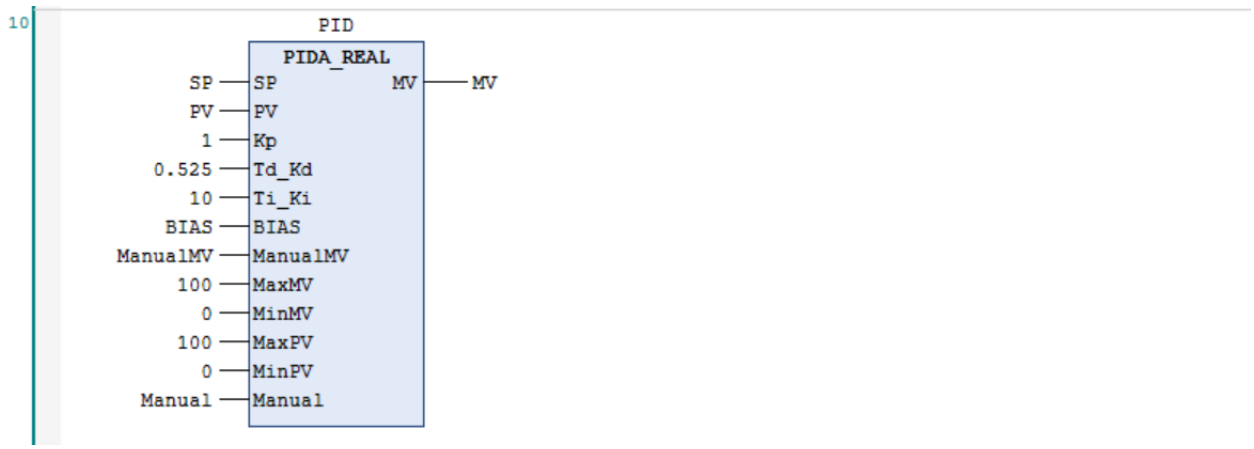

Figure 61: Network 10 of feed-forward example

This is the PID (feedback controller).

Tuning parameters (Kp = 1, Td_Kd = 0.525 s and Ti_Ki = 10 s) considered minimum value of Fi equal 1, so maximum static gain is equal 2 (see Equation 16).

Input BIAS is calculated by the feed-forward controller (networks 7, 8 and 9).

16.8. Simulating the Effects of Feed-Forward Controller

This section simulates the effect of disturbances in Ti and Fi with feed-forward controller enabled and disabled.

16.8.1. Effects of Disturbances in Fi without Feed-Forward Controller

The following figure shows effects of changing Fi from 2.0 to 2.1, and after this from 2.1 to 2.0, with SP = 50, when feed-forward control is disabled.

When Fi increased from 2.0 to 2.1, PV decreased from 50 to 49.177 before returning to 50 (more than 0.8 degrees).

When Fi decreased from 2.1 to 2.0, PV increased from 50 to 50.885 before returning to 50 (more than 0.8 degrees).

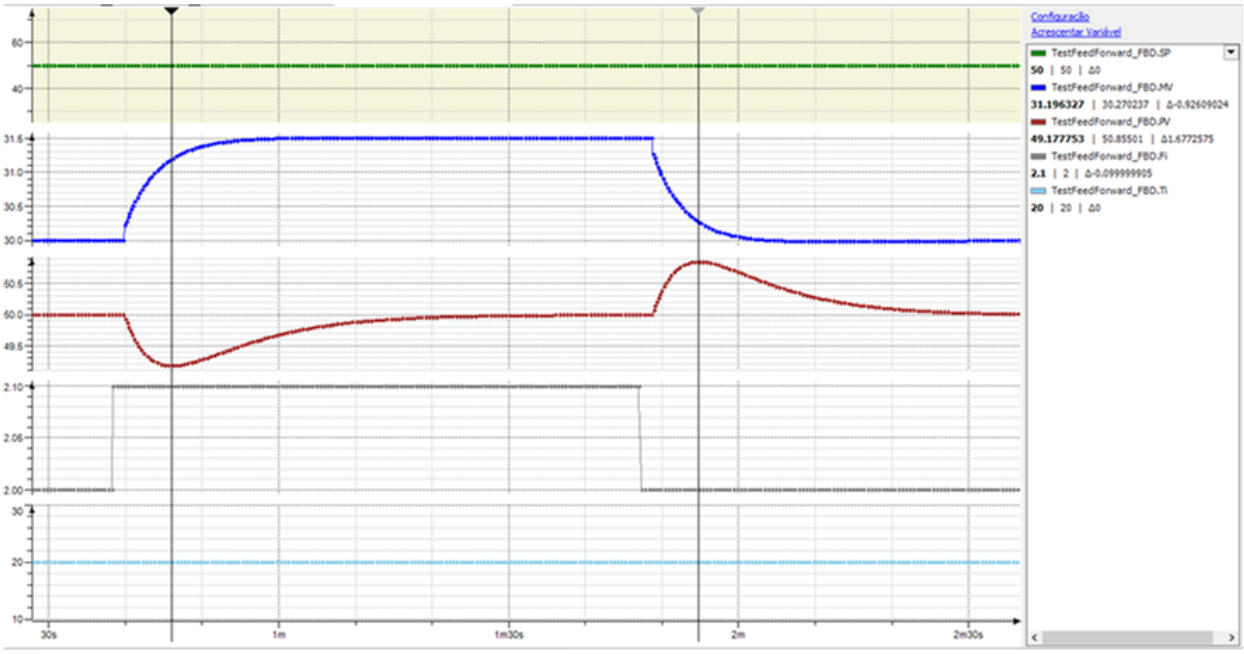

Figure 62: Effect of disturbances in Fi without feed-forward

16.8.2. Effects of Disturbances in Fi with Feed-Forward Controller

The following figure shows effect of changing Fi from 2.0 to 2.1, and after this from 2.1 to 2.0, with SP = 50, when feed-forward control is enabled.

When Fi increased from 2.0 to 2.1, PV decreased from 50 to 49.964 before returning to 50 (less than 0.05 degrees).

When Fi decreased from 2.1 to 2.0, PV increased from 50 to 50.037 before returning to 50 (less than 0.05 degrees).

So, without feed-forward controller, PV deviated about 0.8 degrees from SP, and with feed-forward, only 0.05 degrees.

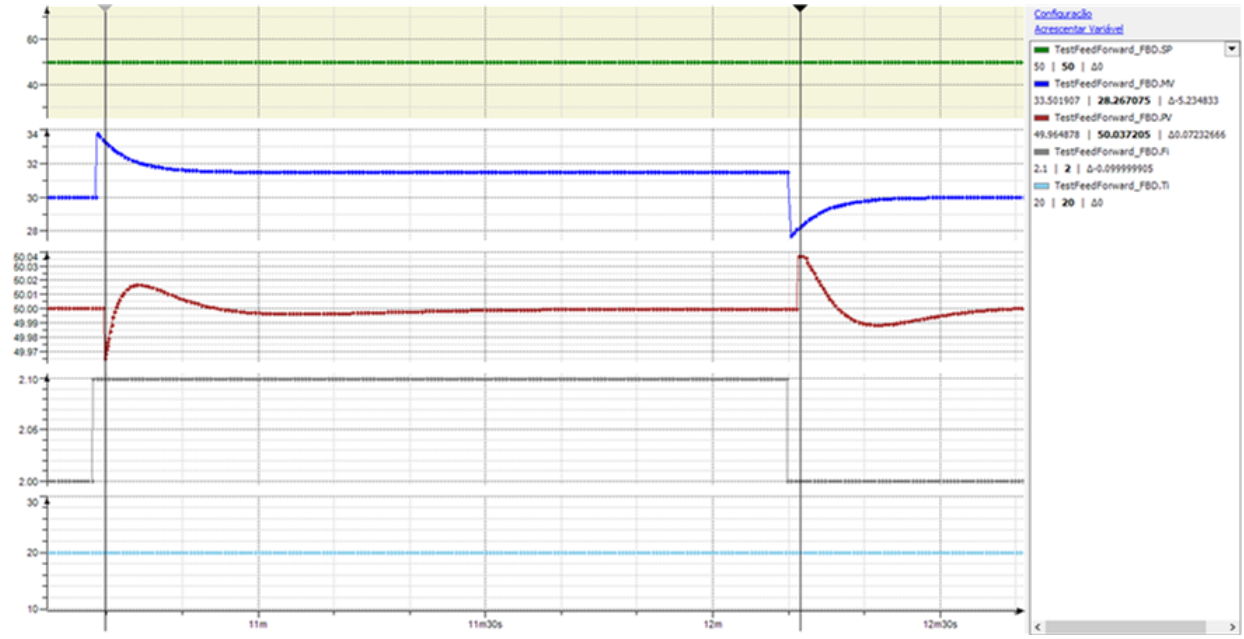

Figure 63: Effect of disturbances in Fi with feed-forward

16.8.3. Effects of Disturbances in Ti without Feed-Forward Controller

The following figure shows effect of changing Ti from 20 to 21, and after this from 21 to 20, with SP = 50, when feed-forward control is disabled.

When Ti increased from 20 to 21, PV increased from 50 to 50.523 before returning to 50 (more than 0.5 degrees).

When Ti decreased from 21 to 20, PV decreased from 50 to 49.476 before returning to 50 (more than 0.5 degrees).

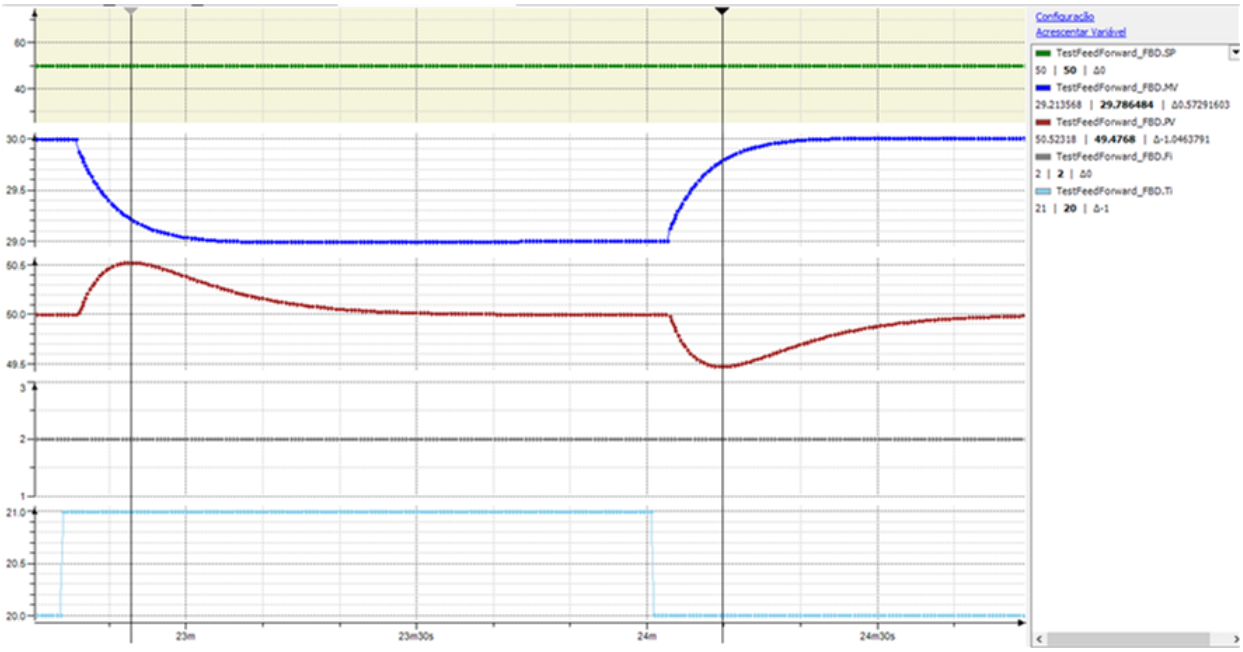

Figure 64: Effect of disturbances in Ti without feed-forward

16.8.4. Effects of Disturbances in Ti with Feed-Forward Controller

The following figure shows effect of changing Ti from 20 to 21, and after this from 21 to 20, with SP = 50, when feed-forward control is enabled.

When Ti increased from 20 to 21, PV increased from 50 to 50.019 before returning to 50 (less than 0.02 degrees).

When Ti decreased from 21 to 20, PV decreased from 50 to 49.980 before returning to 50 (less than 0.02 degrees).

So, without feed-forward controller, PV deviated about 0.5 degrees from SP, and with feed-forward, only 0.02 degrees.

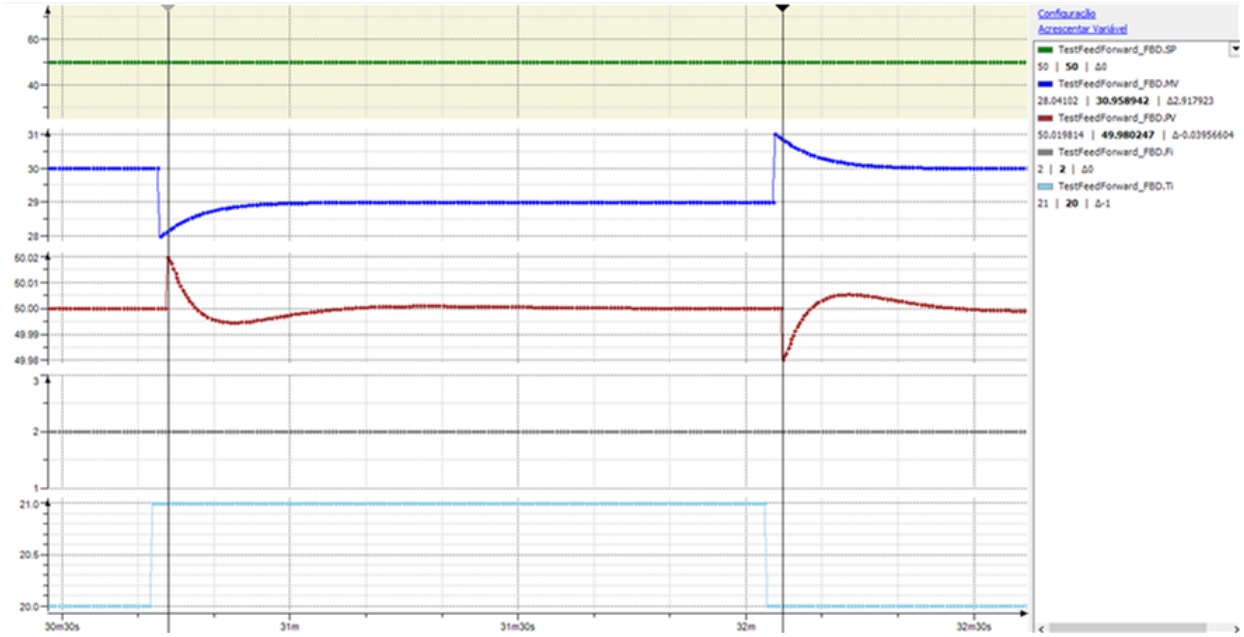

Figure 65: Effect of disturbances in Ti with feed-forward

16.8.5. Conclusions

Feed-forward control helped a lot in controlling the effect of disturbances in Ti and Fi, as demonstrated in previous figures. For implementing feed-forward, it is necessary to measure disturbances and have an approximate mathematical model on how they affect PV.

17. PID Tuning Tips

This chapter will present some basic tuning tips for PID controllers. Tuning PID controllers may be a complex task for some processes. So, keep in mind that these tips are very basic, and may not be useful for all kinds of processes.

These tips will focus linear first order processes with dead time. By the way, this kind of process can be simulated with the ControlDelayLag FB.

17.1. Open Loop Process Characterization

Several tuning methods provide formulas that use process parameters measured using an open loop process characterization. Considering a linear first order with dead time process, the parameters normally needed in the tuning formulas are the following:

- Gs: static gain
- Tc: time constant (seconds)
- Tde: dead time (seconds)

For measuring these parameters, the user normally applies a step input to the process using the MV output of PID controller in manual mode, and observes the response in PV.

The next figure shows the response of PV for a step in MV, in a linear first order with dead time process. The figure also shows how to measure Tde, Tc, and how to calculate Gs.

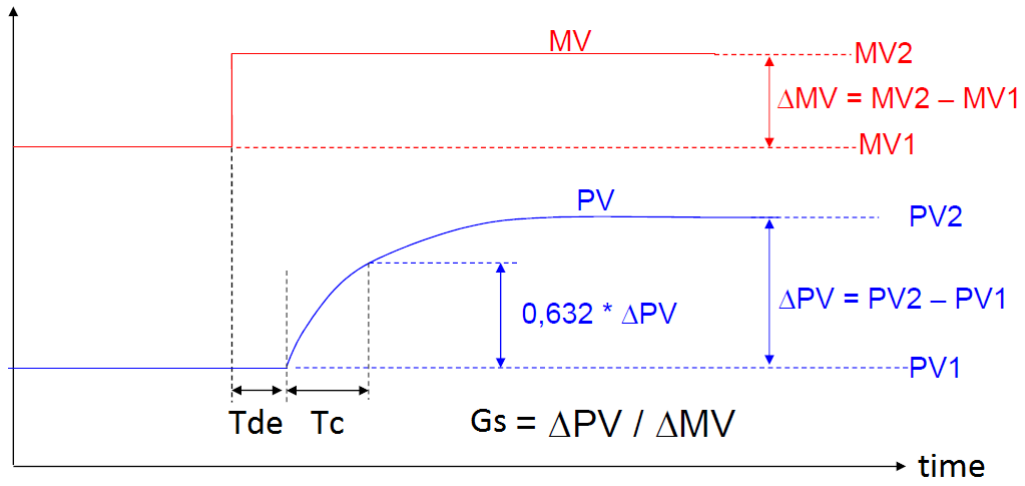

Figure 66: Open-loop process characterization

Observe that FB ControlDelayLag can be used to simulate this kind of process:

- Input Gs = Gs
- Input Tc = Tc
- Input DelayTime = Tde

17.2. Field and Engineering Units

Sometimes the user wants that PID controller manage field units, so that PV can by copied directly from the analog input (e.g.: scale 0 ... 30000), and MV can be copied directly over the analog output (e.g.: scale 0 ... 30000). In this case, the user should use a FB like PIDA_INT or PIDA_TUNE_INT. When PID controller uses field units, normally the SCADA system manages the conversion between field units and engineering units, so that SCADA operator only works with engineering units:

- SCADA converts engineering units to field units before writing values like SP in the PLC;
- SCADA converts field units to engineering units after reading values like PV and MV from the PLC.

Other times, the user wants that PID controller manage engineering units, so that the SCADA system writes and reads values to PLC directly in engineering units. In this case, the user should use a FB like PIDA_REAL or PIDA_TUNE_REAL. When PID controller uses engineering units the PLC manages the conversion between field units and engineering units:

- The reading from analog input must be converted from field units (e.g.: 0 ... 30000) to engineering units (e.g.: 0ºC to 500ºC), and then copied to PV;
- The output MV of PID FB in engineering units (e.g.: 0% to 100% of valve position) must be converted to field units (e.g.: 0 ... 30000) before copying to analog output.

ATTENTION

Regarding the measurement of static gain (Gs) discussed in section Open Loop Process Characterization, it is important to realize that it must be measured coherently with the type of units used by the PID controller, considering input PV and output MV. The value of Gs calculated using engineering units may be totally different from the value of Gs calculated using field units. So, if the PID controller manages engineering units, Gs must be measured using engineering units; otherwise, Gs must be measured using field units.

17.3. Controllability of Processes

The controllability of a process denotes the easiness to control it. A good way to express this controllability is the ratio between parameters Tc (time constant) and Tde (dead time).

The bigger is Tc/Tde, the easiest is the process control. Big time constants (Tc) mean slow processes that are easier to control. Big dead times (Tde) makes the process control difficult.

Generally a process becomes easy to control when:

$$Tc/Tde > 10 \tag{20}$$

17.4. Maximum Sample Time of Controller

The sample time is the period used to call the PID or ON-OFF controller.

A general hint is that sample time must be smaller than one tenth of the process time constant, that is:

$$SampleTime < Tc/10 \tag{21}$$

17.5. Selection of PID Modes

In a PID controller, it is possible to select individually which actions are enabled or disabled (P, I or D). So, it is possible to have several configurations, like P, PI, PID, I, etc. In some situations, it is also possible to use the ON-OFF controller instead of PID controller.

This section gives some hints about the configuration of PID controllers and the eventual selection of an ON-OFF controller.

First of all, it is important to realize that not always the objective of a PID controller is to keep the error (SP – PV) very close to zero. Sometimes relatively big errors can be accepted, and the main objective is to keep PV in an acceptable range around SP. This can happen in several processes, for instance:

- Level or pressure of intermediary tanks or reservoirs
- Environment temperature (e.g.: conditioning air)

In processes that accept bigger errors, two situations may arise:

1. The acceptable error is not too big, but the process is highly controllable (Tc / Tde > 10);
2. The acceptable error is really big.

In previous situation 1 (highly controllable process, but accepted error not too big):

- Use P or PD controllers with a big proportional gain. The high controllability allows a big proportional gain without oscillation;
- The integral action can be disabled, because the steady state error can be minimized using the big proportional gain;

- Derivative action may help stabilization, compensating dead times if they are relatively big. But remember that time constant must also be big in these cases, because the process must be highly controllable;
- In certain situations, the ON-OFF control can be used, since the time constant is very high, causing a low frequency for switching the MV output between states ON and OFF. In this case, it is possible to adjust a low DeadBand parameter to keep the error low (this is equivalent to a big proportional gain in a pure P controller).

In previous situation 2 (accepted error is big):

- Use P controllers with proportional gain as low as possible, provided this proportional gain is enough to keep the error inside the tolerance range;
- With a low proportional gain in a P controller, the risk of oscillations is low, even if the process controllability is low.

In other situations, the integral action is required, resulting in PI or PID controllers.

The derivative action helps stability when dead time (Tde) is relatively high. Derivative action normally helps in slow processes (big time constant) where overshoot is unacceptable. Derivative action must be avoided in fast processes.

17.6. Synthesis Method

This method calculates the tuning parameter from the process parameters (Gs, Tc, Tde) obtained according section Open Loop Process Characterization, and from a performance specification of the desired time constant in closed loop (Tcl).

With closed loop, that is, with PID controlling the process (automatic mode), the time constant (Tcl) can be smaller than time constant in open loop (Tc). The concept of closed loop time constant corresponds to the time needed for PV travelling 63% of a commanded SP variation. For instance, consider that SP and PV were initially at 50%, and then SP was changed to 60%. Tcl is the time needed for PV travelling from 50% to 56.3%.

Note that dead time (Tde) must be discounted in measurements of Tc and Tcl.

The formulas for synthesis method are the following:

$$Kp = Tc/(Gs * (Tcl + Tde + SampleTime/2)) \tag{22}$$

$$Ti = Tc \tag{23}$$

$$Td = (Tde + SampleTime/2)/2 \tag{24}$$

SampleTime corresponds to the period used for calling the PID FB. The term “SampleTime / 2” was added to dead time because it inserts a delay similar to a dead time. The average delay inserted is half SampleTime.

Note that the smaller is the required Tcl, the bigger will be gain Kp. If the user requires Tcl much smaller than Tc, this may result in an excessive gain Kp that can cause unstable control (oscillations).

Consider an example of process with Gs = 2, Tc = 10 and Tde = 1, simulated with ControlDelayLag FB (parameter Offset = 0).

The following figure shows a step in MV from 0 to 25 in manual mode, resulting in PV going from 0 to 50. In this figure, the two vertical cursors delimit the dead time of 1 second, between the moment MV stepped from 0 to 25, and the moment PV started to move up.

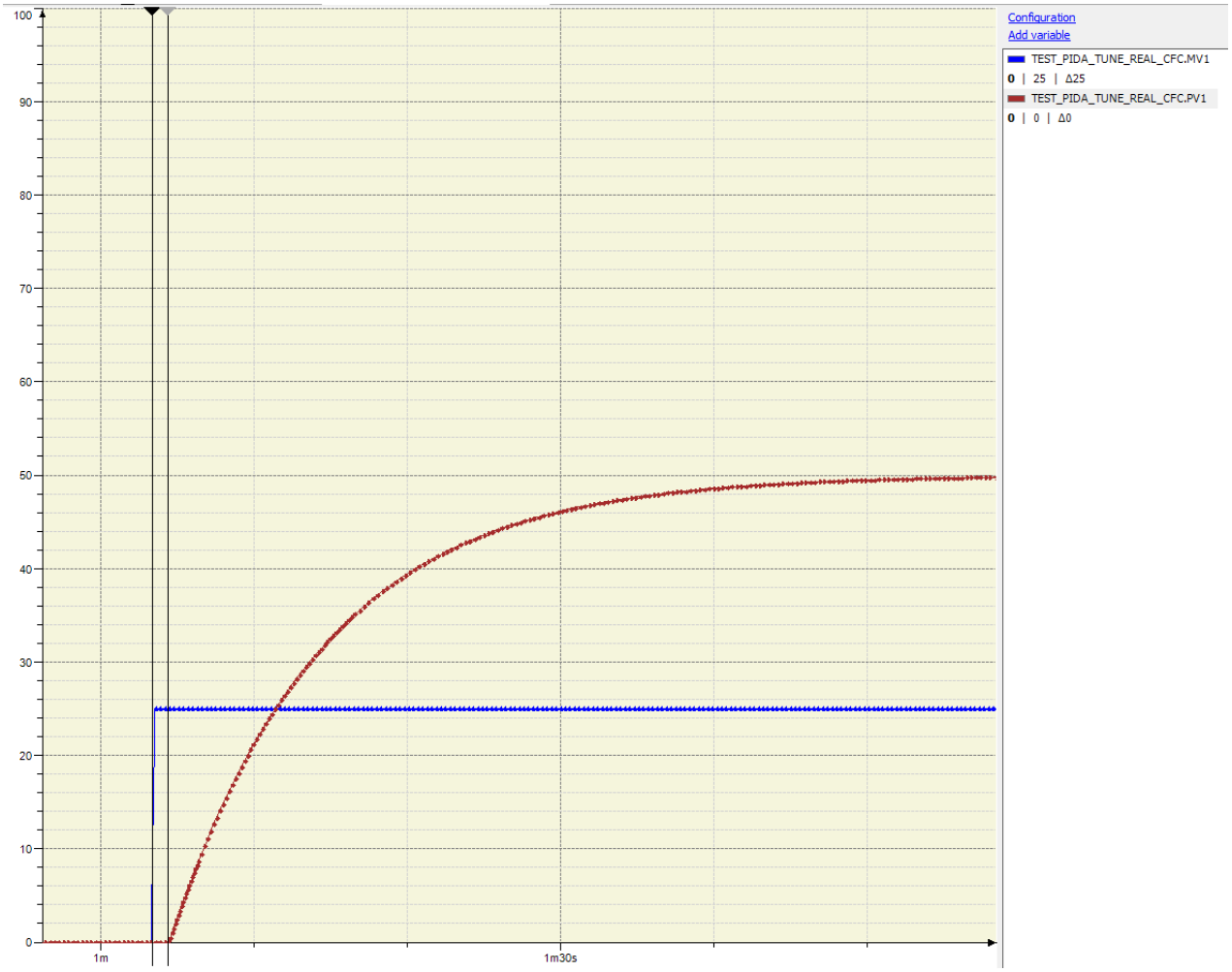

Figure 67: Open-loop characterization of process - dead time

In the next figure (same as above), the two vertical cursors delimit the constant time of 10 seconds, between the moment PV started to move up until the moment it travelled 63% of span (PV = 31.5 = 63% of 50). In steady state (PV = 50), observe that $\Delta PV = 50$ and $\Delta MV = 25$, so $Gs = \Delta PV / \Delta MV = 2$.

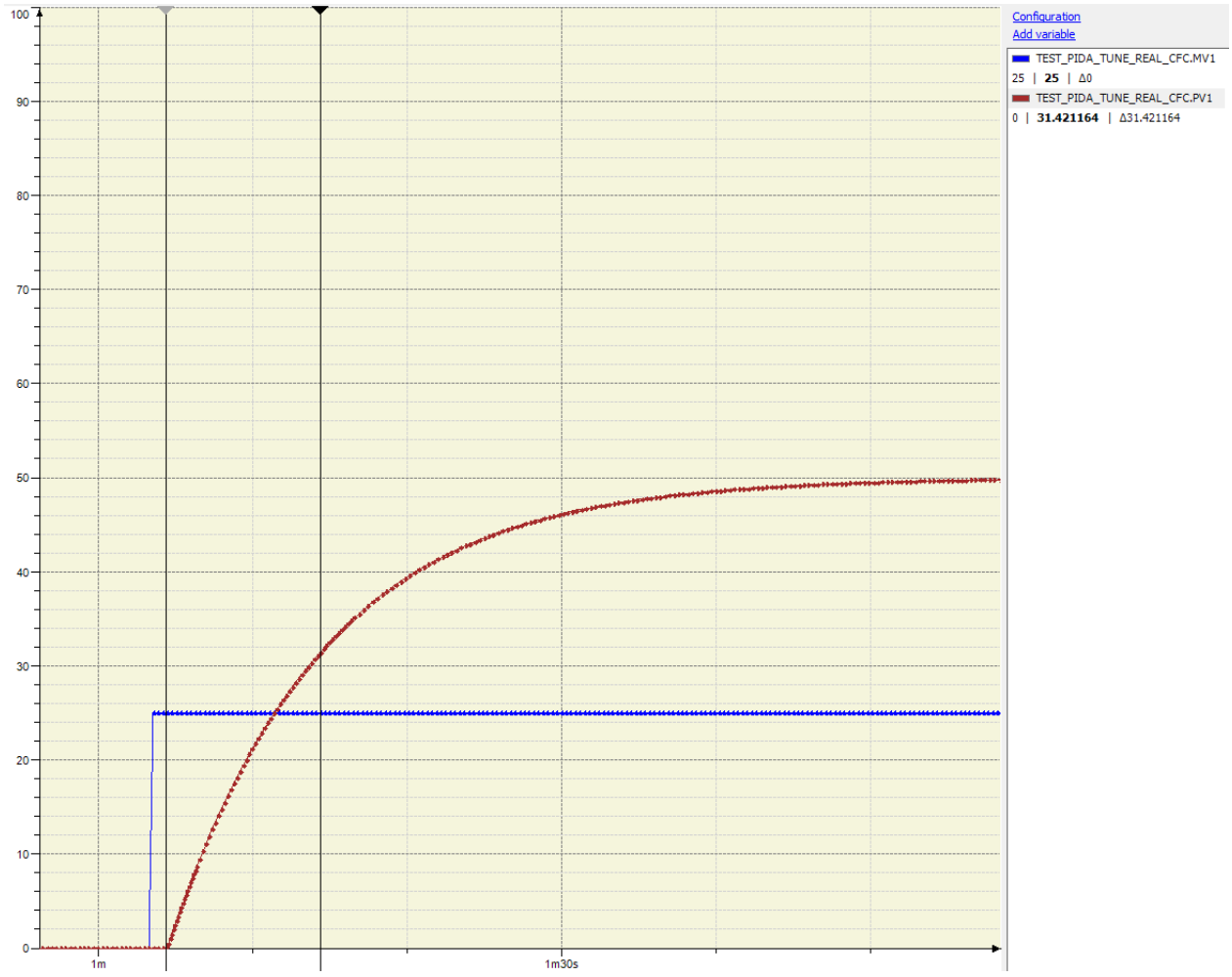

Figure 68: Open-loop characterization of process - time constant

Now, consider the application of the synthesis method, establishing a closed loop time constant of 4 seconds (Tcl = 4), and PID FB being called with SampleTime = 0.1 seconds. The calculated parameters are:

- Kp = 0.99
- Ti = 10
- Td = 0.525

Now let's make SP change from 0 to 10 in automatic mode. The following figure was obtained. The time measured between the vertical cursors corresponds to the closed loop constant time Tcl (measured was 4.4 seconds, and theoretically should be 4 seconds). Note also that the dead time of 1 seconds still appears between the moment SP/MV were changed until PV started to move up.

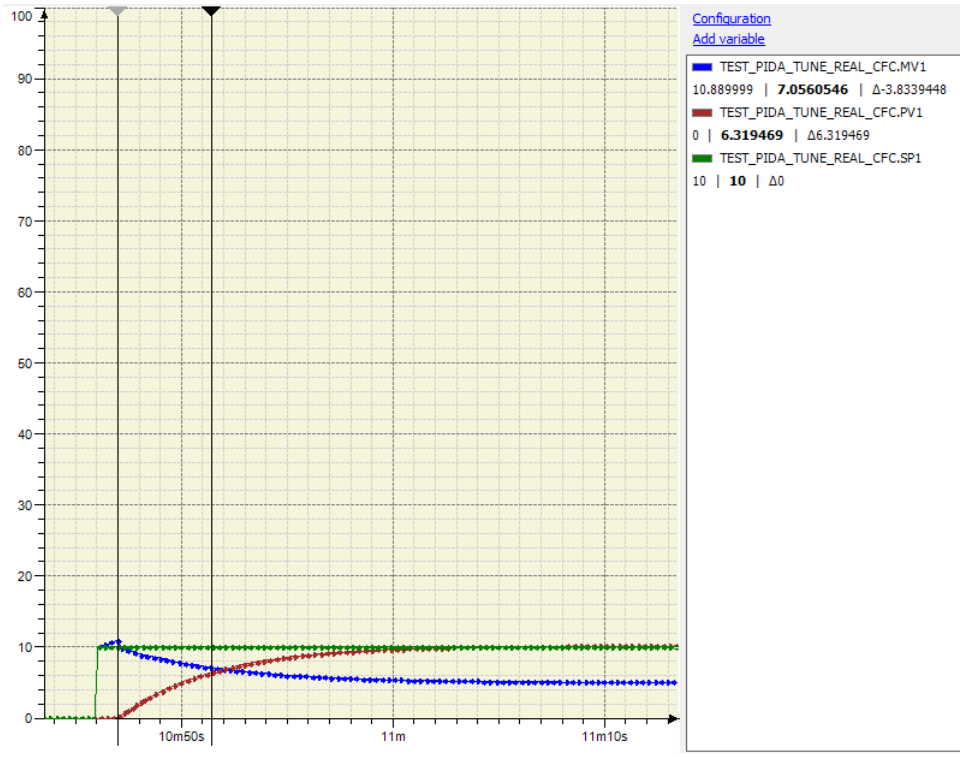

Figure 69: Automatic mode - closed loop time constant

17.7. Minimization of Integral of Absolute Error

This tuning method aims to minimize the integral of absolute value of error (SP - PV).

It calculates the tuning parameter from the process parameters (Gs, Tc, Tde) obtained according section Open Loop Process Characterization.

In the following formulas, consider that Pu (parameter of uncontrollability) is:

$$Pu = (Tde + SampleTime/2)/Tc \tag{25}$$

The formulas are divided in two groups, shown in two following tables:

Formulas optimized for disturbances that change the process; Formulas optimized for setpoint change.

For selecting PID configurations (P, PI and PID), see section Selection of PID Modes.

PID Configuration	Kp	Ti	Td
P	(0.902 / Gs) * $Pu^{-0.985}$	-	-
PI	(0.984 / Gs) * $Pu^{-0.985}$	(Tc / 0.608) * $Pu^{0.707}$	-
PID	(1.435 / Gs) * $Pu^{-0.921}$	(Tc / 0.878) * $Pu^{0.749}$	0.482 * Tc * $Pu^{1.137}$

Table 2: Optimized response to disturbances

PID Configuration	Kp	Ti	Td
PI	(0.758 / Gs) * $Pu^{-0.861}$	Tc / (1.02 – 0.323 * Pu)	-
PID	(1.435 / Gs) * $Pu^{-0.869}$	Tc / (0.74 – 0.13 * Pu)	0.348 * Tc * $Pu^{0.914}$

Table 3: Optimized response to setpoint changes

Consider the same example of process used in previous section about synthesis method (Gs = 2, Tc = 10 and Tde = 1), simulated with ControlDelayLag FB (parameter Offset = 0).

Let's use a PID configuration and group 2 (optimization for response to setpoint changes). The results are:

- Kp = 3.85
- Ti = 13.77
- Td = 0.44

Now let's make SP change from 0 to 10 in automatic mode. The following figure is obtained.

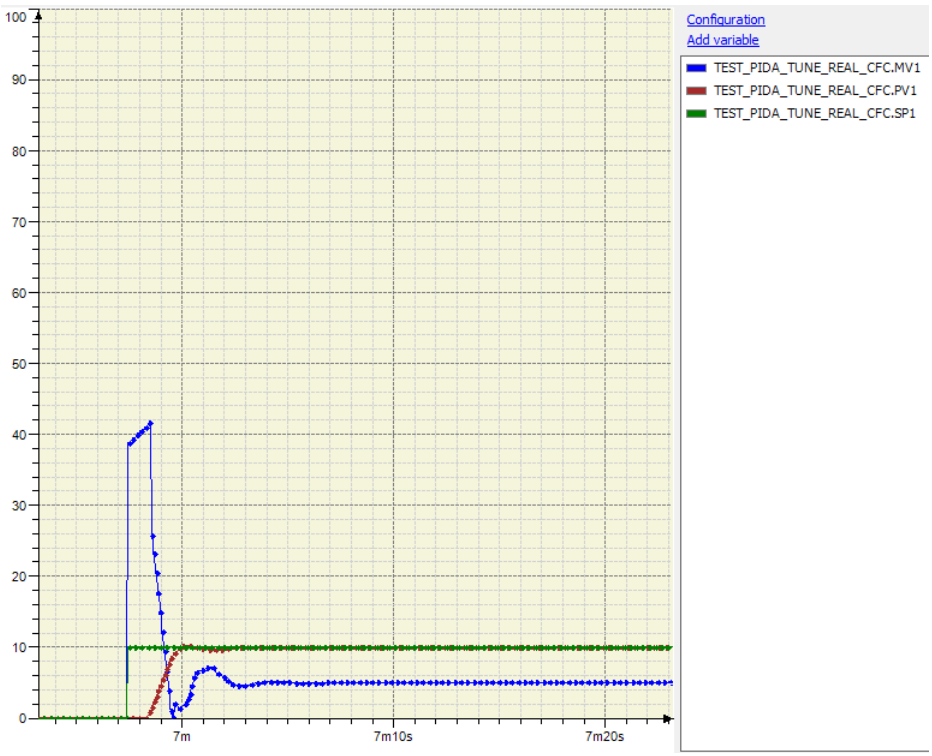

Figure 70: Tuning with minimization of integral of error for setpoint changes

17.8. Hints for Tuning PID in Cascade Configuration

For tuning cascade configuration, some hints are necessary. Let's consider a cascade configuration made of 2 PIDs.

- First tune the inner control loop (the slave PID controller). Make CASC = FALSE in the slave PID while tuning it.
- After the slave PID controller is tuned, tune the master PID. Make CASC = TRUE in the slave PID. Observe that the master controller see its process as the series association of three entities:
  - Slave PID
  - Process controlled by slave PID
  - Process controlled by master PID

The methods for tuning each PID (master and slave) can be one of those discussed in previous sections.

ATTENTION

See section Cascade Configurations inside chapter Application Notes.