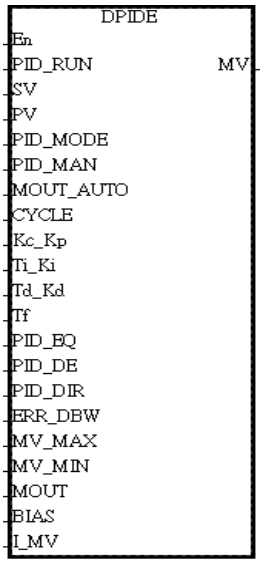
API	Instruction code			Operand	Function
0708	D	PIDE		**As shown in the following table**	PID algorithm

Device	X	Y	M	S	T	C	HC	D	FR	SM	SR	E	K	16#	"$"	F
PID_RUN	●	●	●	●				●								
SV								●								○
PV								●								○
PID_MODE								●					○	○		
PID_MAN	●	●	●	●				●								
MOUT_AUTO	●	●	●	●				●								
CYCLE								●					○	○		
KC_Kp								●								
Ti_Ki								●								
Td_Kd								●								
Tf								●								○
PID_EQ	●	●	●	●				●								
PID_DE	●	●	●	●				●								
PID_DIR	●	●	●	●				●								
ERR_DBW								●	●							○
MV_MAX								●	●							○
MV_MIN								●	●							○
MOUT								●								
BIAS								●	●							○
I_MV								●								
MV								●								

Data type	BOOL	WORD	DWORD	LWORD	UINT	INT	DINT	LINT	REAL	LREAL	TMR	CNT	STRING

Pulse instruction	16-Bit instruction	32-Bit instruction
-	-	AS

Symbol



EN	:	Enable/Disable the instruction
PID_RUN	:	Enable the PID algorithm
SV	:	Target value (SV)
PV	:	Process value (PV)
PID_MODE	:	PID control mode
PID_MAN	:	PID Auto/Manual mode
MOUT_AUTO	:	Manual/Auto output value
CYCLE	:	Sampling time (CYCLE)
Kc_Kp	:	Proportional gain
Ti_Ki	:	Integral coefficient (sec. or 1/sec)
Td_Kd	:	Derivative coefficient (sec)
Tf	:	Derivate-action time constant (sec)
PID_EQ	:	PID formula types
PID_DE	:	Calculation of the PID derivative error
PID_DIR	:	PID forward/reverse direction (PID_DIR)
ERR_DBW	:	Range within which the error value is counted as 0
MV_MAX	:	Maximum output value (MV_MAX)
MV_MIN	:	Minimum output value (MV_MIN)
MOUT	:	Manual output value (MOUT)
BIAS	:	Feed forward output value
I_MV	:	Accumulated integral value
MV	:	Output value (MV)

Explanation

1. This instruction implements the PID algorithm. After the sampling time is reached, the instruction applies PID algorithm. PID stands for Proportional, Integral, Derivative. The PID control is widely applied to mechanical, pneumatic, and electronic equipment.
2. The parameter settings are listed in the following table.

Operand	Data type	Function	Setting range	Description
PID_RUN	BOOL	Enabling the PID algorithm	True: use the PID algorithm. False: reset the output value (MV) to 0, and stop using the PID algorithm.	
SV	REAL	SV	Range of single-precision floating-point numbers	Target value
PV	REAL	PV	Range of single-precision floating-point numbers	Process value
PID_MODE	DWORD/DINT	PID control mode	0: Automatic control When PID_MAN switches from True to False, invoke the output value (MV) in the automatic algorithm. If the MV exceeds the limits of MV_MAX or MV_MIN, I_MV value stays the same and without updating. 1: Auto tuning the parameters for the temperature control. After tuning is done, the system is in auto control mode (PID_MODE is set to 0) and fill in the appropriate parameters (Kc_Kp, Ti_Ki, Td_Kd, and Tf) 2: Automatic control (including I_MV): When PID_MAN switches from TRUE to FALSE, the MV value is invoke the output value (MV) in the	

Operand	Data type	Function	Setting range	Description
			automatic algorithm. If the MV exceeds the limits of MV_MAX or MV_MIN, I_MV use the formula to calculate the correct value and adjust its value accordingly. This can reduce the response time of the reverse action. Note: when the mode is set to 1, auto tuning the parameter, you cannot use numerical value to set up.	
PID_MAN	BOOL	PID A/M mode	True: Manual Output the MV according to MOUT, but it is still between MV_MIN and the MV_MAX. This setting has no effect when PID_MODE is set to 1. False: Automatic Output the MV according to the PID algorithm, and the output value is between MV_MIN and MV_MAX.	
MOUT_AUTO	BOOL	MOUT automatic change mode	True: Automatic MOUT varies with the MV. False: Normal MOUT does not vary with the MV.	
CYCLE	DWORD/DINT	Sampling time (T_S)	1–40,000 (unit: ms)	When the instruction is scanned, use the PID algorithm according to the sampling time, and refresh MV. The PLC requires that the instruction execute; it will not run the

Operand	Data type	Function	Setting range	Description
				sampling time automatically. If T_S is less than 1, it is counted as 1. If T_S is larger than 40,000, it is counted as 40,000. When using the PID instruction in an interval interrupt task, the sampling time is the same as the interval between the timed interrupt tasks. The sampling cycle setting of the sampling cycle is ignored here.
Kc_Kp	REAL	Calculated proportional coefficient (Kc or Kp, according to the settings in PID_EQ)	Range of positive single-precision floating-point numbers	Calculated proportional coefficient (Kc or Kp) If the P coefficient is less than 0, the Kc_Kp is 0. Independently, if Kc_Kp is 0, it is not controlled by P.
Ti_Ki	REAL	Integral coefficient (Ti or Ki, according to the settings in PID_EQ)	Range of positive single-precision floating-point numbers (unit: Ti = sec; Ki = 1/sec)	If the calculated coefficient I is less than 0, Ti_Ki is 0. If Ti_Ki is 0, it is not controlled by I.

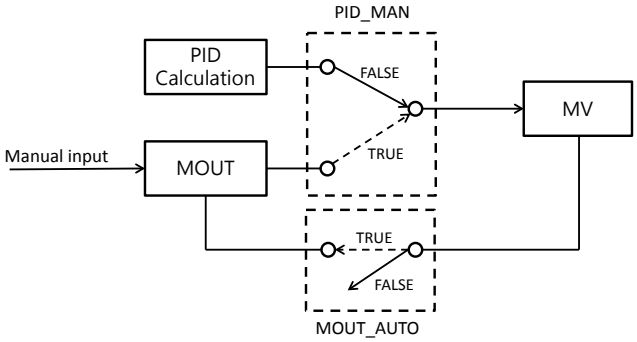
Operand	Data type	Function	Setting range	Description
Td_Kd	REAL	Derivative coefficient (Td or Kd, according to the settings in PID_EQ)	Range of positive single-precision floating-point numbers (unit: sec)	If the calculated coefficient D is less than 0, Td_Kd is 0. If Ti_Ki is 0, it is not controlled by D.
Tf	REAL	Derivate-action time constant	Range of positive single-precision floating-point numbers (unit: sec)	If the derivate-action time constant is less than 0, Tf is 0 and it is not controlled by the derivate-action time constant (derivative smoothing).
PID_EQ	BOOL	PID formula types	TRUE: dependent formula FALSE: independent formula	
PID_DE	BOOL	The calculation of the PID derivative error	TRUE: use the variations in the PV to calculate the control value of the derivative (Derivative of the PV). FALSE: use the variations in the error (E) to calculate the control value of the derivative (derivative of the error).	
PID_DIR	BOOL	PID forward/reverse direction	True: forward action (E=SV-PV) False: reverse action (E=PV-SV)	
ERR_DBW	REAL	Range within which the error value is counted as 0.	Range of single-precision floating-point numbers	The error value (E) is the difference between the SV and the PV. When the setting value is 0, the function disabled; otherwise the CPU

Operand	Data type	Function	Setting range	Description
				module checks whether the present error is less than the absolute value of ERR_DBW, and checks whether the present error meets the cross status condition. If the present error is less than the absolute value of ERR_DBW, and meets the cross status condition, the present error is counted as 0, and the PLC applies the PID algorithm ; otherwise the present error is brought into the PID algorithm according to the normal processing.
MV_MAX	REAL	Maximum output value	Range of single-precision floating-point numbers	Suppose **MV_MAX** is set to 1,000. When MV is larger than 1,000, 1,000 is the output. The value in **MV_MAX** should be larger than that in **MV_MIN**. Otherwise, the maximum MV and the minimum MV are reversed.
MV_MIN	REAL	Minimum output value	Range of	Suppose **MV_MIN** is

Operand	Data type	Function		Setting range	Description
				single-precision floating-point numbers	set to -1,000. When the MV is less than -1,000, -1,000 is the output.
MOUT	REAL	MV		Range of single-precision floating-point numbers	When set to PID Manual, the MV value is output as the setting value for MOUNT, between MV_MAX and MV_MIN.
BIAS	REAL	Feed forward output value		Range of single-precision floating-point numbers	Feed forward output value, used for the PID feed forward.
I_MV (occupies 15 consecutive DWord devices)	REAL	I_MV	Accumulated integral value	Range of single-precision floating-point numbers	Accumulated integral value temporarily stored, and usually for reference. You can still clear or modify it according to your needs. PID mode in 0 (Automatic control): When the MV is greater than the MV_MAX, or when the MV is less than MV_MIN, the accumulated integral value in I_MV is unchanged. PID mode in 2

Operand	Data type	Function		Setting range	Description
					(Automatic control, including I_MV): When PID_MAN switches from TRUE to FALSE, the MV value is invoke the output value (MV) in the automatic algorithm. If the MV exceeds the limits of MV_MAX or MV_MIN, I_MV use the formula to calculate the correct accumulated integral value and adjust its value accordingly. This can reduce the response time of the reverse action.
		I_MV+1	The previous error value is temporarily stored here.		
		I_MV+2– I_MV+5	For system use only		
		I_MV+6	The previous PV is temporarily stored here.		
		I_MV+7– I_MV+14	For system use only		
MV	REAL	MV	The MV is between the MV_MIN and the MV_MAX.		

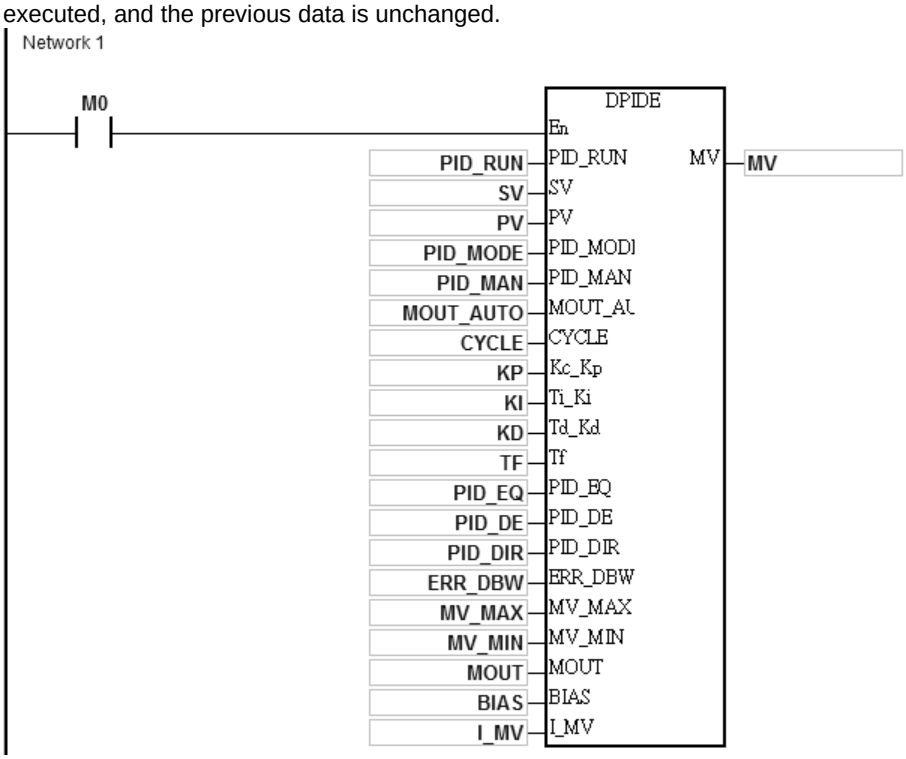
The diagram of switching to PID_MAN / MOUT_AUTO:



1. When switching the control mode **(PID_MAN=0)** from automatic to manual, you can set the flag MOUT_AUTO to 1 and the output value of MOUT goes along with the output value of MV. After switching to the manual mode **(PID_MAN=1)**, you can set the MOUT_AUTO to 0.
2. When **PID_RUN** changes from TRUE to FALSE, the PLC resets the value in MV to 0. When the value in MV is to be retained, you can set EN to FALSE to dismiss the instruction and to keep the output value in MV.

Example 1

1. Set all parameters before executing this instruction.
2. When M0 is ON, the instruction is executed. When PID_RUN is ON, the instruction applies the DPID algorithm. When PIC_RUN is OFF, MV is 0, and store the value in MV. When M0 switches to OFF, the instruction is not executed, and the previous data is unchanged.



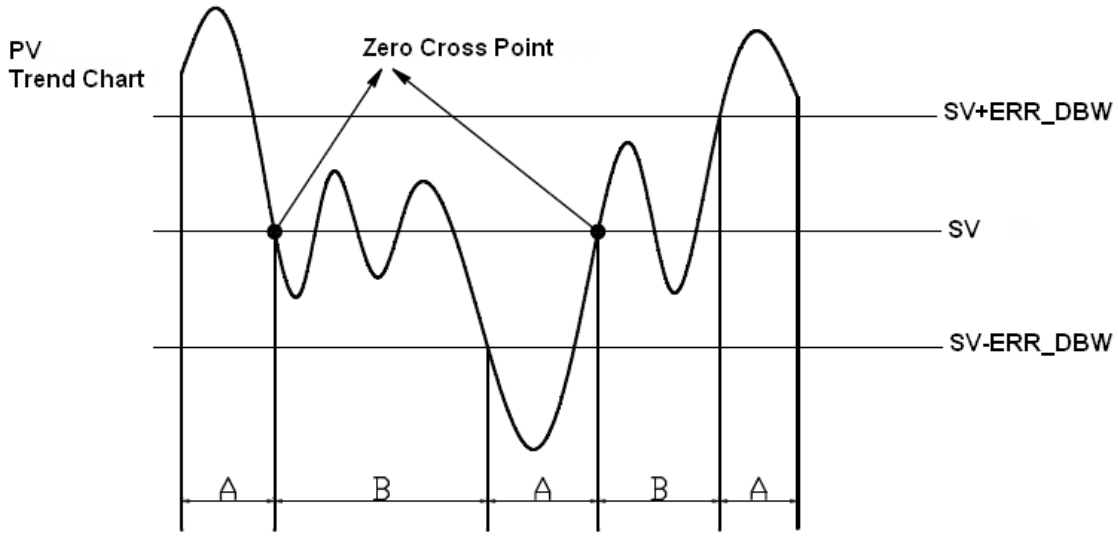
Additional remarks

1. The instruction can be used several times, but the registers specified by **I_MV**–**I_MV**+14 cannot be the same.
2. **I_MV** occupies 30 registers.
3. You can only use the 32-bit instruction in cyclic tasks and interval interrupt tasks. When using the 32-bit instruction in an interval interrupt task, the sampling time (Cycle) is the same as the interval between the timed interrupt tasks.
4. When the instruction is scanned, the 32-bit PID algorithm is applied according to the sampling time (Cycle), and it refreshes MV. When you use the instruction in an interrupt task, the sampling time (Cycle) is the same as the

interval between the timed interrupt tasks. The PID algorithm is applied according to the interval between the timed interrupt tasks.

5. Before the 32-bit PID algorithm is applied, the process value used in the PID instruction has to be a stable value. When you need the input value in the module to implement the DPID algorithm, must note the time it takes for the analog input to be converted into the digital input.

6. When the PV (process value) is in the range of **ERR_DBW**, at the beginning, the present error is brought into the PID algorithm according to the normal processing, and then the CPU module checks whether the present error meets the cross status condition: PV (process value) goes beyond the SV (target value). Once the condition is met, the present error is counted as 0 when applying the PID algorithm. After the PV (process value) is out of the **ERR_DBW** range, the present error is brought into the PID algorithm again. If PID_DE is true, that means it uses the variations in the PV to calculate the control value of the derivative, and after the cross status condition is met, the PLC treats Δ **PV** as 0 to apply the PID algorithm. (Δ **PV**= current **PV** – previous **PV**). In the following example, the present error is brought into the PID algorithm according to the normal processing in section A ,and the present error or Δ **PV** is counted as 0 to apply the PID algorithm in the section B.



The PID algorithm:

1. When you set **PID_MODE** to 0, the PID control mode is the automatic control mode.

- **Independent Formula & Derivative of E** (**PID_EQ**=False & **PID_DE**=False)

$$MV = K_p E + Ki\int_0^t Edt + K_d * \frac{dE}{dt} + BIAS \quad E = SV - PV \text{ or } E = PV - SV$$

- **Independent Formula & Derivative of PV** (**PID_EQ=False & PID_DE=True**)

$$MV = K_p E + Ki\int_0^t Edt - K_d * \frac{dPV}{dt} + BIAS \quad E = SV - PV$$

Or

$$MV = K_p E + Ki\int_0^t Edt + K_d * \frac{dPV}{dt} + BIAS \quad E = PV - SV$$

- **Dependent Formula & Derivative of E** (**PID_EQ**=True & **PID_DE**=False)

$$MV = K_c\left[E + \frac{1}{T_i}\int_0^t Edt + T_d * \frac{dE}{dt}\right] + BIAS \quad E = SV - PV \text{ or } E = PV - SV$$

- **Dependent Formula & Derivative of PV** (**PID_EQ**=True & **PID_DE**=True)

$$MV = K_c\left[E + \frac{1}{T_i}\int_0^t Edt - T_d * \frac{dE}{dt}\right] + BIAS \quad E = SV - PV$$

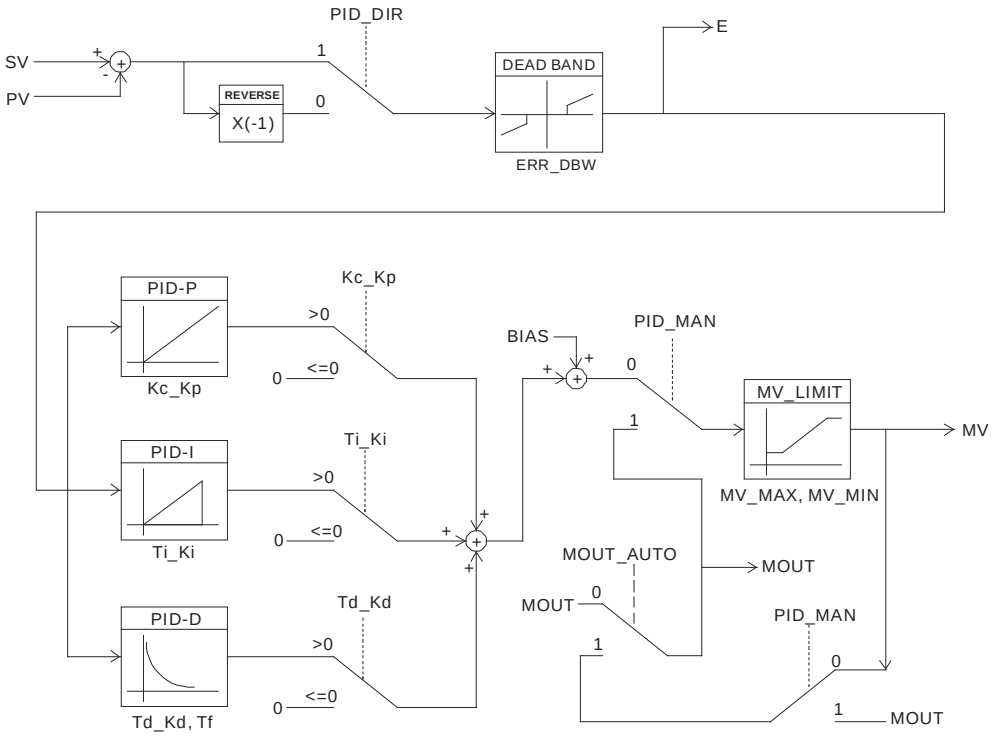
Or

$$MV = K_c\left[E + \frac{1}{T_i}\int_0^t Edt + T_d * \frac{dE}{dt}\right] + BIAS \quad E = PV - SV$$

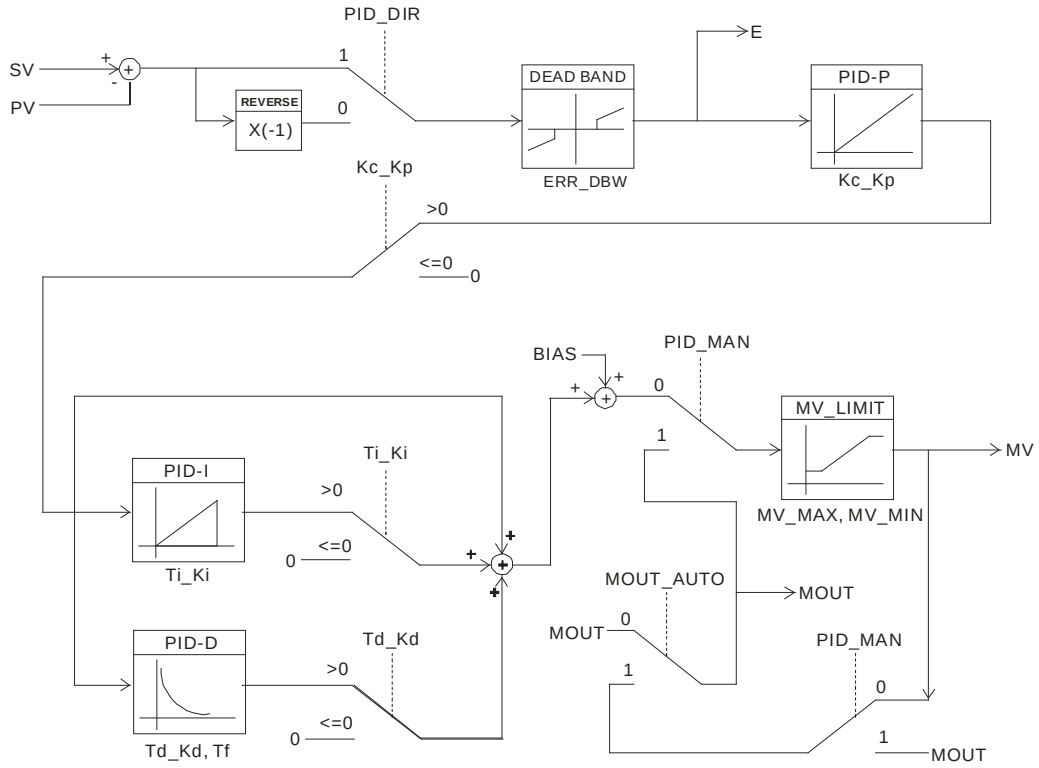
2. When you set **PID_MODE** to 1, the PID control mode is the automatic tuning mode. After the tuning of the parameter is complete, **PID_MODE** is set to 0. The PID control mode then becomes the automatic control mode.

PID Block Diagram:

PID Block Diagram (Independent)



PID Block Diagram (Dependent)



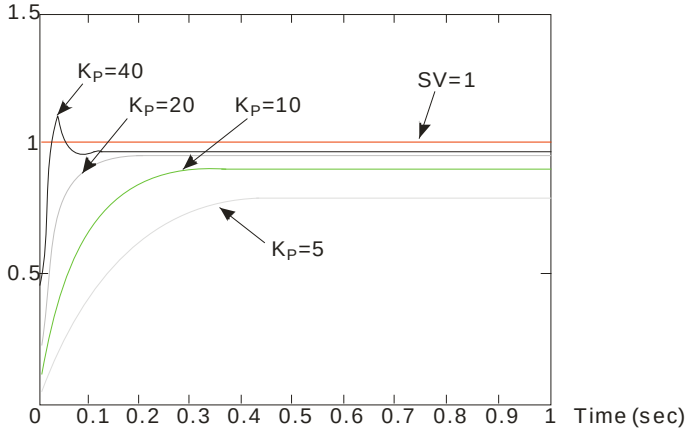
Suggestions

1. Since you can use the 32-bit instruction in a lot of controlled environments, you must choose the appropriate control function. For example, to prevent improper control, do not use **PID_MODE** in the motor controlled environment when it is set to 1.
2. When you tune the parameters **Kc_Kp**, **Ti_Ki**, and **Td_Kd** (PID_MODE is set to 0), you must tune KP first (based on experience), and then set **Ti_Ki** and **Td_Kd** to 0. When you can handle the control, you can increase **Ti_Ki** and **Td_Kd**. When **Kc_Kp** is 1, it means that the proportional gain is 100%. That is, the error value is increased by a factor of one. When the proportional gain is less than 100%, the error value is decreased. When the proportional gain is larger than 100%, the error value is increased.
3. To prevent the parameters that have been tuned automatically from disappearing after a loss of power, you must store the parameters in the latched data registers when **PID_MODE** is set to 1. The parameters that have been automatically tuned are not necessarily suitable for every controlled environment. Therefore, you can modify the automatically tuned parameters; however, it is suggested that you only modify the **Ti_Ki** and the **Td_Kd.**
4. You can use this instruction with many parameters, but to prevent improper control, do not set the parameters randomly.

Example 2: Tuning the parameters used with the PID instruction

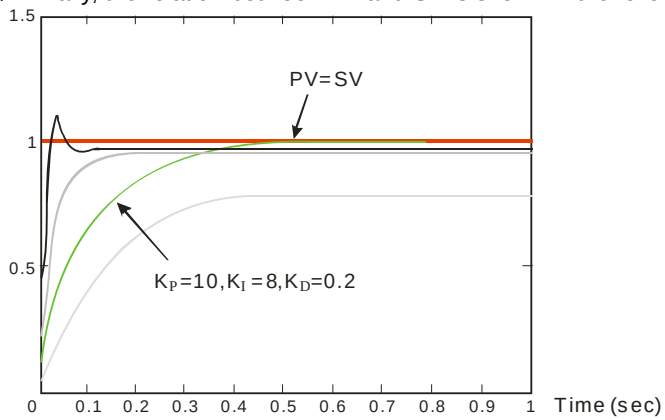
Suppose that the transfer function of the plant is the first-order function $G(s) = \frac{b}{s+a}$, the SV is 1, the sampling time Ts is 10 milliseconds. It is suggested that you follow these steps when tuning the parameters.

Step 1: First, set the K_I and the K_D to 0. Next, set the K_P to 5, 10, 20 and 40 successively, and record the target values and the process values. The results are shown in the following diagram.



Step 2: When the K_P is 40, there is overreaction. When the K_P is 20, the reaction curve of PV is close to SV, and there is no overreaction. However, due to the fast start-up, the transient output value (MV) is big. Neither 40 nor 20 is a suitable value. When the K_P is 10, the reaction curve of PV approaches SV smoothly. When K_P is 5, the reaction is too slow. Therefore, K_P = 10 is the best choice.

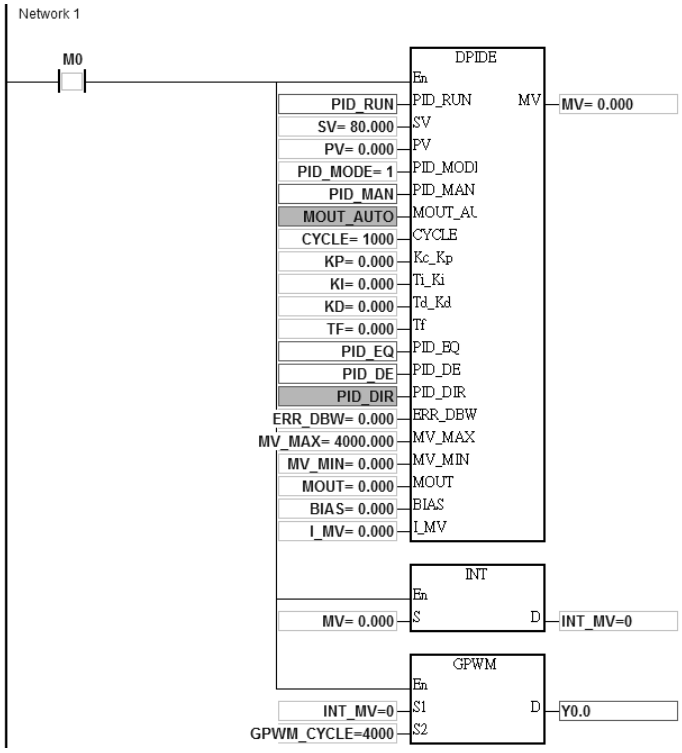
Step 3: After setting K_P to 10, increase K_I. For example, K_I is successively set to 1, 2, 4, and 8. K_I should not be larger than K_P. Then, increase K_D. For example, successively set K_D to 0.01, 0.05, 0.1, and 0.2. K_D should not be larger than ten percent of K_P. Finally, the relation between PV and SV is shown in the following diagram.



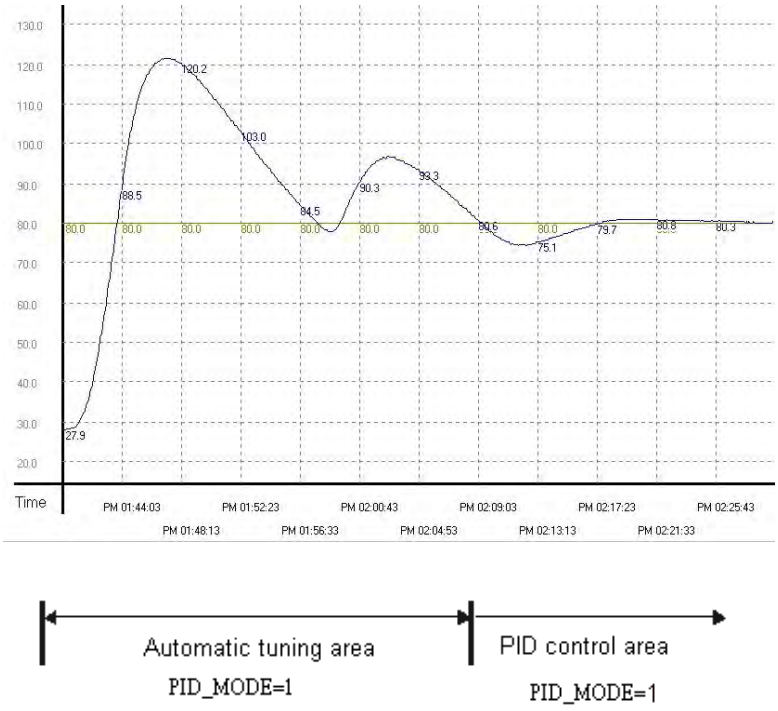
Note: This example is only for reference. You must tune the parameters properly according to the actual condition of the control system.

Example 3: Using the automatic tuning function to control the temperature

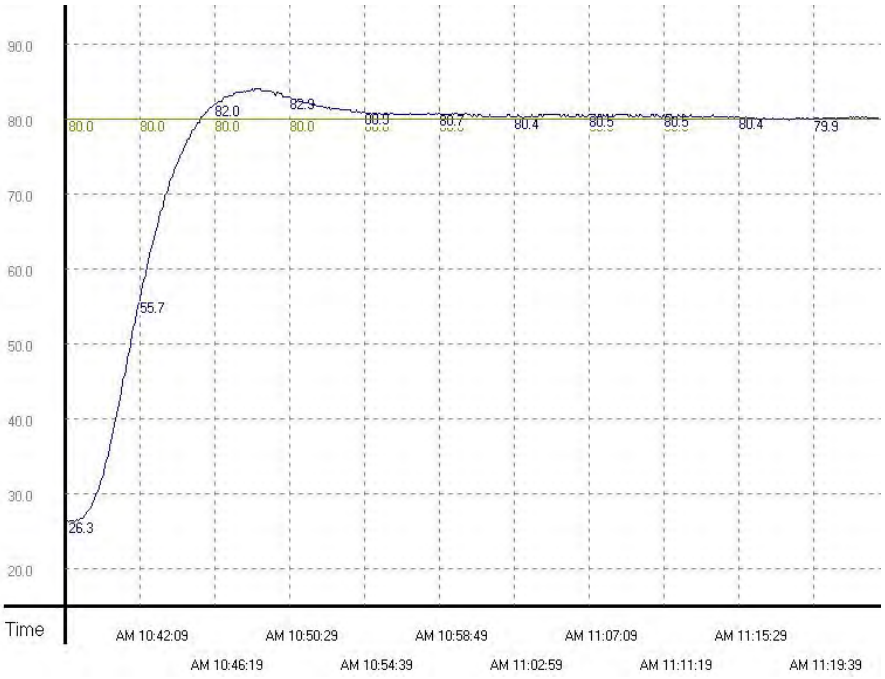
Because you may not be familiar with the characteristics of the temperature environment to be controlled, you can use the automatic tuning function to make an initial adjustment (**PID_MODE** is set to 1). After the automatic tuning of the parameter is complete, **PID_MODE** is set to 0. The controlled environment in this sample is an oven. The following example program shows the setting values for the instruction.



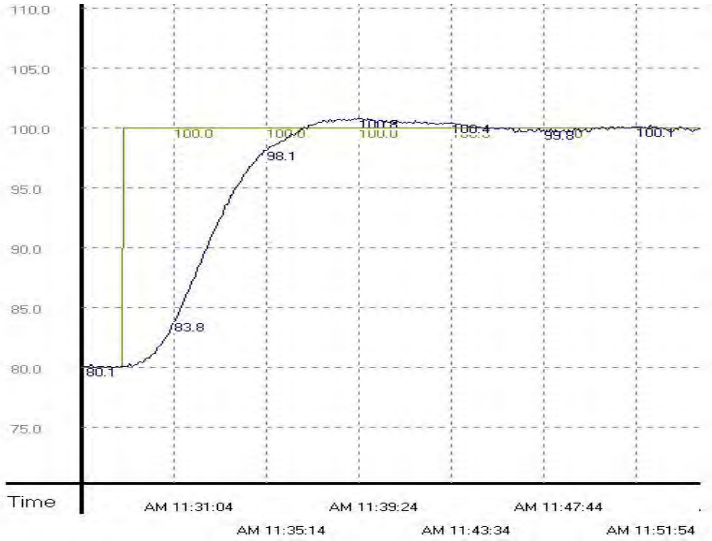
The experimental result of the automatic tuning function is shown in the following graph.



The following graph shows the result of using the automatically tuned parameters to control the temperature.



This graph shows that using automatically tuned parameters can result in a good temperature control result. It only takes about twenty minutes to control the temperature. The following graph shows the result of changing the target temperature from 80°C to 100°C.



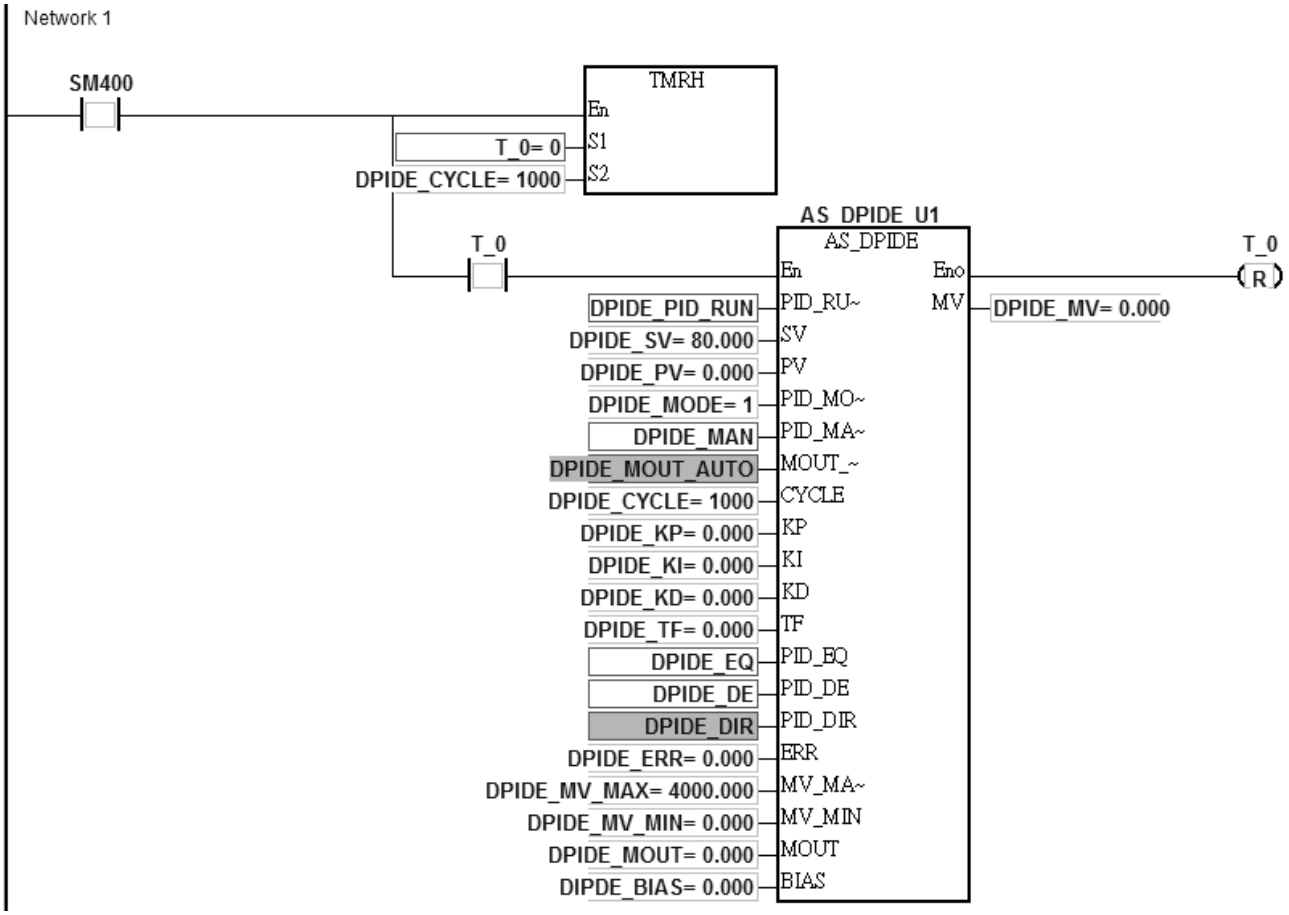
This graph shows that when the target temperature changes from 80°C to 100°C, the automatically tuned parameters still work to control the temperature in a reasonable amount of time.

Example 4: Creating a DPIDE instruction in a function block and setting to the cyclic task mode to read the function block written with a DPIDE instruction to control the temperature.

1. Set the value in DPIDE_CYCLE to 1000 ms, and execute the DPIDE instruction by reading the function block written with a DPIDE instruction. Whenever the function block is scanned, the PID algorithm is applied according to the sampling time (Cycle), and it refreshes the output value (DPIDE_MV).

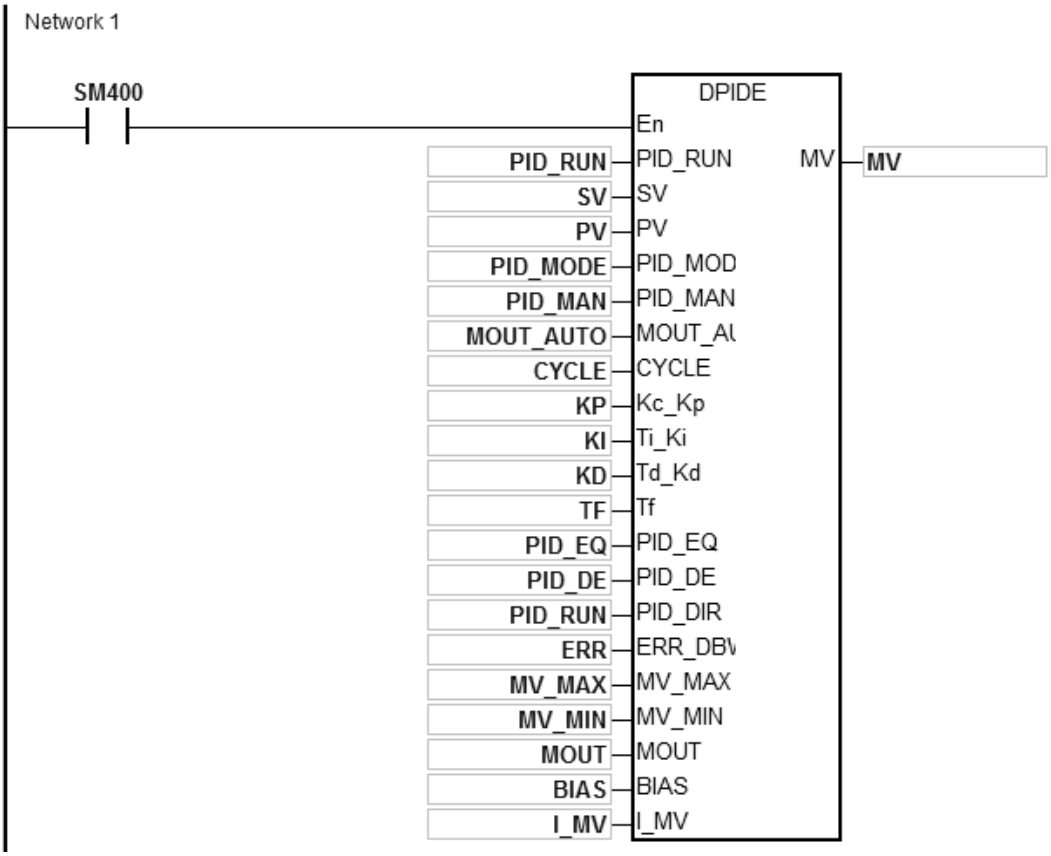
2. Set the DPIDE_MODE =1 for auto tuning the parameters for the temperature control. After tuning is done, the system is in auto control mode (PID_MODE is set to 0) and fill in the appropriate parameters (Kc_Kp, Ti_Ki, Td_Kd, and Tf).

3. Main program (cyclic task): Since PLC only executes the DPIDE instruction when it is scanned. If we use TMRH to work with the DPIDE instruction, for example, set the TMRH to 1000 ms, the system calls the function block written with a DPIDE instruction (AS_DPIDE) every 1000 ms. See the example program below.



4. Function block (AS_DPIDE): Execute the DPIDE instruction by reading the function block written with a DPIDE instruction. Whenever the function block is scanned, the PID algorithm is applied according to the sampling time (Cycle), and it refreshes the output value (DPIDE_MV). (Refer to ISPSoft Manual for more details on how to create a function block.)

NOTE: The parameters PID_MODE, Kc_Kp, Ti_Ki, Td_Kd, Tf and I MV in the function block written with a DPIDE instruction should be declared as VAR_IN_OUT.



Example 5: Creating a DPIDE instruction in a time interrupt program to control the temperature. (Note: use the time interrupt as the cycle time of DPIDE.)

1. Set the time interrupt to 1000 ms in HWCONFIG.
2. Create a DPIDE instruction in a time interrupt program. Whenever a time interrupt occurs, the PID algorithm is applied. The setting in DPIDE_CYCLE is invalid here.
3. Set the DPIDE_MODE =1 for auto tuning the parameters for the temperature control. After tuning is done, the system is in auto control mode (PID_MODE is set to 0) and fill in the appropriate parameters (Kc_Kp, Ti_Ki, Td_Kd, and Tf).

Main program (cyclic task)

Time interrupt program I601 and the setting parameters