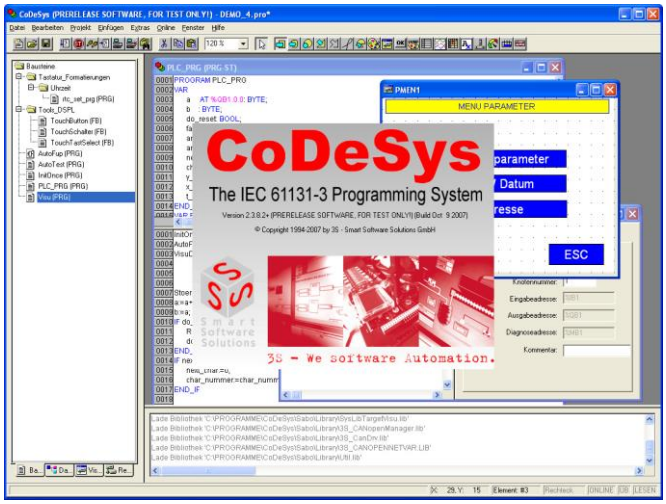
Systemfamilie PLM 700

Modulares CAN I/O-System

System-Handbuch

Regler-Anwendungen



SABO Elektronik GmbH
Lohbachstr. 14
58239 Schwerte
Tel. 02304 / 97102 - 0
Fax 02304 / 97102 - 22
E-Mail info@sabo.de
Internet www.sabo.de

Haftungsausschluss

Der Inhalt des Dokuments wurde auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft; notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Verbesserungsvorschläge sind jederzeit willkommen.

SABO Elektronik GmbH
Lohbachstr. 14
58239 Schwerte
Tel. 02304 / 97102 - 0
Fax 02304 / 97102 - 22

E-Mail info@sabo.de
Internet www.sabo.de

Letzte Aktualisierung: 08. Mrz. 2019

Inhaltsverzeichnis

Seite

1. Wichtige Hinweise

1.1. Bestimmungsgemäße Verwendung

Die Anwendungsgebiete der SABO PLM Baureihe erstrecken sich von der Regelungs- und Steuerungstechnik über die Gebäudeautomation bis zur industriellen Nutzung in der Automatisierung. In allen Anwendungsbereichen ist darauf zu achten, dass im Datenblatt oder Systemhandbuch genannte maximale Spannungen nicht überschritten werden.

Insbesondere ist die Verwendung von SABO-Produkten nicht zulässig für: Überwachung oder Steuerung von Kernreaktionen in Kernkraftwerken, Flugleitsysteme, Flugsicherungssysteme, Steuerung von Massentransportmitteln, medizinische Lebenserhaltungssysteme und Steuerung von Waffensystemen.

1.2. Urheberschutz

Dieses Handbuch sowie alle dazugehörigen Bilder sind Eigentum der SABO Elektronik GmbH und sind urheberrechtlich geschützt. Eine Vervielfältigung, Veränderung oder Veräußerung an Dritte ist nicht gestattet, es sei denn es liegt eine schriftliche Einverständniserklärung der SABO Elektronik GmbH vor. Zuwiderhandlungen ziehen rechtliche Gegenmaßnahmen nach sich.

Die in dieser Dokumentation beschriebenen Produkte und Funktionen können jederzeit den neusten technologischen Entwicklungen angepasst werden. Die gegebenen Informationen können somit nicht als Vertragsgegenstand angesehen werden.

1.3. Personalqualifikation

SABO Produkte dürfen ausschließlich von Fachkräften mit einer Ausbildung in der SPS-Programmierung oder Elektrofachkräften mit einer Unterweisung in den dafür geltenden spezifischen Normen installiert und gewartet werden.

Für Fehler und Schäden, die an SABO Produkten und oder Fremdprodukten entstehen, die auf Missachtung geltender Vorschriften zurückzuführen sind, übernimmt die SABO Elektronik GmbH keine Haftung.

1.4. Sicherheitshinweise

Die Sicherheitshinweise in dieser Dokumentation erheben keinen Anspruch auf Vollständigkeit. Bei Unklarheiten oder der Möglichkeit einer potenziellen Gefährdung von Mensch oder Maschine ist im Zweifelsfall der zuständige Ansprechpartner der SABO Elektronik GmbH hinzuzuziehen. Die in dieser Dokumentation gemachten Hinweise sind Vorschläge und müssen vor dem Übertragen auf die jeweilige Anwendung auf Machbarkeit und Funktionsfähigkeit hin überprüft werden.

Im Allgemeinen dienen die Vorschriften nach VDE beim Umgang mit elektrischen Anlagen als Richtlinie.

2. PID-Regler

2.1. Allgemeines

Häufig besteht die Aufgabe, einen bestimmten Anlagenparameter (z.B. eine Temperatur) auf einen vorgegebenen Wert zu regeln. Die Anlage verfügt dabei über eine sogenannte *Regelstrecke* (z.B. eine Heizung), die entsprechend anzusteuern ist (Abb. 2-1).

Abb. 2-1: Regelstrecke (ohne Regler)

Zur automatischen Regelung solcher Regelstrecken werden u.a. PID-Regler eingesetzt.

Durch einen PID-Regler wird die Regelstrecke zu einem automatischen Regelkreis erweitert (Abb. 2-2). Dazu ist ein geeigneter Sensor (z.B. Temperaturfühler) erforderlich, der dem PID-Regler eine Rückmeldung über den momentanen Ausgangswert der Regelstrecke gibt (Istwert). Der PID-Regler führt die Stellgröße so nach, dass sich der als Sollwert vorgegebene Ausgangswert einstellt.

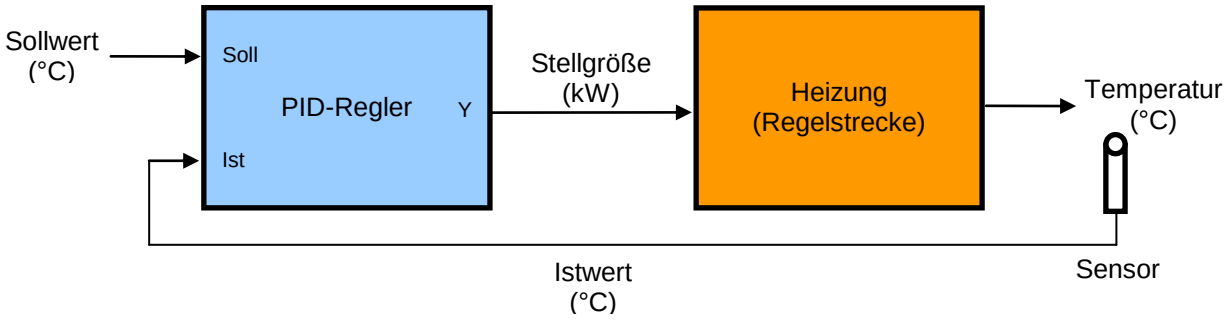


Abb. 2-2: Regelstrecke mit Regler: Regelkreis

Auf die theoretische Beschreibung von Regelkreisen und Reglern kann hier nicht eingegangen werden. Stattdessen wird nur die praktische Realisierung eines Regelkreises mit PLM-Steuerungen dargestellt.

Der PID-Regler benötigt drei Reglerparameter, die sein Regelverhalten bestimmen:

- Kp (Verstärkung, P-Anteil),
- Tn (Nachstellzeit, I-Anteil) und
- Tv (Vorhaltzeit, D-Anteil).

Diese drei Parameter müssen vom Anwender vorgegeben werden.

Die Festlegung dieser drei Parameter erfolgt in der Praxis häufig empirisch, d.h. nach Gefühl und Erfahrungswerten. Außerdem existieren verschiedene *Einstellregeln*, nach denen die drei Parameter Kp, Tn und Tv in Abhängigkeit von der Regelstrecke festgelegt werden können. Die bekanntesten Einstellregeln sind die von Ziegler/Nichols (1942) und deren Weiterentwicklung von Chien/Hrones/Reswick (1952).

Die Bibliothek Plm_Regler.lib stellt einerseits einen standardmäßigen PID-Regler zur Verfügung und enthält andererseits einen Baustein zur halbautomatischen Ermittlung der drei PID-Parameter nach dem Verfahren von Chien/Hrones/Reswick.

2.2. Spezifikation

- PID-Regler
- Stellgrößenbegrenzung
- interne Anti-Windup-Kontrolle (I-Anteil)
- Handbetrieb möglich
- Baustein zur halbautomatischen Parameterermittlung
- benötigt Laufzeitsystem ab Version v21201011

2.3. Benötigte Bibliotheken

Plm_Regler.lib
standard.lib

Die angegebenen Bibliotheken müssen vom Projekt geladen werden. Dazu das Menü *Fenster* → *Bibliotheksverwaltung* öffnen, dort im linken oberen Teilfenster mit der rechten Maustaste klicken und *Weitere Bibliothek...* auswählen.

2.3.1. Plm_PID_Ext3 (Plm_Regler.lib)

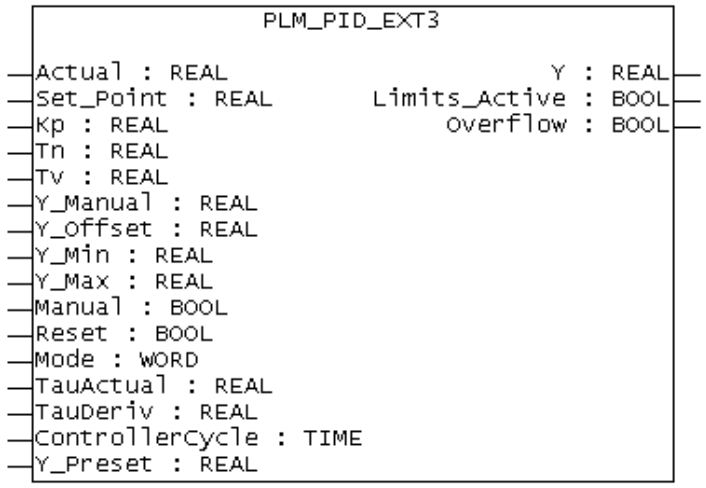


Abb. 2-3: Funktionsblock `Plm_PID_Ext3` (Plm_Regler.lib)

Input-Parameter:		
`Actual`	`REAL`	Istwert (z.B. Rückmeldewert vom Temperatursensor)
`Set_Point`	`REAL`	Sollwert (z.B. Temperaturvorgabe)
`Kp`	`REAL`	P-Reglerparameter
`Tn`	`REAL`	I-Reglerparameter (Einheit: Sekunden), 0 = aus (kein I-Anteil)
`Tv`	`REAL`	D-Reglerparameter (Einheit: Sekunden), 0 = aus (kein D-Anteil)
`Y_Manual`	`REAL`	Wert der Stellgröße `Y` für Handbetrieb (d.h. `Manual` = TRUE)
`Y_Offset`	`REAL`	zusätzl. Offset der Stellgröße `Y`, meist 0
`Y_Min`	`REAL`	untere Begrenzung der Stellgröße `Y` (auch im Handbetrieb), falls `Y_Min` >= `Y_Max`: keine Begrenzung wirksam
`Y_Max`	`REAL`	obere Begrenzung der Stellgröße `Y` (auch im Handbetrieb), falls `Y_Min` >= `Y_Max`: keine

		Begrenzung wirksam
Manual	BOOL	TRUE = Handbetrieb, die Stellgröße Y entspricht dann dem Eingang Y_Manual, auch wenn Reset = TRUE
Reset	BOOL	TRUE = Baustein wird neu initialisiert, Stellgröße Y = Y_Offset
Mode	WORD	Steuerwort für die Betriebsart des Reglers: 0 = Heizbetrieb (pos. Regelabweichung → sinkende Stellgröße) 1 = Kühlbetrieb (pos. Regelabweichung → steigende Stellgröße) + 4 = Regler startet nach Reset mit Y=Y_Preset
TauActual	REAL	Zeitkonstante τ für interne Tiefpassbedämpfung des Eingangs Actual, 0 = kein Tiefpass
TauDeriv	REAL	Zeitkonstante τ für zusätzliche Tiefpassbedämpfung des D-Anteils, 0 = kein Tiefpass. TauDeriv sollte deutlich kleiner sein als Tv
ControllerCycle	TIME	Reglerzykluszeit, 0 = Ausführen bei jedem Aufruf, sonst interne Ausführung nur alle x Sekunden (Regler muss trotzdem zyklisch aufgerufen werden)
Y_Preset	REAL	Y-Wert nach Reset wenn Mode = +4 und Tn <> 0
Output-Parameter:		
Y	REAL	Stellgröße, Ausgang des Reglers zur Regelstrecke (z.B. Heizleistung)
Limits_Active	BOOL	TRUE = Stellgröße Y wird im Moment durch Y_Min oder Y_Max begrenzt
Overflow	BOOL	TRUE = interne Störung, z.B. Überlauf I-Anteil oder ungültige Reglerparameter Kp, Tn, Tv oder falsches LZS

Der Baustein Plm_PID_Ext3() muss in regelmäßigen Abständen aufgerufen werden.

Die Berechnung der Stellgröße Y erfolgt intern gemäß folgender Formel (vereinfacht):

$$abw = Set\_Point - Actual$$

$$Y = Y\_Offset + (Kp \times abw) + ((Kp / Tn) \times \text{Integral}(abw)) + (Kp \times Tv \times \text{Differential}(abw))$$

Durch Nullsetzen der Parameter Tn oder Tv wird entsprechend der I- oder D-Anteil abgeschaltet (dies ist für Tn nicht aus der dargestellten Formel ersichtlich). Falls Kp Null ist, ist der Regler unwirksam und der Ausgang Overflow ist TRUE.

Der Baustein ermittelt intern die Zeit Δt, die seit dem letzten Aufruf vergangen ist (bei zyklischem Aufruf: Δt = 20 ms) und normiert die Parameter Tn und Tv entsprechend. Tn und Tv werden daher immer in der Einheit *Sekunde* angegeben.

Die Reglerparameter Kp, Tn und Tv können z.B. bei einer Temperaturregelung mit Heizung anschaulich so gedeutet werden:

Eingang Actual: Istwert (Sensor) in °C

Eingang Set_Point: Sollwert (Vorgabe) in °C

Ausgang `Y`:	Vom Regler berechnete Heizleistung in Prozent (z.B. 0...100)
Parameter `Kp`:	Proportionale Reaktion des Reglers auf Temperaturabweichungen vom Sollwert, z.B. `Kp` = 10 %/°C → pro Grad Temperaturabweichung wird die Heizleistung um 10 % erhöht bzw. verringert
Parameter `Tn`:	Nachstellzeit in Sekunden, diese wird intern auf Kp bezogen und ist ein Maß für die Reaktionsgeschwindigkeit des Reglers, z.B. `Tn` = 20 s → pro Grad Temperaturabweichung über 20 Sekunden wird zusätzliche Heizleistung gemäß Kp nachgeregelt, z.B. wird die Heizleistung zusätzlich um 10 % alle 20 Sekunden erhöht. Wird `Tn` größer gewählt, wird der I-Anteil geringer. `Tn` = 0 schaltet den I-Anteil ab.
Parameter `Tv`:	Vorhaltzeit in Sekunden, z.B. `Tv` = 5 → bei Änderung der Temperaturabweichung um 1 °C pro Sekunde wird zusätzlich die Heizleistung sofort mit 5 × `Kp` nachgeregelt. Hier wird also die Heizleistung zusätzlich um 50 % erhöht, solange die Isttemperatur mit 1 °C pro Sekunde fällt. Wird `Tv` größer gewählt, wird auch der D-Anteil größer. `Tv` = 0 schaltet den D-Anteil ab.

Bei einer Heizregelung muss `Mode` = 0 sein, bei einer Kühlregelung `Mode` = 1.

Für viele Anwendungen ist ein PI-Regler gut geeignet, d.h. `Kp` und `Tn` müssen parametriert werden und `Tv` = 0.

Der Bereich, in dem sich die Ausgangsstellgröße `Y` bewegt, wird durch die Eingänge `Y_Min` und `Y_Max` begrenzt. Wenn die Begrenzung anspricht, wird `Limits_Active` = TRUE. Die Begrenzung kann abgeschaltet werden, indem `Y_Max` auf den gleichen oder einen kleineren Wert wie `Y_Min` gesetzt wird, z.B. `Y_Max` = `Y_Min` = 0. Bei einem Heizregler entspricht z.B. die Stellgröße der Heizleistung im Bereich 0...100 %, dann ist `Y_Min` = 0 und `Y_Max` = 100.

Ein Werterauschen des Sensorwerts (`Actual`) ist für die interne Berechnung ungünstig. Verrauschte Sensorwerte können durch eine Tiefpassfilterung geglättet werden, dies kann durch eine Angabe der Tiefpass-Zeitkonstanten `TauActual` (in s) erfolgen. Bei `TauActual` = 0 ist der Tiefpass ausgeschaltet. Alternativ kann der Bibliotheksbaustein `Plm_PID_Average_Real()` verwendet werden. Bei Verwendung von `Tv` kann ein zusätzlicher Tiefpass nur für den D-Anteil mit `TauDeriv` konfiguriert werden.

Bei langsamen Regelstrecken reicht es aus, den Regler nicht in jedem IEC-Zyklus, sondern z.B. nur einmal pro Sekunde auszuführen. Dadurch verringert sich einerseits die Rechenlast der CPU, andererseits steigt die Regelgenauigkeit, da die interne Zeitmessung genauer wird und Sensorwerte (`Actual`) besser gemittelt werden können.

Beim Einsatz mehrerer PID-Regler mit langsamen Regelstrecken empfiehlt es sich, die Regler nicht alle im selben IEC-Zyklus aufzurufen, um die Rechenlast der CPU zu minimieren. Stattdessen sollten die Regelbausteine in aufeinanderfolgenden IEC-Zyklen aufgerufen werden.

2.3.2. Plm_PID_Average_Real (Plm_Regler.lib)

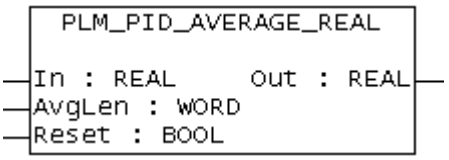


Abb. 2-4: Funktionsblock `Plm_PID_Average_Real` (Plm_Regler.lib)

Input-Parameter:		
`In`	`REAL`	Neuer Eingangswert
`AvgLen`	`WORD`	Anzahl der letzten Eingangswerte, über die gemittelt werden soll (1...100)
`Reset`	`BOOL`	TRUE = setze Mittelwert auf Eingangswert `In`
Output-Parameter:		
`Out`	`REAL`	Arithmetisches Mittel über die letzten `AvgLen` Eingangswerte

Der Baustein `Plm_PID_Average_Real()` bildet das arithmetische Mittel über die letzten `AvgLen` Eingangswerte.

Bei jedem Aufruf von `Plm_PID_Average_Real()` wird der neue Eingangswert `In` übernommen und der älteste Eingangswert verworfen.

`AvgLen` kann im Bereich 1...100 liegen, d.h. es können bis zu 100 Werte für die Mittelwertbildung berücksichtigt werden. Eine Änderung von `AvgLen` löst intern einen Reset des Bausteins für einen Zyklus aus. Falls `AvgLen` Werte größer als 100 annimmt, hat `Out` den festen Wert 0.

Ein interner Reset des Bausteins erfolgt außerdem beim ersten Aufruf nach Start des IEC-Programms.

2.3.3. Plm_PID_AutoParam (Plm_Regler.lib)

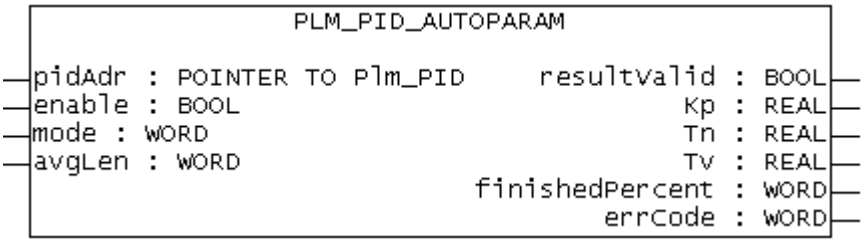


Abb. 2-5: Funktionsblock `Plm_PID_AutoParam` (Plm_Regler.lib)

Input-Parameter:		
`pidAdr`	`POINTER TO Plm_PID`	Adresse `ADR()` des PID-Reglerbausteins vom Typ `Plm_PID()`, für den die Reglerparameter ermittelt werden sollen
`enable`	`BOOL`	Start der halbautomatischen Parameterermittlung (s.u.)
`mode`	`WORD`	Art der Regelstrecke und Art der Parameterermittlung (s.u.), zusätzlich: + 16#8000 = Berechnung nicht einfrieren nachdem `resultValid` = TRUE
`avgLen`	`WORD`	Für interne Mittelwertbildung: Anzahl der Reglereingangswerte `Actual`, über die gemittelt werden soll (0...100)

Output-Parameter:		
resultValid	BOOL	TRUE: Parameterermittlung abgeschlossen
Kp	REAL	Vorgeschlagener Wert für Reglerparameter Kp
Tn	REAL	Vorgeschlagener Wert für Reglerparameter Tn
Tv	REAL	Vorgeschlagener Wert für Reglerparameter Tv
finishedPercent	WORD	Fortschrittsindikator 0...100, Angabe in Prozent
errCode	WORD	Fehlercode (Bitmuster): 0 = kein Fehler + 16#0001 = PID-Regler ist nicht im manuellen Betrieb + 16#0002 = PID-Regler Actual-Eingang zeigt keine Reaktion auf manuelle Stellgrößenänderung + 16#0004 = PID-Regler Stellgrößenausgang Y zeigt keine Reaktion auf manuelle Stellgrößenänderung + 16#0008 = Folgefehler aus 16#0002 oder 16#0004 + 16#0010 = Eingang mode ungültig + 16#0020 = avgLen ist größer als 100

Der Baustein `Plm_PID_AutoParam()` ermittelt für einen Baustein vom Typ `Plm_PID()` halbautomatisch einen Satz von Reglerparametern `Kp`, `Tn` und `Tv`.

Die ermittelten Parameter sind als Vorschlag zu verstehen und bedürfen anschließend der manuellen Überprüfung und Optimierung. Für die Ergebnisse kann keine Gewähr übernommen werden. Die Firma SABO Elektronik GmbH übernimmt keinerlei Verantwortung für die korrekte Funktion eines Reglers bei Einsatz dieser Parameter oder für Schäden, die während der halbautomatischen Parameterermittlung oder durch Einsatz der ermittelten Parameter entstehen.

Die Parameterermittlung erfolgt nach den Regeln von Chien/Hrones/Reswick. Das Verfahren basiert auf der Untersuchung einer *Sprungantwort* des *ungeregelten* Systems (vgl. Abb. 2-1). Als Sprungantwort wird das Verhalten der Ausgangsgröße der Regelstrecke (z.B. Temperatur) als Reaktion auf eine sprunghafte Änderung der Stellgröße `Y` (z.B. Heizleistung) verstanden.

Zur Ermittlung einer Sprungantwort ist der zugehörige PID-Regler im Handbetrieb zu betreiben (`Manual` = TRUE). Am Eingang `Y_Manual` kann in diesem Zustand die jeweils gewünschte Stellgröße `Y` für die Regelstrecke eingestellt werden.

Die beiden manuell einzustellenden Stellgrößenwerte sind so zu wählen, dass die Sprungantwort relativ groß ausfällt. Außerdem muss sich für beide Stellgrößenwerte ein stabiler Ausgangszustand einstellen.

Beim manuellen Ermitteln der Sprungantwort darf zu keiner Zeit die Anlagensicherheit gefährdet sein; z.B. kann ein Heizsystem normalerweise nicht einfach von Null auf volle Heizleistung gestellt werden, da es sich möglicherweise überhitzen würde. Zwei passende Werte für die Stellgröße sind vom Anwender in eigener Verantwortung festzulegen. Die Regelstrecke ist während der Durchführung der halbautomatischen Parameterermittlung ständig persönlich zu überwachen und bei Eintreten eines gefährlichen Anlagenzustands sofort abzuschalten.

Abb. 2-6 zeigt als Beispiel die Sprungantwort eines Heizsystems. Die Stellgröße (Heizleistung in Prozent) wurde zunächst auf 0 gestellt (Heizung aus) und das Erreichen einer stabilen Ausgangstemperatur abgewartet (hier: 20 °C Umgebungstemperatur). Im Moment t = 0 s wurde die Heizleistung auf 10 % erhöht. Die Ausgangstemperatur stabilisierte sich nach einiger Zeit bei ca. 55 °C.

Aus dem zeitlichen Verlauf von Stellgröße und Ausgangstemperatur kann nach Chien/Hrones/Reswick ein Satz von Reglerpametern `Kp`, `Tn` und `Tv` für den PID-Regler abgeschätzt werden.

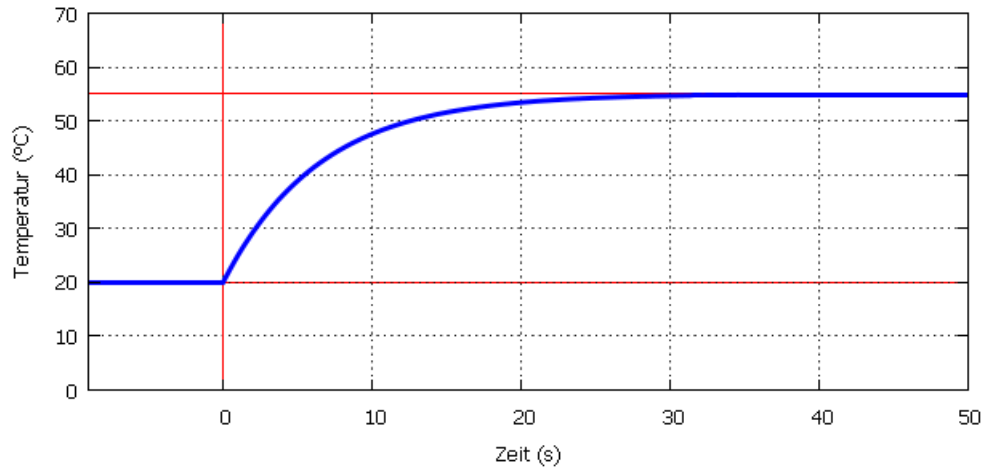


Abb. 2-6: Sprungantwort eines Heizsystems, Heizleistung bei t = 0 s von 0 % auf 10 % erhöht

Falls die Regelstrecke nicht im Handbetrieb betrieben werden kann, ist das Verfahren nicht anwendbar. Außerdem muss es sich um eine *Regelstrecke mit Ausgleich* handeln, die sich bei den gewählten Werten der Stellgröße jeweils auf einen stabilen Ausgangswert (z.B. Temperatur) einstellt.

Der Eingang `mode` legt die Art der Parameterberechnung fest und kann folgende Werte annehmen:

mode	Überschwingen bei Regelung	Reglerverhalten optimal für	Art des Reglers
0	nein	Führung	PID
1	nein	Führung	PI
2	nein	Führung	P
3	nein	Störung	PID
4	nein	Störung	PI
5	nein	Störung	P
10	ca. 20 %	Führung	PID
11	ca. 20 %	Führung	PI
12	ca. 20 %	Führung	P
13	ca. 20 %	Störung	PID
14	ca. 20 %	Störung	PI
15	ca. 20 %	Störung	P

Bei einer Regelung ohne Überschwingen wird der Sollwert langsamer, aber dafür ohne Überschreitung erreicht. Bei Regelung mit Überschwingen wird der Sollwert schneller erreicht, dafür wird ein Überschwingen von ca. 20 % der Sollwertänderung in Kauf genommen (siehe Abb. 2-7). Diese Angaben sind als Richtwerte zu betrachten.

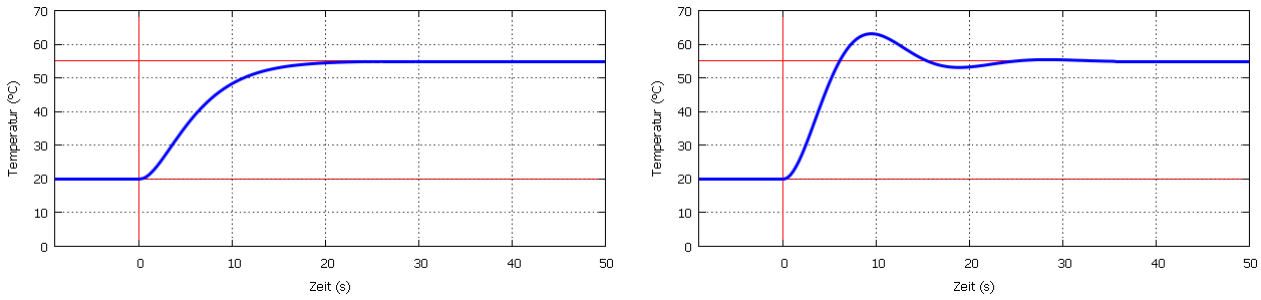


Abb. 2-7: Regelung ohne Überschwingen (links) und mit ca. 20 % Überschwingen (rechts)

Optimales Führungsverhalten bedeutet, dass der Regler Sollwertänderungen optimal folgt. Optimales Störungsverhalten bedeutet, dass der Regler Störungen der Regelstrecke (äußere Beeinflussung des Istwerts) optimal ausgleicht.

Der Wert von `mode` kann auch noch während und nach Abschluss einer halbautomatischen Parameterermittlung geändert werden.

2.4. Vorgehensweise bei der halbautomatischen Parameterermittlung

1. Das IEC-Programm muss bereits einen vollständigen Regelkreis mit einem PID-Regler vom Typ `Plm_PID()` enthalten. Der Regelkreis enthält normalerweise einen Sensoreingang mit dem Istwert, der gemittelt und an `Actual` angelegt wird, außerdem einen Steuerausgang für die Regelstrecke, der über die Stellgröße `Y` angesteuert wird (vgl. Abb. 2-2). Der Baustein `Plm_PID_AutoParam()` wird nach dem zugehörigen Reglerbaustein `Plm_PID()` aufgerufen, dabei wird die Adresse des Reglerbausteins in `pidAdr` übergeben.
2. Der PID-Regler wird in den Handbetrieb versetzt (`Manual` = TRUE) und der erste gewünschte Wert der Stellgröße `Y` an den Eingang `Y_Manual` angelegt. Der Wert muss zwischen den Grenzwerten `Y_Min` und `Y_Max` liegen. Anschließend wird gewartet, bis die Regelstrecke eingeschwungen ist und einen konstanten Istwert liefert (Sensorwert ändert sich nicht mehr). Die Regelstrecke ist dabei ständig auf gefährliche Anlagenzustände hin zu überwachen.
3. Die halbautomatische Parameterermittlung wird jetzt gestartet, indem der Eingang `enable` auf TRUE gesetzt wird. Danach ca. 10 s warten.
4. Der zweite gewünschte Wert der Stellgröße `Y` wird an den Eingang `Y_Manual` angelegt. Der Wert muss zwischen den Grenzwerten `Y_Min` und `Y_Max` liegen. Die Regelstrecke ist dabei ständig auf gefährliche Anlagenzustände hin zu überwachen.
5. Der Ausgang `finishedPercent` gibt einen Anhaltspunkt zum Fortschritt der Parameterermittlung. Bei 100 % springt der Ausgang `resultValid` auf TRUE und die bis dahin ermittelten Werte von `Kp`, `Tn` und `Tv` werden eingefroren. Nachträgliche Änderungen am Eingang `mode` sind möglich und bewirken eine sofortige Neuberechnung der Werte auf Basis der vorher analysierten Sprungantwort.
6. Nach Abschluss der Parameterermittlung wird `enable` auf FALSE gesetzt. Die Werte für `Kp`, `Tn` und `Tv` sind auf Plausibilität zu untersuchen und können anschließend vom Anwender an die entsprechenden Eingänge des PID-Reglers angelegt werden.
7. Der auf diese Weise neu parametrierte PID-Regler ist zunächst gründlich zu testen (Sollwertänderungen, Störgrößeneinwirkungen). Falls der Regelkreis schwingt, ist testweise der `Kp`-Wert zu reduzieren. Falls der Sollwert mit einer konstanten Abweichung nicht erreicht wird, ist testweise `Kp` zu erhöhen. Andere praktische Hilfen sind der Literatur zu entnehmen.

2.5. Programmbeispiel

Das folgende Programmbeispiel realisiert eine Anwendung mit einem PID-Regler. Das Beispiel ist teilweise in ST, teilweise in FUP implementiert.

In der Anwendung wird eine Heizplatte über ein potenzialgetrenntes 230 V-Halbleiterelais angesteuert. Die Temperatur der Heizplatte soll geregelt werden.

Die Temperatur der Heizplatte (Istwert) wird mit einem Pt 100-Fühler gemessen, der von einer Erweiterungsbaugruppe EWB.730.10 ausgewertet wird. Das Halbleiterrelais ist an einen 24 V-Digitalausgang der EWB.730.10 angeschlossen. Die Ansteuerung des Halbleiterrelais' erfolgt mit einem langsamen PWM-Signal (Pulspaketmodulation), welches vom IEC-Programm aus dem Ausgangswert `Y` (Stellgröße) des PID-Reglers erzeugt wird.

Die eingelesenen Sensorwerte von der EWB.730.10 enthalten die Temperatur in °C × 10. Die Sensorwerte werden in °C umgerechnet, über 20 Messwerte gemittelt und liefern den Istwert `Actual` für den Regler.

Der PID-Regler befindet sich im Programmblock `ReglerWork()`, der einmal pro Sekunde aufgerufen wird.

Der Ausgang `Y` des PID-Reglers (Stellgröße) liefert Werte im Bereich 0...100 (`Y_Min`...`Y_Max`). Diese Werte werden durch den Baustein `ControlPercentPulse()` aus der Bibliothek `PlmUtil.lib`, der ebenfalls im Sekundentakt aufgerufen wird, in Pulse zur Ansteuerung des Halbleiterrelais' umgewandelt.

PLC_PRG (PRG-ST)

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     pwmOut: ControlPercentPulse;
0004     sensorAvg: Plm_PID_Average_Real;
0005     anaIn AT %IB1.0.1: INT;
0006     digOut AT %QX1.0.0.0: BOOL;
0007     timer: TON;
0008     reglerPulse: BOOL;
0009     istTemp: REAL;
0010     sollTemp: REAL;
0011     stellgr: REAL;
0012 END_VAR
0013
```

```
0001
0002 (* Sensorwert einlesen und mitteln *)
0003 sensorAvg(
0004     In := INT_TO_REAL(anaIn) / 10.0,  (* EWB Analogeing. Pt100 *)
0005     AvgLen := 20,
0006     Out => istTemp
0007 );
0008
0009 (* Sekundenpuls erzeugen *)
0010 timer();
0011 IF timer.Q OR timer.M = FALSE THEN
0012     timer( IN:=FALSE );
0013     timer( IN:=TRUE, PT:=T#1s );
0014     reglerPulse := TRUE;
0015 END_IF
0016
0017 (* Aufruf einmal pro Sekunde *)
0018 IF reglerPulse THEN
0019     reglerPulse := FALSE;
0020     (* PID-Regler und Auto-Param *)
0021     ReglerWork(
0022         istTemp := istTemp,
0023         sollTemp := sollTemp,
0024         output => stellgr
0025     );
0026     (* Stellgröße in PWM umwandeln *)
0027     pwmOut(
0028         Enable := TRUE,
0029         Nominal := REAL_TO_WORD(stellgr),
0030         MaxSignal := 100,  (* entspricht Y_Max am PID_Regler *)
0031         Q => digOut  (* EWB Digitalausgang *)
0032     );
0033 END_IF
0034
```

Abb. 2-8: Beispiel für PID-Regler-Anwendung (Teil 1)

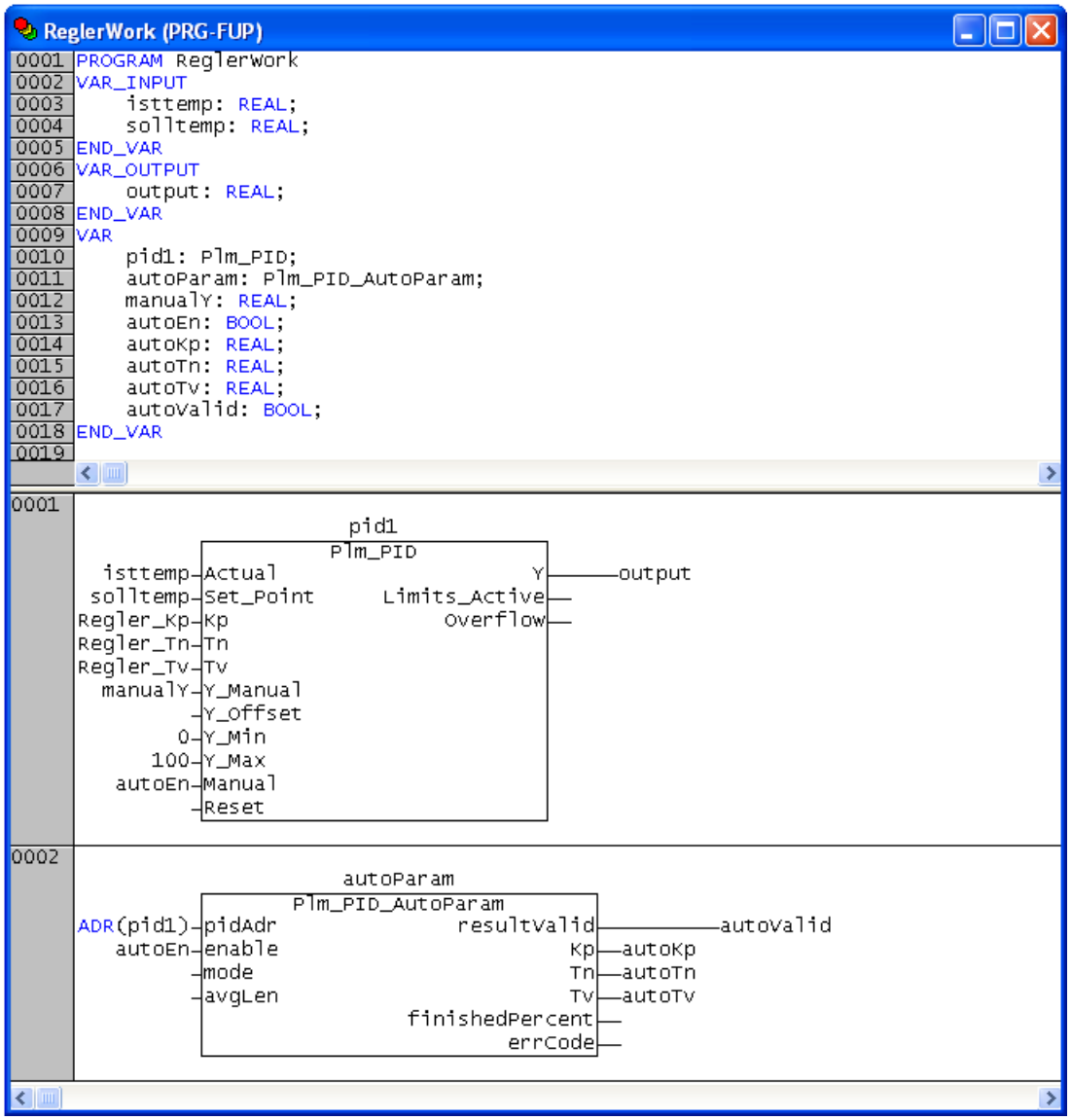


Abb. 2-9: Beispiel für PID-Regler-Anwendung (Teil 2)