Versuch 4: Programmieren von Regelalgorithmen

Teilnehmer: _____

_____

_____

Versuchsziel

In diesem Versuch steht die Implementierung von Regelalgorithmen auf Mikrocontrollern im Mittelpunkt. Im Gegensatz zu Hochleistungsmikroprozessoren besitzen Mikrocontroller im allgemeinen keinen Gleitkommacoprozessor, so daß die Regelalgorithmen im Hinblick auf die Rechenzeiten mit Ganzzahlarithmetik realisiert werden müssen. Durch die Ganzzahl-(Integer)-arithmetik können Rundungsfehler und Überläufe auftreten, die zu Genauigkeitseinbußen oder sogar zum völligen Versagen der Regelung führen können. An Beispielen wird gezeigt, wie diese Probleme durch geeignete Normierung der Signale und Parameter vermieden werden können.

Alle Beispiele werden in C programmiert und in einer Laufzeitumgebung unter Windows erprobt. Die Laufzeitumgebung erlaubt (näherungsweise) einen Echtzeitbetrieb, auch wenn die gewählten Abtastzeiten (>50ms) mit Rücksicht auf die Kompatibität auch mit älteren Windows-Versionen nicht allzu schnell sind. Mit neueren Windowsversionen (insbesondere Windows NT) lassen sich Abtastzeiten bis zu wenigen ms erreichen, allerdings weisen diese Abtastzeiten einen erheblichen Jitter auf, da Windows natürlich kein wirkliches Echtzeitbetriebssystem ist. Wenn Ihr PC mit einer A/D-D/A-Wandlerkarte ausgestattet wäre, könnte man mit diesem Programmsystem aber immerhin langsamere reale Regelstrecken wie Temperatur-, Druck-, Helligkeits- oder auch (größere) Antriebsysteme regeln.

Zur Übersetzung der Programme wird ein Borland C-Compiler (ab Version 3.1) benötigt. Die Steuerung des Regelungsprogramms und die graphische Anzeige der Ergebnisse erfolgt mit MATLAB (ab Windows-Version 4).

Versuchsdurchführung

Der Versuchsumdruck ist als Tutorial geschrieben. **Der Versuch soll selbständig durchgeführt werden.** Je Gruppe muß ein Versuchsprotokoll mit den Ergebnissen zu den Aufgaben (Berechnungen, Ausdruck der Zeitverläufe der Simulationen, Listing der Programmdateien, Diskette mit lauffähigen Lösungen) abgegeben werden. Bitte benennen Sie die lauffähigen Versionen auf der Disketten nach dem Schema RTCx_x.exe, wobei x_x für die Nummer der Aufgabe steht; also z.B. RTC3_1.exe für die Lösung zur Aufgabe 3.1.

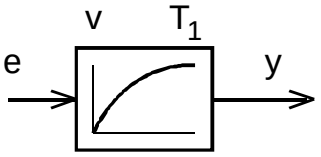
1. Von der Übertragungsfunktion zum Algorithmus

Während in MATLAB mit Übertragungsfunktionen, dh. im Frequenzbereich, gearbeitet werden kann, muß auf Mikrocontrollern direkt der zeitdiskrete Algorithmus programmiert werden, der die zugehörige Differentialgleichung in eine zeitdiskrete Form bringt. Da zudem in einem Echtzeitsystem der Verlauf des Eingangssignals nicht im voraus bekannt ist, muß diese Form rekursiv sein, d.h. aus dem aktuellen und gegebenenfalls früheren Eingangs- und Ausgangssignalen muss jeweils das Ausgangssignal für den nächsten Abtastschritt berechnet werden.

1.1 Grundlegende Vorgehensweise am Beispiel eines PT1-Gliedes

Die hierfür notwendigen Schritte sollen zunächst an einem PT1-Glieds gezeigt werden:

$$G(s) = \frac{Y(s)}{E(s)} = \frac{v}{1 + s\,T_1} \qquad (1.1)$$



Schritt 1: Umformung der Übertragungsfunktion

$$Y(s) = G(s) \cdot E(s) = \frac{v}{1 + s\,T_1} E(s) \qquad (1.2)$$

Mit dem Nenner durchmultipliziert

$$Y(s) \cdot [\,1 + s\,T_1] = v \cdot E(s) \qquad (1.3)$$

Schritt 2: Rücktransformation in den Zeitbereich

$$y(t) + T_1 \cdot \frac{dy(t)}{dt} = v \cdot e(t) \qquad (1.4)$$

Nach dem Differentialquotienten aufgelöst:

$$T_1 \cdot \frac{dy(t)}{dt} = v \cdot e(t) - y(t) \qquad (1.5)$$

Da in einem Echtzeitsystem der zeitliche Verlauf des Eingangsignals e(t) im voraus nicht bekannt ist, kann dieses Differentialgleichung nicht direkt gelöst werden.

Schritt 3: "Abtasten"

Darüberhinaus besitzt der Mikrocontroller eine endliche Rechenzeit. y(t) kann daher nur zu den diskreten Zeitpunkten $t = k \cdot T$ mit $k = 0,1,2,\ldots$ berechnet werden. Dabei muß die "Abtastzeit" T größer (oder gleich) der Rechenzeit gewählt werden.

Ersetzt man $\frac{dy(t)}{dt} \rightarrow \frac{y(k) - y(k-1)}{T}$ (1.6)

$$e(t) \rightarrow e(k) \qquad \text{(sh. auch Hinweis 1 auf Seite 5)} \qquad (1.7)$$

und $$y(t) \rightarrow y(k\text{-}1) \qquad \text{(sh. auch Hinweis 1 auf Seite 5)} \qquad (1.8)$$

so ergibt sich $$T_1 \frac{y(k)-y(k\text{-}1)}{T} = v \cdot e(k) - y(k\text{-}1) \qquad (1.9)$$

Schritt 4: Rekursionsformel

Auflösen nach y(k): $$y(k) = y(k\text{-}1) + \frac{T}{T_1}[v \cdot e(k) - y(k\text{-}1)] \qquad (1.10)$$

andere Darstellung: $$y(k) = [1 - \frac{T}{T_1}]\, y(k\text{-}1) + \frac{T}{T_1} v \cdot e(k) \qquad (1.10a)$$

Mit dieser **Rekursionsformel** kann man also die Ausgangsgröße y(k) zum aktuellen Abtastzeitpunkt t = kT aus der aktuellen Eingangsgröße e(k) und der Ausgangsgröße y(k-1) des vorigen Abtastzeitpunkts (k-1)T berechnen.

1.2 Übertragungsglieder höherer Ordnung am Beispiel eines PT2-Gliedes

Diese Vorgehensweise ist auch auf Übertragungsglieder höherer Ordnung anwendbar, wobei sich allerdings im Schritt 2 eine Differentialgleichung höherer Ordnung (allgemein: DGL n.Ordnung) ergibt, die in einem Zwischenschritt in n Differentialgleichungen 1.Ordnung umgeformt werden muß.

Beispiel PT2-Glied: $$G(s) = \frac{Y(s)}{E(s)} = \frac{v}{1 + s\,2DT_0 + s^2 T_0^2} \qquad (1.11)$$

Schritt 1: Umformung der Übertragungsfunktion

$$Y(s)\,[1 + s\,2DT_0 + s^2 T_0^2] = v \cdot E(s) \qquad (1.12)$$

Schritt 2: Rücktransformation in den Zeitbereich

$$y(t) + 2DT_0 \cdot \frac{dy(t)}{dt} + T_0^2 \frac{d^2 y(t)}{dt^2} = v \cdot e(t) \qquad (1.13)$$

Schritt 2a: Umformung in ein System von n DGL 1. Ordnung (hier n = 2)
Einführung von (n-1) Zwischengrößen

$$x(t) = T_0 \frac{dy(t)}{dt}, \qquad x_2(t) = T_0^2 \frac{d^2 y(t)}{dt^2} \quad \ldots \quad x_{n-1}(t) = T_0^{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}}$$

hier: $$T_0 \frac{dy(t)}{dt} = x(t) \qquad (1.14)$$

in (1.13) eingesetzt und nach dem Differentialquotienten aufgelöst:

$$T_0 \frac{dx(t)}{dt} = v \cdot e(t) - y(t) - 2D\, x(t) \qquad (1.15)$$

Schritt 3: "Abtasten"

in (1.14) $$T_0 \frac{y(k)-y(k-1)}{T} = x(k-1) \qquad (1.16)$$

in (1.15) $$T_0 \frac{x(k)-x(k-1)}{T} = v \cdot e(k) - y(k-1) - 2D\, x(k-1) \qquad (1.17)$$

Schritt 4: Rekursionsformeln

Auflösen nach y(k): $$y(k) = y(k-1) + \frac{T}{T_0} x(k-1) \qquad (1.18)$$

Auflösen nach x(k): $$x(k) = x(k-1) + \frac{T}{T_0} [v \cdot e(k) - y(k-1) - 2D\, x(k-1)] \qquad (1.19)$$

1.3 Zeitdiskrete Übertragungsfunktionen am Beispiel eines I-Reglers

Wenn bereits die **zeitdiskrete Übertragungsfunktion** gegeben ist, entfällt selbstverständlich der Schritt 3.

Beispiel: Zeitdiskreter I-Regler $$G(z) = \frac{Y(z)}{E(z)} = k_p \frac{T}{T_n} \frac{z}{z-1} \qquad (1.20)$$

Schritt 1: Umformung der Übertragungsfunktion

$$Y(z) [1 - z^{-1}] = E(z) \cdot k_p \frac{T}{T_n} \qquad (1.21)$$

Schritt 2: Rücktransformation in den Zeitbereich

$$y(k) - y(k-1) = k_p \frac{T}{T_n} e(k) \qquad (1.22)$$

Schritt3: entfällt

Schritt 4: Rekursionsformel

$$y(k) = y(k-1) + k_p \frac{T}{T_n} e(k) \qquad (1.23)$$

Aufgabe 1.1

Bestimmen Sie die Rekursionsformeln für

a) ein P-Glied $G(s) = G(z) = k_P$

b) für ein zeitdiskretes D-Glied $G(z) = k_P \cdot \frac{T_V}{T} \cdot \frac{z-1}{z}$

und *c) für ein zeitkontin. DT1-Glied* $G(s) = \frac{sT_V}{1+sT_1}$.

2. Vom Algorithmus zum C-Programm

Die Rekursionsformeln können im Prinzip direkt programmiert werden. Für das PT2-Glied nach Gl. (1.18) und (1.19) beispielsweise ergibt sich folgendes Programm:

```
        float       e;                  // Eingangssignal
        float       x=0, y=0;           // Ausgangs- und Zwischensignal mit Anfangswert
        const float T_T0=0.1,           // Parameter des PT2-Glieds T/T0,  2·D,  v
                    _2D=2*0.7,  v=1;
        float dx, dy;                   // Hilfsgrößen   dx=x(k)-x(k-1)   dy=y(k)-y(k-1)

void PT2 (void)
{       e= ... ;                        // Eingangssignal z.B. von A/D-Wandler lesen
        dy=T_T0 * x;                    // Hilfsgrößen in Gl.(1.18) und (1.19) berechnen
        dx=T_T0 * (v * e - y - _2D * x);
        y=y+dy;                         // Gl.(1.18)
        x=x+dx;                         // Gl.(1.19)
        ...                             // Ausg.signal z.B. an D/A-Wandler ausgeben
}
```

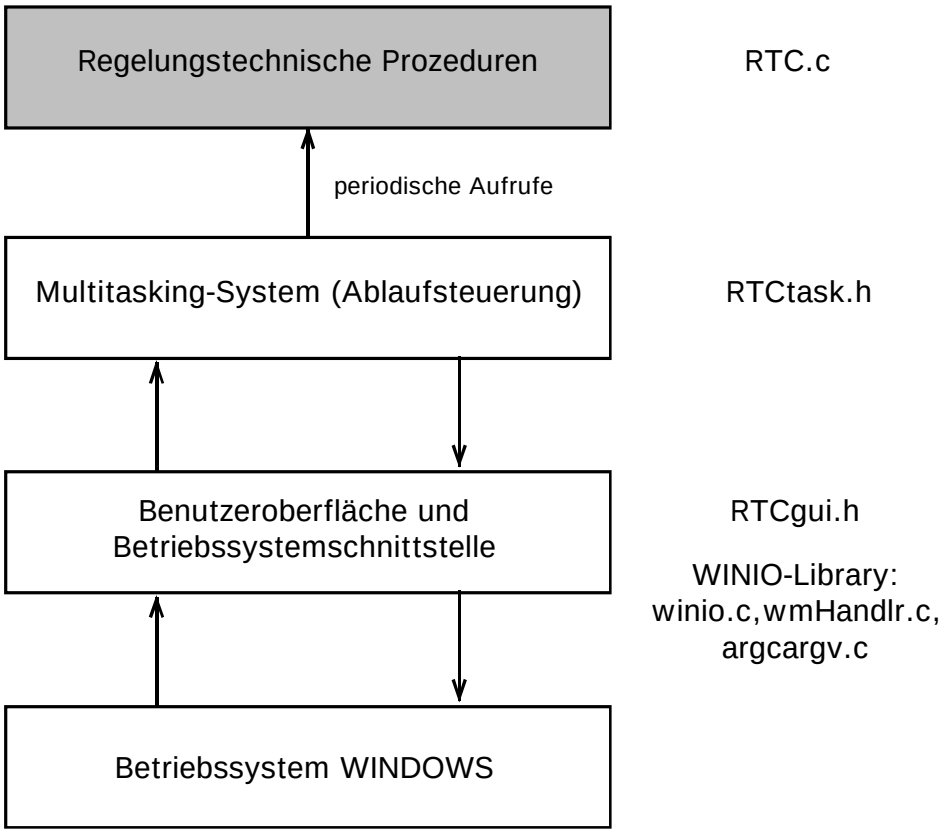
Damit dieses Programm einsetzbar wird, wird allerdings zusätzlich ein einfaches 'Betriebssystem' benötigt, das die Prozedur PT2( ) periodisch aufruft. Wenn auf demselben Rechner mehrere solcher Prozeduren, gegebenenfalls mit unterschiedlichen Abtastzeiten, bearbeitet werden sollen, ist eine Ablaufsteuerung (Multitaskingsystem) notwendig.

Hinweis 1 zu Seite 3:
An dieser Stelle taucht immer wieder die Frage auf, weshalb bei e(t) $t = k \cdot T$, bei y(t) dagegen $t = (k-1) \cdot T$ verwendet wird? Beides ist weder völlig richtig noch völlig falsch, da es sich schließlich um eine Näherung der kontinuierlichen Zeit t durch die diskreten Zeitpunkte $t = T, 2T, 3T, \ldots$ handelt. Die Antwort muß daher pragmatisch gegeben werden:
Da die Ausgangsgröße y(t) zum Zeitpunkt $t = k \cdot T$ berechnet werden soll, ist y(k) noch nicht bekannt, daher wird bei y(t) $t = (k-1) \cdot T$ eingesetzt. Die Eingangsgröße e(t) dagegen ist zum Zeitpunkt $t = k \cdot T$ bekannt, da sie (wenn man die dafür eigentlich notwendige Meß- und Rechenzeit vernachlässigt) zu diesem Zeitpunkt gemessen werden kann. Berücksichtigt man die Meß- und Rechenzeit, so muß man bei e(t) entweder auch $t = (k-1) \cdot T$ setzen oder man setzt $t = k \cdot T$ und vernachlässigt, daß das Berechnungsergebnis erst um die Meß- und Rechenzeit später zur Verfügung steht (s.h. dazu Vorlesung, Abschnitt F).

3. Die Programm- und Testumgebung für den Laborversuch

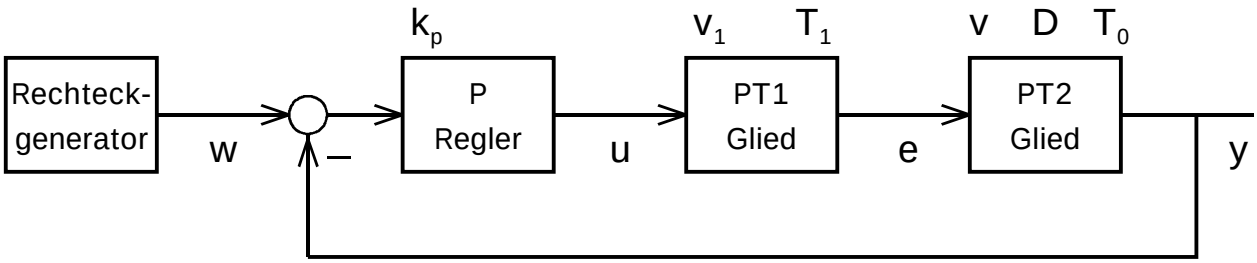
3.1 Überblick

Für den vorliegenden Laborversuch wird eine in vier Schichten strukturierte Ablaufumgebung verwendet:



- Wegen der einfachen Verfügbarkeit in den Rechnerpools und Labors wird als unterlagertes **Betriebssystem** WINDOWS (3.1, 9x, NT) eingesetzt, über das alle Ein- und Ausgaben abgewickelt werden.
- Als **Schnittstelle zum Betriebssystem und** als **Benutzeroberfläche** unseres Programms wird die WINIO-Library von A.Schulmann verwendet [sh. 'A.Schulman, D.Maxey, M.Pietrek: Undocumented Windows, Verlag Addison-Wesley' bzw. 'A.Schulman: Unauthorized Windows, IWT-Verlag'], die es erlaubt, auf Anwenderebene Windows-Programme wie 'normale' DOS-Programme zu schreiben. Die Benutzeroberfläche befindet sich in der Datei RTCgui.h, die WINIO-Library besteht aus den Programmen winio.c, wmHandlr.c, argcargv.c und den zugehörigen Include-Dateien winio.h und wmHandlr.h.
- Das einfache **Multitasking-System** in der Datei RTCtask.h, von dem aus die verschiedenen regelungstechnischen Prozeduren periodisch aufgerufen werden, basiert auf dem periodischen Zeitgeberinterrupt, den Windows bereitstellt.
- Auf der obersten Ebene in der Datei RTC.c finden sich die regelungstechnischen Prozeduren.

Das Programm RTC bildet den folgenden Regelkreis nach:

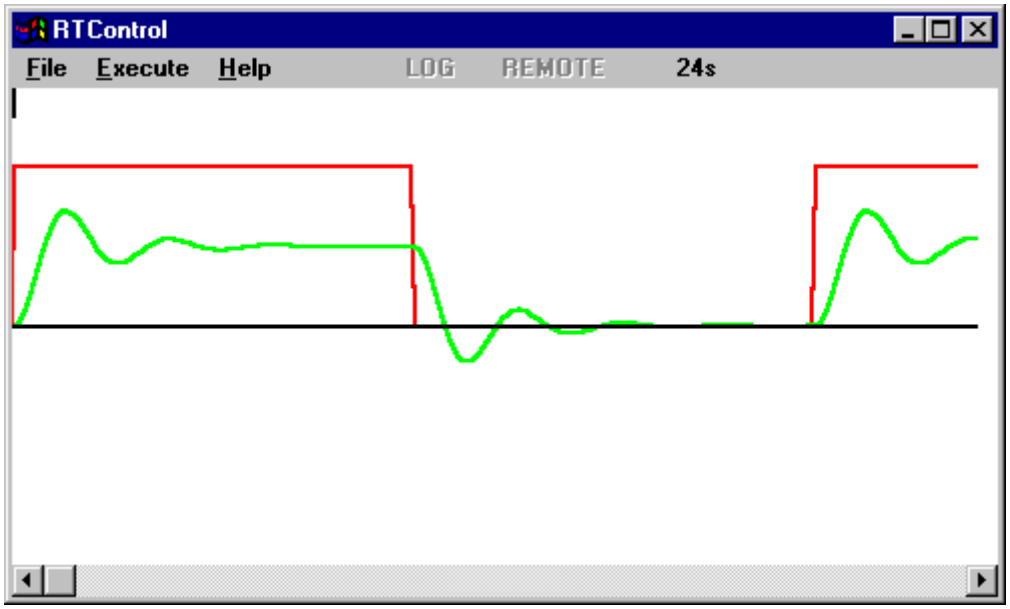


Aufgabe 3.1a

Die zugehörigen Programmdateien sowie das zugehörige Makefile finden Sie auf meiner Web-Seite "www.it.fht-esslingen.de/~zimmerma/vorlesungen/rt" auf dem PC-Server des Rechenzentrums. Das Programm und das Makefile sind für den 16bit Borland-C-Compiler BCC geschrieben (getestet mit Version 3.1, 4.5 und 5.0).

Kopieren Sie die Programmdateien in ein eigenes Verzeichnis und übersetzen Sie die Dateien, indem Sie von der DOS-Kommandozeile aus (z.B. in einem DOS-Fenster in Windows) 'make' aufrufen. Falls der Borland-C-Compiler nicht über die DOS-Pfadangabe erreichbar ist, editieren Sie die Datei MAKEFILE und tragen Sie dort den Pfad zum C-Compiler ein. Falls Sie einen anderen Compiler (z.B. BCC32 oder Visual C++) verwenden wollen oder andere Probleme haben, lesen Sie die zugehörige Datei READ.M).

Starten Sie nach erfolgreicher Übersetzung von Windows aus das Programm RTC.exe und wählen Sie den Menüpunkt Execute-Run-Ok aus. Sie sollten folgende Bildschirmdarstellung erhalten:



Halten Sie das Programm mit dem Menüpunkt Execute-Stop wieder an.

3.2 Programmbeschreibung

Da in diesem Laborversuch die regelungstechnische Fragen im Vordergrund stehen sollen, betrachten wir im folgenden lediglich die oberste Schicht (Datei RTC.c). Das 'Innenleben' der übrigen Schichten wird nur grob beschrieben. Alle von Ihnen in diesem Laborversuch durchzuführenden Programmänderungen betreffen nur diese oberste Schicht.

Den Programmcode von RTC.c finden Sie auf den folgenden Seiten. Die hier aufgeführten Punkte beziehen sich auf die mit [...] markierten Stellen im Programm:

[1] Die **Ein- und Ausgangssignale**, einige interne Signale sowie die Parameter **aller Blöcke** werden **als globale Signale** definiert (Abschnitt [1] im Programm).

[2] Als **Datentyp für** die **Signale** wird **SIGNAL**, als **Datentyp für** die **Parameter** wird **PARAM** definiert, beide Datentypen werden zunächst als 32bit-Gleitkommazahlen (C-Grundtyp 'float') festgelegt.

[3] Jeder **regelungstechnische Block** des Blockschaltbilds wird **als Prozedur** realisiert. Diese Prozeduren sind alle nach dem Schema

```
void Name_der Prozedur (BOOL init)
{     if (init) { . . .          //Initialisierung der Signale und internen Größen
      } else  { . . .            //Rekursionsgleichungen
      }
}
```

aufgebaut. Beim Start wird die Prozedur zunächst einmal mit init = TRUE aufgerufen und muß dann das Ausgangssignal des Blocks und alle internen Signale auf ihre Anfangswerte setzen. Bei den anschließend periodisch mit der Abtastzeit erfolgenden Aufrufen ist init = FALSE, so daß die Rekursionsgleichungen berechnet werden können.

[4] Die **Abtastzeiten** der einzelnen Blöcke **werden im** Anwenderhauptprogramm **mainRealTime definiert**. Dort werden die einzelnen Blöcke **mit** der Prozedur **registerTask** (Name_der_Task, Abtastzeit_in_ms) als Tasks registriert.

[5] Der Aufruf der Tasks erfolgt (bei gleicher Abtastzeit) in der Reihenfolge ihrer Registrierung. **Daher sollte - soweit möglich - ein Block, dessen Ausgangssignal vom nachfolgenden Block verwendet wird, vor dem nachfolgenden Block definiert werden.** (Bei Rückkopplungen, z.B. beim Regler, kann diese Vorgehensweise allerdings nicht eingehalten werden.)
Um mit (fast) allen Windows-Versionen kompatibel zu sein, wird als Zeitraster des Multitaskingsystems lediglich das Standardzeitraster von 50ms verwendet. **Alle Abtastzeiten müssen** daher **Vielfache von 50ms sein.**

[6] Zur graphischen Darstellung der Signale ist im Multitaskingsystem ein "4-Kanal-**Oszilloskop**" realisiert, das immer **als letzte Task** registriert werden muß.

[7] Die **Eingangssignale des Oszilloskops müssen definiert werden** (Struktur oszi.Kanal1 bis ozsi.Kanal4, bitte beachten Sie, daß dabei die Adressen der Signale **mit '&...'** angegeben werden müssen!).

RTC.c

```
//---Typdefinitionen----------------------------------------------------------------------------------------------
typedef         float           SIGNAL;                                                          [2]
typedef SIGNAL          PARAM;

//---Konstanten---------------------------------------------------------------------------------------------------
#define GRAPHICS                                        //Graphikanzeige
#define YMAX  1.5                                       //max.Signalamplitude im Graphikfenster ±Ymax

//---Funktionsprototypen------------------------------------------------------------------------------------------
void registerTask(void (*task)(), int abtastZeit);      //Anmelden einer Task bei der Taskverwaltung
void mainRealTime(void);                                //Hauptprogramm für den Anwender
void oszilloskop (void);                                //Meßwerterfassung

//---Include-Dateien----------------------------------------------------------------------------------------------
#include "RTCtask.h"                                    //Multitaskingsystem (Ablaufsteuerung)
#include "RTCgui.h"                                     //Windows-Schnittstelle und Bedienoberfläche

//### Signale, Regler- und Streckenparameter (Globale Variable) ######################## [1]
//---Rechteckgenerator--------------------------------
PARAM           maxWert =1,                             //Maximalwert
                minWert  =0;                            //Minimalwert

//---P-Regler-----------------------------------------------------
SIGNAL          w;                                      //Sollwerteingang
PARAM                   kp=1;                                   //Reglerverstärkung

//---PT1-Glied----------------------------------------------------
SIGNAL          u,                                      //Eingang
                e,                                      //Ausgang
                de;                                     //Hilfsgröße
PARAM           v1=1,                                   //Verstärkung
                T_T1=0.4;                               //relative Zeitkonstante T/T1

//---PT2-Glied----------------------------------------------------
SIGNAL  //      e,                                      //Eingang = Ausgang des PT1-Glieds
                y,                                      //Ausgang
                x, dx, dy;                              //internes Signal und Hilfsgrößen
PARAM           v=1,                                    //Verstärkung
                T_T0=0.1,                               //relative Zeitkonstante T/T0
                _2D=2*0.7;                              //Dämpfung

//### Regelungstechnische Funktionen ############################################## [3]
//---Rechteckgenerator--------------------------------
void rechteckGenerator(BOOL init)
{       if (init)  { w=minWert;                         //Initialisierung
        } else     { if (w) w=minWert; else w=maxWert;  //Rechtecksignal erzeugen
        }
}
//---P-Regler-----------------------------------------------------
void p (BOOL init)
{       if (init)  { ;                                  //Initialisierung
        } else     { u=kp*(w-y);                        //Rekursionsgleichung
        }
}
```



```
//---PT1-Glied----------------------------------------------------
void pt1 (BOOL init)
{       if (init)  { e=0;                                   //Initialisierung
        } else   { de=T_T1 * (v1 * u - e);                  //Rekursionsgleichung
                   e=e+de;
        }
}
//---PT2-Glied----------------------------------------------------
void pt2 (BOOL init)
{       if (init)  { y=0; x=0;                              //Initialisierung
        } else   { dy=T_T0 * x;                             //Rekursionsgleichungen
                   dx=T_T0 * (v * e - y - _2D * x);
                   y=y+dy;
                   x=x+dx;
        }
}

//### Anwender-Hauptprogramm ################################################  [4]
void mainRealTime(void)
{
//---Registrieren der Tasks und Definition der Abtastzeiten (in ms)----------
        registerTask (rechteckGenerator, 10000);    //Periodendauer 2 * 10000ms   [5]
        registerTask (pi,   500);                   //Abtastzeit 500ms
        registerTask (pt1, 50);                     //Abtastzeit   50ms
        registerTask (pt2, 50);                     //Abtastzeit   50ms
        registerTask (oszilloskop, 100);            //Abtastzeit 100ms            [6]

//---Eingangssignale des Oszilloskops--(Oszilloskop sollte immer als letzte Task definiert werden!)
        oszi.Kanal1 = &w;                           // 1.Kanal: Signal w          [7]
        oszi.Kanal2 = &y;                           // 2.Kanal: Signal y
        oszi.Kanal3 = &nc;                          // 3.Kanal: nicht angeschlossen (Signal nc)
        oszi.Kanal4 = &nc;                          // 4.Kanal: nicht angeschlossen
}
```

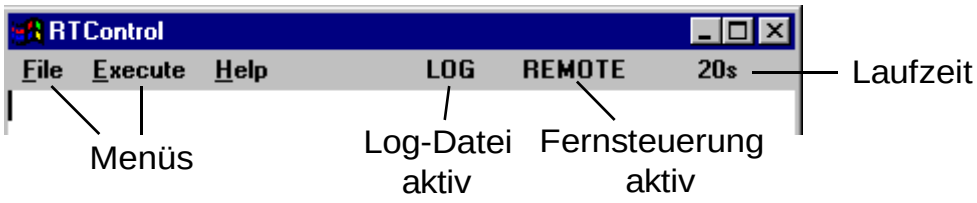
Hinweis: Die in der Ihnen vorliegenden Datei verwendeten Regler- und Streckenparameter können sich von den hier dargestellten Werten unterscheiden.

Das **Multitasking-System** (in der Datei RTCtask.h) selbst ist sehr einfach aufgebaut. Alle Tasks werden in eine Taskliste eingetragen (Funktion **registerTask**). Bei der Initialisierung des Multitasking-Systems wird diese Liste durchlaufen und jede Task, dh. jeder Block, einmal mit dem Parameter init = TRUE aufgerufen. Weiterhin wird für jede Task ein Zähler initialisiert (Funktion **initializeAllTasks**). Die Kernfunktion **scheduleAndDispatch** des Multitasking-Systems wird vom Windows-Systemzeitgeber alle 50ms (Standardzeitraster, exakt eigentlich 55ms) als sogenannte Callback-Funktion (eine Art synchronisierter Interrupt) aktiviert. Bei jeder Aktivierung werden die Zähler der registrierten Tasks inkrementiert. Beim Erreichen der Abtastzeit wird die jeweilige Task aufgerufen und der Zähler zurückgesetzt.

(Hinweis: Unter Verwendung der sogenannten Multimedia-Timer sind auf schnellen Rechnern Abtastzeiten bis zu 1ms möglich, allerdings unterliegen diese Abtastzeiten einem sehr großen Jitter. Diese Option (#define MMTimer) ist im Programm vorhanden, im Laborversuch wird aber nur mit T = 50ms gearbeitet.

3.3 Bedienungsanleitung für RTC

Die in der Datei RTCgui.h realisierte Bedienoberfläche bietet folgende Möglichkeiten:



• Menü **Execute**	
Run	**Starten** des Multitaskingsystems
Stop	**Anhalten**
Continue	**Fortsetzen**
• Menü **File**	
Graphics	Wenn dieser Menüpunkt ausgewählt ist, erfolgt eine **graphische Ausgabe** innerhalb des RTC-Fensters. Wenn Sie das Fenster für eigene Ein-Ausgabeoperationen verwenden wollen, wählen Sie diesen Menüpunkt ab oder kommentieren Sie '#define GRAPHICS' in RTC.c aus. Die Grafikfunktion ist sehr primitiv. Der erste Kanal wird rot, der zweite grün, der dritte blau und der vierte gelb dargestellt. Es gibt keine automatische Skalierung. Die Nulllinie ist in Fenstermitte, die Fensterhöhe entspricht der Signalamplitude ±YMAX.
Log to file	Wenn dieser Menüpunkt ausgewählt ist, werden die **Oszilloskop-Kanäle in** einer **ASCII-Datei protokolliert**. Die Datei ist für die spätere Weiterverarbeitung mit MATLAB vorgesehen. Der Text 'LOG' im Hauptmenü zeigt an, daß die Protokollierung aktiv ist. Verwenden Sie für die Log-Datei möglichst den voreingestellten Namen RTC.log.
Exit	Programm **beenden**

3.4 Fernsteuerung von RTC über das MATLAB-Skript RTCshow.m

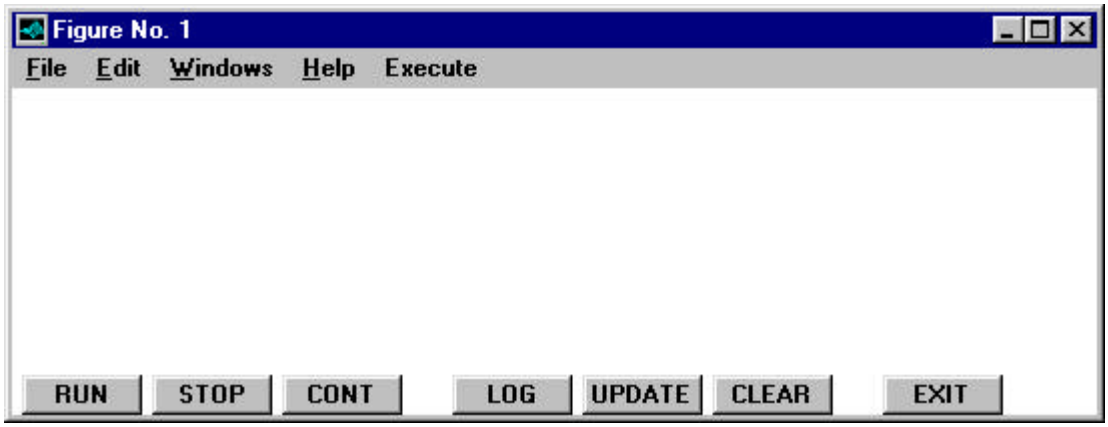
Da die Graphikanzeige in RTC sehr primitiv ist, kann eine **Fernsteuerung und Graphikanzeige über MATLAB** erfolgen.

Aufgabe 3.1b

Beenden Sie RTC.exe, falls es noch läuft. Starten Sie MATLAB. Wechseln Sie in Ihr Verzeichnis und rufen Sie das MATLAB-Skript 'RTCshow' auf.

Dieses MATLAB-Skript startet RTC.exe neu und öffnet zusätzlich das auf der folgenden Seite dargestellte MATLAB-Grafikfenster. Im RTC-Fenster wird die Anzeige REMOTE (rechts oben im Hauptmenü) eingeschaltet.

MATLAB-Grafikfenster zur Fernsteuerung von RTC



Von der unteren Button-Leiste dieses Fenster aus kann RTC ferngesteuert werden:

RTC-Fenster	MATLAB-Fenster	
Execute - Run	**RUN**	
Execute - Stop	**STOP**	
Execute - Cont	**CONT**	
File - Log to File	**LOG**	wählen Sie immer den Dateinamen RTC.log !
File - Exit	**EXIT**	
	CLEAR	**Löschen** der MATLAB-**Graphik**anzeige
	UPDATE	**Einlesen** der von RTC mit LOG erstellten **Protokolldatei** RTC.log

Wenn Sie das Multitaskingsystem vom Matlab-Fenster aus mit RUN starten, erhalten Sie eine kontinuierlich mitlaufende Graphikanzeige der vier in RTC definierten Oszilloskopkanäle. Die Messkanäle stehen als Variablen t, k1, ..., k4 in MATLAB zur Verfügung, so daß Sie sie mit den üblichen MATLAB-Befehlen plot(), ... beliebig neu skalieren, einzelne Kanäle herauszeichnen können usw., nachdem Sie das Multitaskingsystem mit STOP angehalten haben.

Die Fernsteuerung verwendet den Windows-RPC-Mechanismus 'Dynamic Data Exchange' (DDE). Leider ist die Matlab-DDE-Implementierung ziemlich 'wackelig'. **Bitte beachten Sie** daher folgendes, wenn Sie die Fernsteuerung verwenden:

- RTC.exe immer über RTCshow starten.
- Beim Beenden zuerst das Matlab-Graphikfenster über EXIT schließen.
- Wenn MATLAB 'abstürzt' (allgemeine Schutzverletzung), Windows neu starten, ein Neustart von MATLAB allein genügt in der Regel nicht.
- **Wenn** Sie ständig **Probleme mit der Grafikanzeige und/oder der Fernsteuerung** haben, können Sie sie **dauerhaft abschalten**, indem Sie die Datei RTCshow.m editieren. Setzen Sie dort die Variable 'option = 2':

  option = 0 Fernsteuerung und Graphikanzeige zur Laufzeit

option = 1 nur Fernsteuerung, keine Graphikanzeige zur Laufzeit
option = 2 nur Einlesen von Protokoll-(Log-)dateien

Die Buttons im MATLAB-Graphikfenster werden dann teilweise nicht dargestellt und Sie müssen die Log-Datei verwenden. Jedesmal wenn Sie auf den Button 'UPDATE' klicken wird die Log-Datei neu eingelesen und Sie erhalten eine neue Graphikanzeige, auch während RTC läuft.

- RTC enthält eine experimentelle Möglichkeit, über TCP/IP statt über DDE zu kommunizieren sowie ein JAVA-Programm statt MATLAB als Anzeigeprogramm einzusetzen (siehe Datei READ.ME).

Aufgabe 3.2

Fertigen Sie eine Sicherungskopie von RTC.c an. Ersetzen Sie dann den P-Regler durch einen PI-Regler ($k_P = 1$, Abtastzeit 500ms). Wählen Sie durch Ausprobieren T/T_n so, daß sich ein Überschwingen von unter 20% ergibt und daß der Regelkreis innerhalb von 10s annähernd (Toleranzbereich 5%) eingeschwungen ist. Als Startwert eignet sich z.B. $T/T_n = 0,1$.

Sie können die Änderung eines Reglerparameters auch durchführen, ohne daß Sie das Programm jedes mal neu compilieren. Fügen Sie einfach in den Initialisierungsteil des Blocks, dessen Parameter Sie ändern wollen, die entsprechenden Ein-/Ausgabebefehle ein. Hier also:

```
char    puffer[80];
void pi(BOOL init)
{       if (init) {  ...                                    //Initialisierung
                     printf("neuer Wert für T_Tn: ");       //Änderung von T/Tn
                     gets(puffer);
                     T_Tn=atof(puffer);
        } else   {  ...                                     //Rekursionsgleichung
        }
}
```

Beachten Sie dabei, daß Sie in einem Windows-Programm mit graphischer Bedienoberfläche die Standard-C-Ein-/Ausgabebefehle eigentlich gar nicht verwenden können. Die WINIO-Library bildet aber einige dieser Funktionen nach:

puts, putchar, fputchar, printf für die Ausgabe,
gets, getchar, fgetchar für die Eingabe

Da mit den Eingabefunktionen nur Zeichen oder Strings eingelesen werden können, muß mit atoi bzw. atof

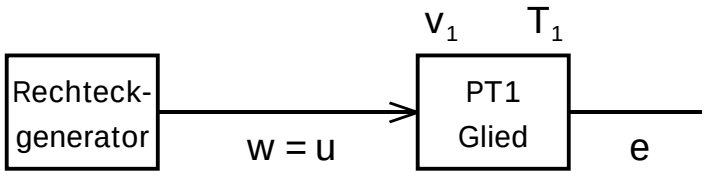
eine Umwandlung in eine Integer- bzw. Float-Zahl erfolgen.

4. Festkommaarithmetik für die Realisierung auf Mikrocontrollern

Im Gegensatz zu High-End-Mikroprozessoren, wie sie in PCs und Workstations eingesetzt werden, besitzen die in der Steuer- und Regelungstechnik, z.B. in Kfz-Motormanagement-, ABS- oder Maschinensteuerungen, verwendeten Mikrocontroller in der Regel keinen Gleitkommaprozessor. Auf solchen Mikrocontrollern müssen arithmetische Operationen mit Gleitkommazahlen durch Funktionen des C-Compilers "emuliert" werden, dh. mit Ganzzahl-(Integer)-arithmetik nachgebildet werden. Die Rechengeschwindigkeit mit solchen Gleitkomma-Softwarebibliotheken ist um den Faktor 10 bis 100 schlechter als mit arithmetischen Coprozessoren.

Bei den in der Steuerungs- und Regelungstechnik üblichen Echtzeitanwendungen ist es daher häufig notwendig, die Algorithmen so zu verändern, daß sie ausschließlich Berechnungen mit Ganzzahlarithmetik verwenden. Wir wollen die sich daraus ergebenden Konsequenzen an einem Beispiel betrachten:

Aufgabe 4.1



Im folgenden soll die Sprungantwort des PT1-Glieds gemessen werden. w und e sollen über das 'Oszilloskop' angezeigt werden. Die Abtastzeit für das PT1-Glied und das Oszilloskop sei 50ms, die relative Zeitkonstante ist $T/T_1 = 0{,}125$. Die Definition der Signale und Parameter (Sprung von 0 auf 1, $v_1 = 1$) als 'float' bleibt zunächst erhalten. Fertigen Sie eine weitere Sicherungskopie von RTC.c an, ändern Sie Ihr Programm entsprechend ab und nehmen Sie die Sprungantwort auf.

4.1 Normierung der Signale

Wenn Sie diese Sprunganwort betrachten, ist klar, daß wir beim Übergang zur Ganzzahlarithmetik die Skalierung ändern müssen, da alle Signale bisher im Wertebereich zwischen 0 und 1 liegen. Um eine ausreichende Auflösung von n bit, zu erhalten, bietet es sich an, alle Signale, dh. hier w und e, auf $1/2^n$ zu normieren, dh. mit 2^n zu multiplizieren.

Die Rekursionsgleichung des PT1-Glieds war nach Gl. (1.10),

$$e(k) = e(k\text{-}1) + \frac{T}{T_1}\,[\, v_1 \cdot w(k) - e(k\text{-}1)\,] \qquad (4.1)$$

wobei berücksichtigt wurde, daß das Eingangssignal hier mit w und das Ausgangssignal mit e bezeichnet wurde.

Bezeichnet man die normierten Größen als $e' = e \cdot 2^n$ und $w' = w \cdot 2^n$, so ergibt sich statt (4.1):

$$e'(k) = e'(k-1) + \frac{T}{T_1} [v_1 \cdot w'(k) - e'(k-1)]$$

(4.2)

Da dieselbe Normierung sowohl für das Ein- als auch für das Ausgangssignal des PT1-Glieds durchgeführt wird, bleiben v_1 und T_1 unverändert.

In praktischen Anwendungen liegt es nahe, n gleich der Anzahl der Bits der eingesetzten Analog-Digital- bzw. Digital-Analogumsetzer zu wählen. Deren Auflösung wird im wesentlichen von der gewünschten und möglichen (!) Genauigkeit der Istwerterfassung bestimmt. Mit $n = 10$ ergibt sich bereits eine Auflösung von unter $1/2^{10} = 0{,}1\%$ des Meßbereichs, Auflösungen von mehr als $n = 12$ sind in regelungstechnischen Anwendungen nur dann erforderlich, wenn ein sehr großer Dynamikbereich abgedeckt werden muß.

Aufgabe 4.2a

Hier soll $n = 10$, dh. $2^n = 1024$ sein. Berücksichtigen Sie dies, indem Sie in Ihrem Programm folgende Änderungen vornehmen:

```
#define  YMAX      1.5*1024      //Wertebereich für die Bildschirmgrafik
PARAM    maxWert = 1024,         //Sprunghöhe des Rechteckgenerators
```

Alle übrigen Werte bleiben zunächst unverändert.

Überzeugen Sie sich, daß sich, vom anderen Endwert abgesehen, dieselbe Sprungantwort des PT1-Glieds ergibt wie in Aufgabe 4.1.

Aufgabe 4.2 b

Wir wollen jetzt zur Ganzzahlarithmetik übergehen. Ändern Sie dazu die Typdefinition der Signale und Parameter in

```
typedef      int     SIGNAL;     // 'int' statt 'float'
```

und ermitteln Sie wiederum die Sprungantwort. Diese Sprungantwort müßte eigentlich genauso aussehen wie in 4.2a ! Ist das wirklich der Fall? Wenn nein, weshalb nicht?

Bei Betrachtung der Rekursionsgleichung des PT1-Glieds und der Zahlenwerte der Parameter $v_1 = 1$ und $T/T_1 = 0.125$ wird klar, weshalb die Sprungantwort in Aufgabe 4.2b nicht den erwarteten Verlauf haben konnte:

$$e'(k) = e'(k-1) + \frac{T}{T_1} [v_1 \cdot w'(k) - e'(k-1)] \qquad ((4.2))$$

Während die Multiplikation mit dem Wert v_1 in unserem Beispiel auch bei Ganzzahlarithmetik unkritisch war, da hier $v_1 = 1$ ohnehin ganzzahlig war, **führt die Multiplikation mit T/T_1 bei Ganzzahlarithmetik grundsätzlich zu unbrauchbaren Ergebnissen**. Um die Sprungantwort des PT1-Gliedes, die ja nur zu diskreten Zeitpunkten berechnet werden kann, sinnvoll nachzubilden, muß stets

$$T < T_1$$

sein. Dies bedeutet aber, daß auch stets $\frac{T}{T_1} < 1$

ist, dh. bei Ganzzahlarithmetik ist der Faktor, mit dem die Klammer [ ] multipliziert wird, gleich Null.

Auf den ersten Blick liegt es nahe, statt der Multiplikation mit $T/T_1 < 1$ einfach eine Division durch den Kehrwert $T_1/T > 1$, hier also durch $1/0{,}125 = 8$, durchzuführen. Dagegen sprechen jedoch zwei Gründe:

- Multiplikationen und Divisionen sind auch bei Ganzzahlarithmetik erheblich aufwendiger als Additionen oder Subtraktionen. Insbesondere Divisionen benötigen daher bei praktisch allen heutigen Mikroprozessoren wesentlich mehr Zeit. Zur Beschleunigung von Multiplikationen haben viele Mikroprozessoren zusätzliche Schaltnetze bzw. Schaltwerke ("Hardwaremultiplizierer"). Divisionen dagegen werden praktisch immer durch eine größere Anzahl von langsamen Mikroprogrammschritten realisiert. Manche Prozessoren, z.B. Signalprozessoren, wie sie bei der Sprach- und Audiosignalübertragung eingesetzt werden, haben in der Regel sogar gar keinen Divisionsbefehl. Divisionen sollten daher im Echtzeitbetrieb so weit wie möglich vermieden werden.

  Beispiele: (Best-Case Werte, alle Operanden in Registern)

	Addition	Multiplikation	Division	
Pentium (100MHz) (Mikroprozessor)	0,1µs	1,1µs	3µs	Ganzzahl
	0,3µs	0,3µs	3,9µs	Gleitkomma (mit Coprozessor)
80C166 (40MHz) (Mikrocontroller)	0,1µs	0,5µs	1µs	Ganzzahl

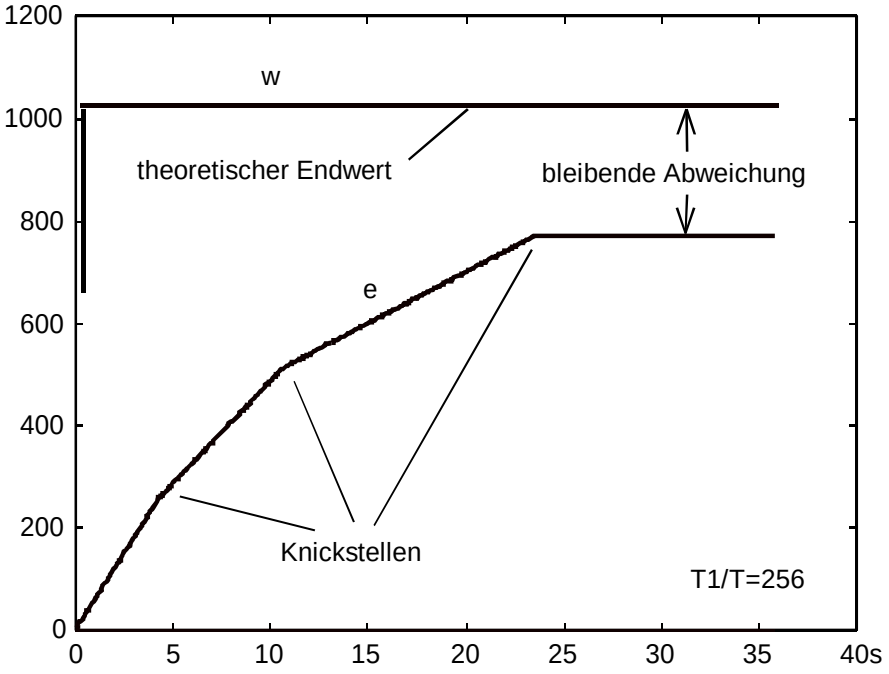
- Bei der ganzzahligen Division kann es zu erheblichen Rundungsfehlern kommen. Als Beispiel wird die Rekursionsgleichung des PT1-Glieds (4.2) betrachtet, die so umgestellt wurde, daß eine Division verwendet wird:

$$e'(k) = e'(k\text{-}1) + [\, v_1 \cdot w'(k) - e'(k\text{-}1) \,] \;/\; \frac{T_1}{T} \qquad (4.3)$$

  Durch die ganzzahlige Division wird die zu e'(k-1) addierte zeitliche Änderung stark gerundet. Sobald $[\, v_1 \cdot w'(k) - e'(k\text{-}1) \,] < \frac{T_1}{T}$

wird, fällt der Ausdruck in der Klammer [ ] sogar ganz „unter den Tisch “, da das ganzzahlige Ergebnis der Division dann 0 ist. Dies führt dazu, daß das Ausgangssignal des PT1-Glieds den Endwert nicht mehr erreicht und für $k \to \infty$ eine bleibende Abweichung von $v_1 \cdot w' - e' = \frac{T_1}{T}$ auftreten würde.

Um diesen Effekt besonders augenfällig zu demonstrieren, wurde für das folgende Bild die Zeitkonstante $T/T_1 = 1/256$ gewählt. Im Ausgangssignal e sind deutlich mehrere Knickstellen sowie die bleibende Abweichung zu sehen:



4.2 Normierung der Zeitkonstanten (und Verstärkungsfaktoren)

Aus den oben genannten Gründen wird die Multiplikation also beibehalten und eine weitere Normierung für die Zeitkonstante eingeführt. In der Rekursiongleichung

$$e'(k) = e'(k\text{-}1) + \frac{T}{T_1} [\, v_1 \cdot w'(k) - e'(k\text{-}1) \,] \qquad ((4.2))$$

wird die bezogene Zeitkonstante T/T_1 mit einer weiteren Normierungskonstanten M multipliziert. Damit ergibt sich:

$$e'(k) = \{\; e'(k\text{-}1) \cdot M + \frac{T}{T_1} M \cdot [\, v_1 \cdot w'(k) - e'(k\text{-}1) \,] \;\} \;/\; M \qquad (4.4)$$

Dabei wählt man M so, daß $\frac{T}{T_1} M \geq 1$ wird

und $M = 2^m$ eine 2er Potenz ist.

Da $M = 2^m$ eine 2er Potenz ist, kann die „Multiplikation " $e'(k-1) \cdot M$ als Links- und die „Division " $\{...\} / M$ als Rechts-Schiebeoperation um m Stellen durchgeführt werden. Schiebeoperationen sind bei den meisten Mikroprozessoren wesentlich schneller als echte Multiplikationen. Der gesamte Faktor $\frac{T}{T_1} M$ sowie v werden als jeweils ein Parameter abgespeichert, so daß nur zwei echte Multiplikationen notwendig sind.

Aufgabe 4.3

Ändern Sie Ihr Programm aus Aufgabe 4.2b so ab, daß die Rekursionsgleichung nach Gl. (4.4) mit 2 Schiebeoperationen und Multiplikationen realisiert wird. Da hier $T / T_1 = 0.125$ ist, ist $M = 8$ ein geeigneter Wert.

Erreicht die Sprungantwort den theoretischen Endwert $v_1 \cdot w(\infty) = 1024$ oder gibt es eine bleibende Abweichung?

Zufällig wird bei den von uns gewählten Zahlenwerten $T/T_1 \cdot M = 0{,}125 \cdot 8 = 1$, dh. der Faktor ist ganzzahlig und daher exakt zu realisieren. Wäre aber beispielsweise $T/T_1 = 0.1$, dann liese sich der Faktor $T/T_1 \cdot M = 0{,}1 \cdot 8 = 0{,}8$ nicht exakt realisieren, er müßte auf 1 gerundet werden, was einem Fehler der wirksamen Zeitkonstante von 25% entsprechen würde.

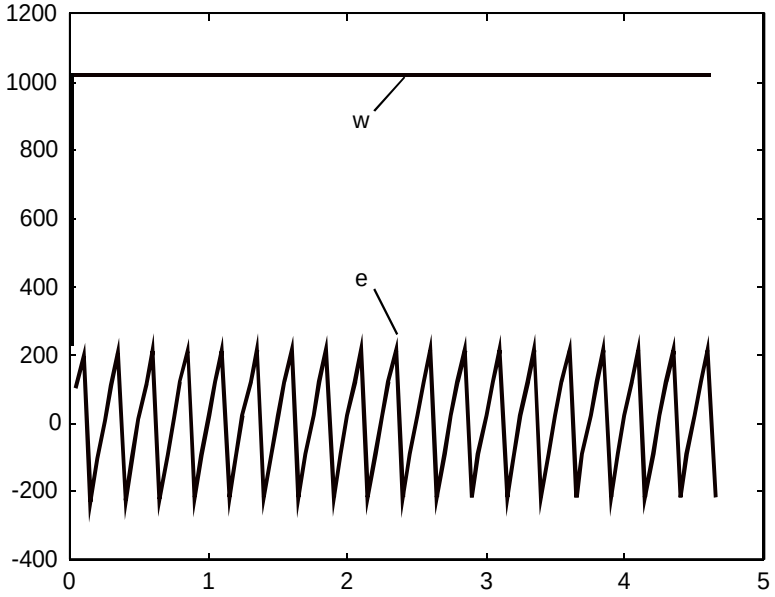
Als Abhilfe verändert man entweder die Abtastzeit T, soweit diese frei wählbar ist, oder man wählt $\frac{T}{T_1} M \gg 1$, um eine ausreichende Genauigkeit zu erhalten. Praktisch genügt in der Regel $\frac{T}{T_1} M > 10$,
wobei M selbst wieder als Zweierpotenz gewählt wird.

Aufgabe 4.4

Ändern Sie Ihr Programm aus Aufgabe 4.3 so ab, daß $T / T_1 = 0{,}1$ ist und wählen Sie $M = 128$. Damit wird $T/T_1 \cdot M = 0{,}1 \cdot 128 = 12{,}8 \approx 13$. (Falls der entstehende Fehler der Zeitkonstante von 1,5% nicht akzeptabel erscheint, muß M noch größer gewählt werden.)
Funktioniert Ihr Algorithmus mit diesen Zahlenwerten tatsächlich immer noch?

4.3 Numerische Überläufe bei Ganzzahlarithmetik

Vermutlich hat Ihre Sprungantwort in Aufgabe 4.4 ähnlich ausgesehen wie dieses Bild:



Da alle Signale und Parameter als 16bit-Integer-Größen definiert wurden, kommt es z.B. bei der Berechnung zu einem Überlauf, sobald $e'(k-1) \cdot M > 32767$ ist, dh. bei $M = 128$ schon bei $e'(k-1) = 256$. Im Gegensatz zu einem Verstärker, der bei Übersteuerung wenigstens sein maximal mögliches Ausgangssignal ausgibt, führt die bei Mikroprozessoren übliche 2er-Komplementdarstellung der Werte dazu, daß die Größen „kippen“, dh. ihr Vorzeichen wechseln. Nur wenige Mikroprozessoren verfügen über eine sogenannte Sättigungsarithmetik, die sich ähnlich verhält wie ein Verstärker. Bei einem Regler ist ein solcher ungewollter Vorzeichenwechsel der Ausgangsgröße katastrophal, da er zu einem Wechsel des Regelsinns führt und aus einer gewünschten Gegenkopplung schlagartig eine Mitkopplung macht. **Überläufe müssen** daher **zwingend vermieden werden**.

Kann man M aus Gründen der gewünschten Auflösung nicht kleiner wählen, so muß man die Rekursionsgleichung trotz der nur 16bit großen Ein- und Ausgangssignale des PT1-Glieds mit 32bit-Arithmetik durchführen. Dazu definiert man die

Zwischengrößen $$\Delta e''(k) = \frac{T}{T_1} M \cdot [\, v_1 \cdot w'(k) - e'(k-1) \,]$$

und $$e''(k) = e''(k-1) + \Delta e''(k)$$

als 32bit-Größen (long), während das Eingangssignal w' und das Ausgangssignal

$$e'(k) = e''(k) / M$$

weiterhin als 16bit-Größen (int) definiert bleiben.

Aufgabe 4.5

Korrigieren Sie Ihr Programm aus Aufgabe 4.4 entsprechend.

Aufgabe 4.6

Im vorigen Beispiel war der Verstärkungsfaktor $v_1 = 1$ ganzzahlig. ***Bei nicht ganzzahligen Werten müssen die Verstärkungsfaktoren in ähnlicher Weise normiert werden wie die Zeitkonstanten.*** *Bauen Sie in Ihr Programm aus Aufgabe 4.5 eine zusätzliche Normierung für v_1 so ein, daß der Verstärkungsfaktor mit 7bit (entsprechend 1%) aufgelöst wird und testen Sie Ihr Programm mit $v_1 = 1{,}25$.*

Wie genau erreichen Sie den theoretischen Endwert von $e' = 1{,}25 \cdot 1024 = 1280$?

Aufgabe 4.7

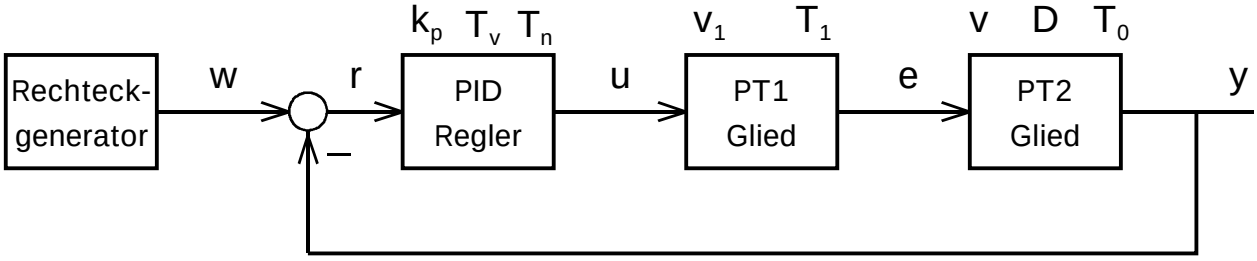
Stellen Sie das gesamte Programm aus Aufgabe 3.1 von Gleitkomma auf Ganzzahlarithmetik um. Wählen Sie die Normierungsfaktoren für die dort vorgegebenen Zeitkonstanten und Verstärkungsfaktoren so, daß sie mit höchstens 1% Fehler aufgelöst werden.

Testen Sie die einzelnen Blöcke, indem Sie für jeden Block (P-Regler, PT1-Glied, PT2-Glied) getrennt die Sprungantwort 'messen'.

Stellen Sie dann die Sprungantwort des geschlossenen Regelkreises dar. Der Zeitverlauf sollte mit dem Signalverlauf bei der Gleitkommaarithmetik ohne erkennbaren Unterschied übereinstimmen.

5. Digitales Differenzieren - Begrenzer

Ersetzen Sie den P-Regler aus Aufgabe 4.7 durch einen PID-Regler.



Die Übertragungsfunktion des zeitdiskreten PID-Reglers gemäß Vorlesung ist:

$$G_{R,PID}(z) = \frac{U(z)}{R(z)} = k_P + k_P \cdot \frac{T}{T_n} \cdot \frac{z}{z-1} + k_P \cdot \frac{T_V}{T} \cdot \frac{z-1}{z}$$

Daraus ergibt sich der Algorithmus $u(k) = u_P(k) + u_I(k) + u_D(k)$

mit dem P-Anteil $u_P(k) = k_P \cdot r(k)$

dem I-Anteil $u_I(k) = u_I(k-1) + k_P \cdot \frac{T}{T_n} r(k)$

dem D-Anteil $u_D(k) = k_P \cdot \frac{T_V}{T} [\, r(k) - r(k-1) \,]$

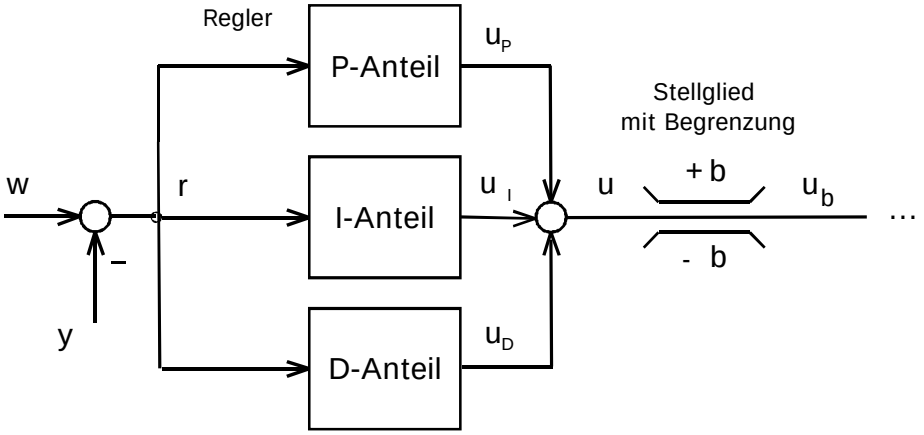
Aufgabe 5a

Bauen Sie den PID-Regler (Abtastzeit $T = 50ms$, $k_P = 4$, $T/T_n = 0.1$, $T_V/T = 4$) in Ihr Programm aus Aufgabe 4.7 ein und ermitteln Sie die Sprungantwort dieses Regelkreises.

Wie Sie aus dem Laborversuch 'Digitale Regler', Abschnitt 8 wissen, reagieren Regelkreise auf Begrenzungen, die bei praktischen Anwendungen im allgemeinen im Stellglied auftreten, empfindlich.

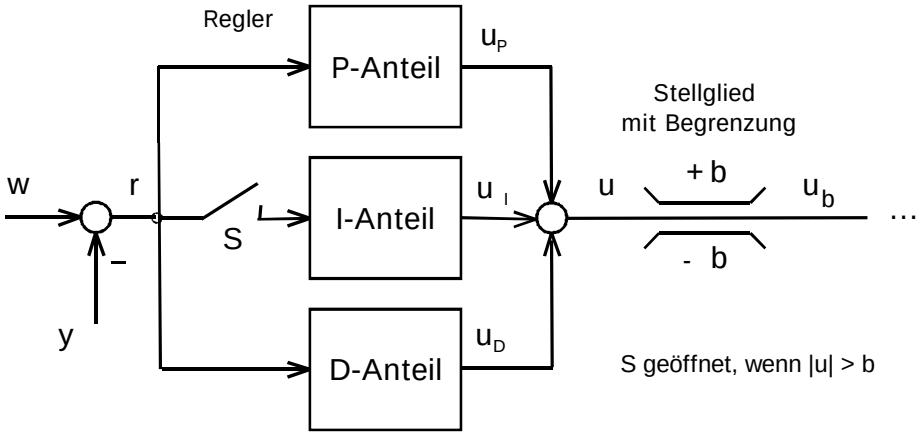
Aufgabe 5b

Bauen Sie in den PID-Regler eine Begrenzung ein, mit der die Ausgangsgröße des PID-Reglers auf den Wert $b = \pm 1024 \cdot 1{,}2 = \pm 1229$ begrenzt wird und ermitteln Sie die Sprungantwort des Regelkreises mit dieser Begrenzung.



Aufgabe 5c

Bauen Sie in den schliesslich in den PID-Regler auch noch die im Versuch Digitale Regler beschriebene Anti-Windup-Maßnahme ein, bei der die Berechnung des I-Anteils 'eingefroren' wird, sobald die Ausgangsgröße des PID-Reglers den Wert $b = \pm 1024 \cdot 1{,}2 = \pm 1229$ überschreitet und ermitteln Sie wiederum die Sprungantwort des Regelkreises.



Anhang:

1. Alternative Form des PID-Algorithmus

Der PID-Algorithmus nach Abschnitt 5 und die zugehörige z-Übertragungsfunktion wurden in der Vorlesung aus der Differentialgleichung des zeitkontinuierlichen PID-Reglers hergeleitet. Aus der z-Übertragungsfunktion kann man den Algorithmus auch rückwärts ableiten:

$$G_{R,PID}(z) = \frac{U(z)}{R(z)} = k_P + k_P \cdot \frac{T}{T_n} \cdot \frac{z}{z-1} + k_P \cdot \frac{T_V}{T} \cdot \frac{z-1}{z}$$

$$= k_P \cdot \frac{z(z-1) + \frac{T}{T_n} z^2 + \frac{T_V}{T} (z-1)^2}{z(z-1)}$$

$$\rightarrow \quad U(z)\,[1 - z^{-1}] = R(z) \cdot k_P \cdot [\,1 + \frac{T}{T_n} + \frac{T_V}{T} - z^{-1}\,(1 + 2\frac{T_V}{T}) + z^{-2}\,\frac{T_V}{T}\,]$$

$$\rightarrow \quad u(k) = u(k-1) + \cdot k_P \cdot [\,1 + \frac{T}{T_n} + \frac{T_V}{T}\,] \cdot r(k) - k_P \cdot (1 + 2\frac{T_V}{T})\, r(k-1) + \frac{T_V}{T}\, r(k-2)$$

In dieser Darstellung sind P, I- und D-Anteil nicht mehr getrennt zu erkennen, so daß diese Form des Algorithmus, die häufig in der Literatur zu finden ist, für die Realisierung von Anti-Windup-Maßnahmen ungeeignet ist.

2. 32bit-Compiler

Die im Laborversuch gewählten Zahlenwerte und Normierungen sind so gewählt, daß die dargestellten Rundungs- und Überlaufeffekte bei 16bit-Integer-Arithmetik sichtbar werden. Wenn Sie einen 32bit-Compiler (z.B. Borland bcc32.exe oder Microsoft Visual C++) verwenden, gehen die meisten dieser Effekte verloren, weil der 32bit Compiler viele Zwischenergebnisse als 32bit Größen berechnet (automatisches Type Casting), auch wenn Sie die Signale und Parameter als 16bit short int definieren. Durch andere Wahl der Zahlenwerte und Normierungen könnte man die Effekte auch mit 32bit-Arithmetik darstellen, aber die Zahlen würden viel zu unhandlich werden (bis zu 10stellige Dezimalzahlen), so daß Sie unbedingt einen 16bit Compiler verwenden sollten.

Falls dies nicht möglich ist, kann die 16bit-Arithmetik auch mit einem 32bit-Compiler simuliert werden, indem die Signale als C++Klassen definiert und die Arithmetikoperatoren geeignet überladen werden. Einzelheiten dazu siehe in der Datei READ.ME.

3. Grafikanzeige mit Java-Viewer statt Matlab

Falls Sie den Microsoft Internet Explorer verwenden, können Sie zur Grafikanzeige statt MATLAB auch ein JAVA-Programm verwenden. Einzelheiten siehe READ.ME.