Informatikwerkstatt, Foliensatz 12 Motorregelung

G. Kemnitz

14. Dezember 2020

Inhalt:

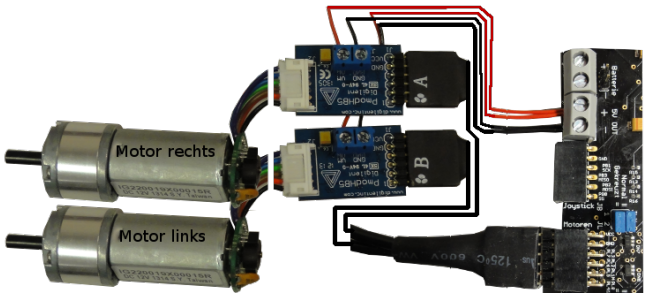
Inhaltsverzeichnis

Interaktive Übungen:

1. Messung der Umdrehungsgeschwindigkeit (rotmess)
2. Bestimmung der Motorkennline
3. PI-Regler (test_regelung)

1 Wiederholung »test_rotmess«

Testprogramm »test_rotmess« starten



- 2×H-Brücke PmodHB5 über Y-Kabel an JL.
- Motoren an die H-Brücken stecken.
- Spannungsversorgungdrähte anschrauben.

- PmodUSBUSART an JH oben und USB-Verbindung zum PC.
- JHX und JLX auf »gekreuzt (=)«.
- Projekt »F11-rotmess\rotmess« öffnen, übersetzen, starten.
- HTerm starten. 8N1 9600 Baud. COM einstellen. Connect.

Testbeispiele mit HTerm

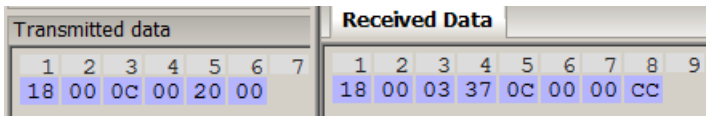
Das Programm »test_rotmess.c« wartet vom PC auf 6 Bytes:

- Byte 1 und 2: Pulslänge Motor R (OCR5B),
- Byte 3 und 4: Pulslänge Motor L (OCR5C),
- Byte 5 und 6: Pulsperiode Motor R und L (OCR5A).

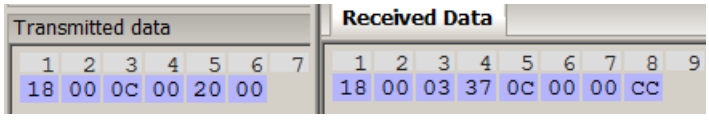
und sendet nach Bewegungsabschluss 8 Bytes zurück:

- Byte 1 und 2: empfangene Pulslänge Motor R:
- Byte 3 und 4: Winkelschritt pro s Motor R,
- Byte 5 und 6: empfangene Pulslänge Motor L:
- Byte 7 und 8: Winkelschritt pro s Motor L.

Ein- und Ausgabe über HTerm:

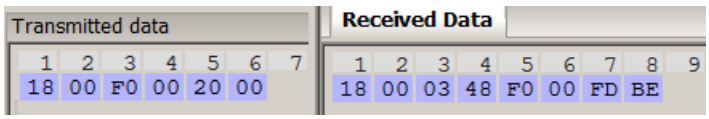


Testbeispiel 1:



PWM_R	speed_R	PWM_L	speed_L
$\frac{0x1800}{0x2000} = 75\%$	$\frac{0x337}{228^*} = 3{,}61\frac{U}{s}$	$\frac{0x0C00}{0x2000} = 37{,}5\%$	$\frac{0x0CC}{228^*} = 0{,}89\frac{U}{s}$

Testbeispiel 2:



PWM_R	speed_R	PWM_L	speed_L
$\frac{0x1800}{0x2000} = 75\%$	$\frac{0x348}{228^*} = 3{,}68\frac{U}{s}$	$\frac{-0x1000}{0x2000} = -50\%$	$\frac{-0x242}{228^*} = -2{,}54\frac{U}{s}$

Absolute Pulsweite in den Beispielen :0x2000/8 MHz $\approx$ 1 ms

* 12 Schritte je Motordrehung und 19 Motorumdrehungen je Radumdrehung.

2 Motorkennlinie

Bestimmung der Motorkennlinien

Für die Konzeption der Fahrzeugsteuerung wird die Funktion

$$\omega = f(\eta, \ldots)$$

(ω – Winkelgeschwindigkeit; η – relative Pulsweite; ... – weitere Einflüsse wie Pulsperiode, Versorgungsspannung, ...) benötigt.
Bestimmbar mit HTerm und vielen Einzelmessungen.

Alternative: Programmgesteuert mit Python-Programm.

PC-Programm »rotmess.py«

```
wiederhole für PWM-Periode ∈ {2ms, 1ms, 0,5ms}
  wiederhole für pwm=-100% bis 100% in 5%-Schritten
    bestimme Winkelgeschwindigkeit
    Ausgabe der Werte als Tabelle
    Sammeln der Werte von Motor R für eine Graphik
```

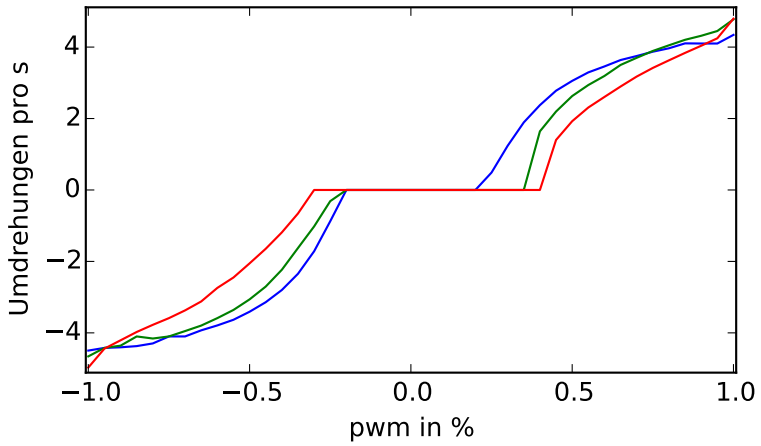
```
import serial                #Modul serial importieren
ser = serial.Serial("COM9") #COM anpassen!
import matplotlib.pyplot as plt#Plotfunktion importeren
for Periode in (0x4000,0x2000,0x1000):#Periodenwerte
 pwm_list=[]                 #leere Listen für die gra-
 speed_list=[]               #phisch darzustellenden Werte
 pwm=-1.0
 while pwm<=1.01:            #für pwm = -1 bis 1
```

```
  #Umwandlung der %-Zahl in einen OCR-Wert
  ocr = int(pwm*Periode)

  #OCR-Wert => Byte-Array, Länge 2, signed
  bocr = ocr.to_bytes(2, byteorder='big', signed=True)

  #Periode => Byte-Array, Länge 2, unsigned
  bper = Periode.to_bytes(2, byteorder='big')
  smsg = bocr + bocr + bper # 6-Byte-Array

  ser.write(smsg)           #senden von 6 Bytes
  rstr=ser.read(8)          #auf 8 Bytes warten

  #Bytevektor in Ergebniswerte umrechnen und ausgeben
  pwmR = float(int.from_bytes(rmsg[0:2],
   byteorder='big', signed=True))/Periode * 100
  spR  = float(int.from_bytes(rmsg[2:4],
   byteorder='big', signed=True))/228
  pwmL = float(int.from_bytes(rmsg[4:6],
   byteorder='big', signed=True))/Periode * 100
  spL  = float(int.from_bytes(rmsg[6:8],
   byteorder='big', signed=True))/228
```

```
    print('Periode = %4.2fms'%(Periode/8E3),
      'pwmR = %6.1f'%pwmR + '%', 'spR = %4.2f U/s'%spR,
      ' | ', 'spL = %4.2f U/s'%spL)

    #PWM-Wert und Ergebnisse Motor R an Listen hängen
    pwm_list.append(pwm)
    speed_list.append(spR)
    pwm +=0.05                  #Erhöhung der Pulsbreite um 5%

  #Für jede PWM-Periode Winkelgesch. als xy-Graph ausgeben
  plt.plot(pwm_list, speed_list)

ser.close()                     #COM-Port schliessen
plt.xlabel('pwm in %')          #Achsen beschriften
plt.ylabel('Umdrehungen pro s')
plt.show()
```

Ergebnis

```
Periode=2.05ms pwmR=-100.0% spR=-4.50 U/s | spL=-4.47 U/s
Periode=2.05ms pwmR= -95.0% spR=-4.45 U/s | spL=-4.35 U/s
Periode=2.05ms pwmR= -90.0% spR=-4.41 U/s | spL=-4.32 U/s
...
```



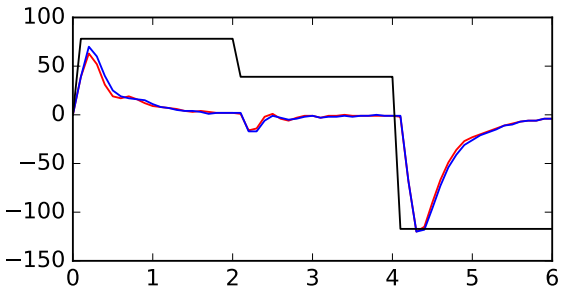
Periode: rot 2 ms, grün 1 ms, blau 0,5 ms

- Die Kennlinie ist nichtlinear mit einem Totbereich zwischen ca. -25% bis +25%.
- Vernünftig steuern lässt sich der Motor nur im Betragsbereich von 1 bis 4 Radumdrehungen pro Sekunde.
- Langsame und genaue Bewegungsvorgaben verlangen eine Regelung.
- PWM-Periode ca. 1 ms ist ein vernüftiger Wert.

3 Motorregelung

Motorregelung

- Bildung der Reglerabweichung durch Subtraktion des Ist-Werts (Position, Geschwindigkeit, ...) vom Soll-Wert.
- Berechnung der neuen Stellgröße aus der aktuellen Stellgröße und der Reglerabweichung so, dass die Reglerabweichung gegen null strebt.



Zweipunktregelung

Am einfachten zu programmierender Regler:

- Soll/Ist-Abweichung zu fahrender Weg für beide Räder:

  ```
  int32_t pos_diff_right, pos_diff_left;
  ```

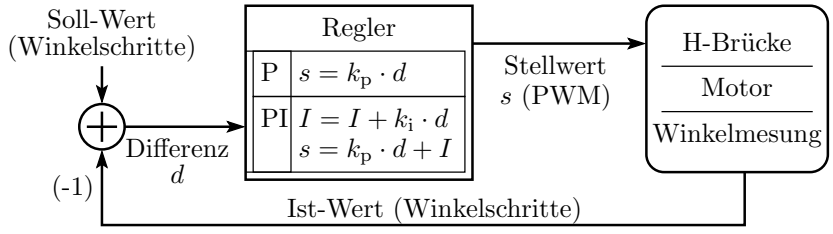
- In der Timer-ISR für die Geschwindigkeitsmessung:
  - Addition von je einem zum innerhalb der ISR-Periode zu fahrenden Weg proportionalen Wert $v_{\mathrm{r/l}}$ mit $\left|v_{\mathrm{r/l}}\right| < v_{\max}$.
  - Subtraktion des Messwerts für die Anzahl der Umdrehungsschritte $\{-1, 0, 1\}$ multipliziert mit $v_{\max}$.
- Wenn vorwärts, wenn Diff. positiv[1], Motor vorwärts an, sonst aus.
- sonst, wenn Differenz negativ[2], Motor rückwärts an, sonst aus.m

---

- Bahnabfahrgenauigkeit gut, aber
- sehr »ruckhaftes« Fahrverhalten. (Bitte selbst ausprobieren.)

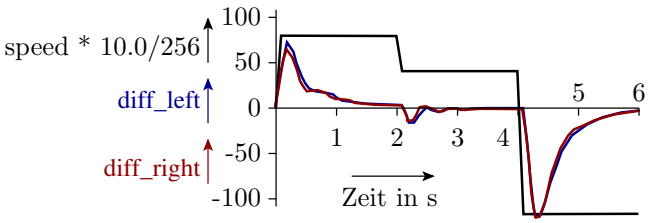
3.1 PI-Regler

Funktion von P- und PI-Reglern



[1] *Rad dreht zu langsam.*
[2] *ZU langsame Rückwärtsdrehung.*

- Stellwert s gleich $k_\mathrm{p}\cdot$ Regelabweichung plus skalierte Summe der Regelabweichung (Integralteil I).
- Je größer k_p desto kleiner Regelabweichung, aber wenn k_p zu klein $\Rightarrow$ Schwingung, d.h. ruckhaftes« Fahrverhalten, auch schlimmer als beim Zweipunktregler.
- Mit einem zusätzlichen Integralanteil I und passendem k_i verringert sich die Soll-/Ist-Abweichung auf nahe null.
- k_p und k_i experimentell bestimmbar.



Nach Änderungssprüngen des Sollwerts kann die Reglerabweichung

- stetig abklingen (sanftes Verhalten),
- eine abklingende Schwingung (schneller) oder
- eine Dauerschwingung sein (gefährlich).

Experimentelle Bestimmung von k_p und k_i (WEB-Suchbegriff: »PI-Regler empirisch einstellen«):

- $k_\mathrm{i}=0$, k_p erhöhen, bis die Reglung schwingt. Davon auf 60%.
- k_i erhöhen, bis die Reglung schwingt, davon auch 60%.

Problem: Man muss die Relerabweichung als Funktion der Zeit experimentell bestimmen.

3.2 Beispielprogramm PI-Regler

Private Variablen

$$\begin{aligned} I &= I + k_\mathrm{I}\cdot d \\ s &= k_\mathrm{P}\cdot d + I \end{aligned}$$

- Implementierung der Reglervariablen als Festkommazahlen mit 8 NKB (Nachkommabits):

```
uint16_t kp, ki;          //Reglerkoeffizient, 8 NKB
int32_t diffR, diffL;     //Schrittdifferenz, 8 NKB
int32_t integR, integL;   //Integralanteile, 8 NKB
int32_t pwmR, pwmL;       //Stellwerte, 8 NKB
int16_t speedR, speedL;   //Sollgeschwindigkeit, 8 NKB
```

- Ausschluss Überlauf: 0x7FFFFF.FF $\Leftrightarrow -$0x800000:

```
void limit(int32_t *val, int32_t max){
 if (*val >  max)
  *val = max;
 else if (*val < -max)
  *val = -max;
}
```

- Multiplikation: »a = limit(((int32_t)b*c)>>8, abs_max);«

Programm für einen Reglungsschritt

```
if (rotmess_get(&sr, &sl)){//wenn neuer Messwert
 if (rotmess_err())        //bei Fehler im Treiber
  lcd_incErr(ERR_RMESS);   //Fehlerzähler erhöhen
 //Ausführung eines PID-Reglerschritts je Motor
 diffR  = diffR + speedR - (sr << 8);
 limit(&diffR, 0x100000);  //Begrenzung der Differenz
 integR = integR + ((ki * diffR)>>8);
 limit(&integR, 0x400000); //Begrenzung Integralteil
 pwmR = (integR>>8) + ((kp * diffR)>>12);
 limit(&pwmR, 0x4000);     //Begrenzung Stellgrösse
 pwm_set_R(pwmR);
 ... //dasselbe für Motor L
 ... //Ausgabe Regelabweichungen und Integralteile
 ... //zur Kontrolle auf das LC-Display
}
```

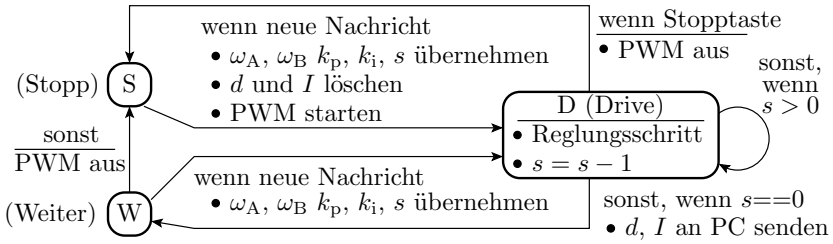
Treiber »rotmess.c« wird so konfiguriert, dass die Funktion

```
uint8_t rotmess_get(int16_t *spR, int16_t *spL);
```

die Winkelschritte für 20 ms liefert ⇒ Regler-Berechnungsinvall 20 ms.

3.3 Fahrzeugsteuerung

Fahrzeugsteuerung als Automat



Automatenzustände:

S	Antriebe gestoppt
D	Abarbeitung einer Teilbewegung
W	Übergang zur nächsten Teilbewegung ohne Stopp

Nachrichtenbestandteile / Beschreibung einer Teilbewegung:

ω_A, ω_B	Soll- Winkelgeschwindigkeit
k_p, k_i	Reglerparamerter
d, I, s	Differenzen (Reglerabweichung), Integralteile, Schrittzahl

- Wenn nach Ablauf der Schrittzahl eine neue Nachricht da ist, wird die Bewegung ohne Zwischenhalt fortgesetzt.
- Sonst oder wenn die Stopptaste betätigt wird, bricht die Bewegung ab und starte mit der nächsten Nachricht neu.
- Nach Abarbeitung einer Teilbewegung werden die Differenzen und Intgralteile an den PC gesendet.
- Ist-Positionen und -Geschwindigkeiten für die graphische Darstellung ergeben sich aus den Sollwerten und Differenzen.
- Für eine flüssige Bewegung ohne »Stopp« muss der PC die Folgenachricht vor der Anwort auf die aktuelle Nachricht absenden.

Ausgabe auf dem LCD-Monitor:

- aktuelle Soll/Ist-Abweichungen, Integralanteile,
- Zähler für Bewegungsstopps,
- Automatenzustand und
- Fehlerzähler (Empfangs-Timeout, Sendeversagen, Winkelmessfehler, »falsche Interrupts«).

Konstanten zur Definition der LCD-Ausgabe


```
#define INITSTR "A:xxx xxx T:xx .B:xxx xxx E:...."
#define LCP_DIFF_R   2   //Differenz Motor R
#define LCP_INTEG_R  6   //Integralteil Motor R
#define LCP_STPCT   12   //Zähler "Bewegungsstopps"
#define LCP_STATE   15   //Zustand des Testprogramms
#define LCP_DIFF_L  18   //Differenz Motor L
#define LCP_INTEG_L 22   //Integralteil Motor L
#define LCP_ERR     28   //Beginn Fehlerzähler
#define ERR_SEND    28   //Zähler Sendversagen
#define ERR_ETO     29   //Zähler Empfangs-Timeout
#define ERR_WMESS   30   //FZ Winkelmessung
//Zeichen 31 ist der Zähler für falsche Interrupts
```

Hardware-Konfiguration und Treiber

Hardware-Konfiguration:

- Taster- oder Schalter-Modul an JA (Not-Aus)
- LC-Display an JD (LCD Monitor), JDX »gekreuzt (=)«.
- H-Brücken mit Motoren an JL, JLX »gekreuzt (=)«.

- PModUSBUSART an JH und PC, JHX »gekreuzt (=)«.

Treiber:

```
#include "pwm.h"       //Motorsteuerung ueber PWM
#include "rotmess.h"   //Messung der Rotationsschritte
#include "comir_pc.h" //Kommunikation mit dem PC
#include "comir_lcd.h"//LCD-Kontrollmonitor
#include <stdlib.h>    //daraus wird abs(...) genutzt
```

Einstellung im Headern »rotmess.h« 20 ms Messzeit:

```
#define ABTASTSCHRITTE 40   //40*0,5ms = 20ms
```

Einstellung im Header »comir_pc.h« :

```
#define COM_PC_RMSG_LEN 10  //Anzahl Empfangsbytes
#define COM_PC_SMSG_LEN  8  //Anzahl Sendebytes
```

Private globale Daten:

```
uint8_t stop_ct=1;          //Zähler Bewegungsstopps
uint16_t step_ct;           //Schrittzähler
uint16_t kp, ki;            //Reglerkoeffizienten
int32_t diffR, diffL;       //Schrittdifferenz
int32_t integR, integL;     //Integralanteile
int32_t pwmR, pwmL;         //Stellwerte
int16_t speedR, speedL;     //Soll-Geschwindigkeit
uint8_t mrmsg[COM_PC_RMSG_LEN]; //Empfangsnachricht
uint8_t msmsg[COM_PC_SMSG_LEN]; //Sendenachricht
```

(Re-)Initialisierungsfunktion der reglerinternen Größen:

```
void regelung_reset(){//Regelung initialisieren
 diffR = 0; diffL = 0;//Regelungsabweichungen löschen
 integR= 0;integL = 0;//Integralanteile löschen
}
```

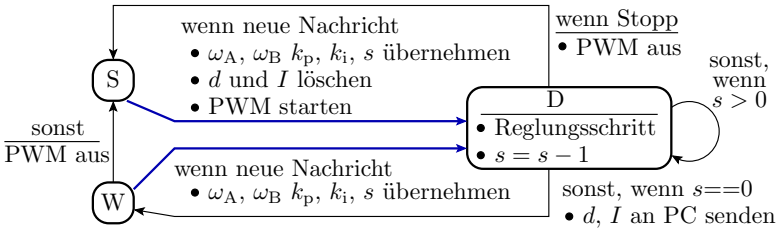
Programmrahmen mit Initialisierung:

```
int main(){
 uint8_t state = 'S';  //Anfangszustand gestoppt
 com_pc_init();        //Treiber initialisieren
 rotmess_init();
 pwm_init();
 lcd_init((uint8_t*)INITSTR);
 DDRA = 0;             //für Tastereingabe
 sei();                //Interrupts ein
 while(1){ ... }       //Endlosschleife
}
```

Übernahme Eingabenachricht (Programmbaustein, 2x genutzt):

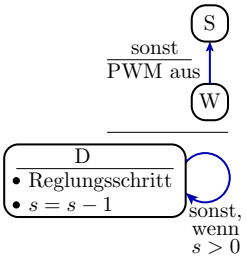
```
  speedR = ((int16_t)mrmsg[0] <<8) + mrmsg[1];
  speedL = ((int16_t)mrmsg[2] <<8) + mrmsg[3];
  step_ct= ((uint16_t)mrmsg[4]<<8) + mrmsg[5];
  kp =     ((uint16_t)mrmsg[6]<<8) + mrmsg[7];
  ki =     ((uint16_t)mrmsg[8]<<8) + mrmsg[9];
```

Ablauf in der Endlosschleife:

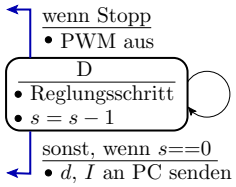



```
lcd_disp_chr(state, LCP_STATE);//Zustand anzeigen
if (state=='W'||state=='S'){
 if (com_pc_get(mrmsg)){   //wenn neue Nachricht
  ...                      //übernehmen
  if (state=='S'){
   regelung_reset();
   pwm_start();
  }
  state = 'D';             //Folgezustand "Bewegung"
 }
```

```
//wenn im Zustand Weiter und noch keine neue Eingabe
else if (state == 'W'){
 pwm_stop();     //Motoren anhalten
 state = 'S';    //Zustand "Stop"
 lcd_disp_val(++stop_ct, LCP_STPCT, 2);
}
```




```
if ((step_ct>0) && state=='D'){
 int16_t sa, sb;
 if (rotmess_get(&sa, &sb)){
  if (rotmess_err())        //bei Fehler in "rotmess"
   lcd_incErr(ERR_WMESS);   //Fehlerzähler erhöhen
  ... //Ausführung eines PID-Regelschritts je Motor
  step_ct--;
  lcd_disp_val(abs(diffR)>>8, LCP_DIFF_R, 3);
  lcd_disp_val(abs(integR)>>8, LCP_INTEG_R, 3);
  lcd_disp_val(abs(diffL)>>8, LCP_DIFF_L, 3);
  lcd_disp_val(abs(integL)>>8, LCP_INTEG_L, 3);
 }
}
```




```
if ((step_ct==0) && state=='D'){
  //bei Bewegungsende
  //Daten zum PC
  msmsg[0] =  diffR>>16;
  msmsg[1] =  diffR>>8 & 0xFF;
  msmsg[2] = integR>>16; msmsg[3] = integR>>8 & 0xFF;
  msmsg[4] =  diffL>>16; msmsg[5] =  diffL>>8 & 0xFF;
  msmsg[6] = integL>>16; msmsg[7] = integL>>8 & 0xFF;
  if (!com_pc_send(msmsg))  //wenn Senden versagt
    lcd_incErr(ERR_SEND);   //Fehlerzähler erhöhen
  state='W';                //Zustand => "Weiter"
}
if (PINA){            //bei Tastendruck an Port A
  pwm_stop();         //Motoren anhalten
  state = 'S';        //Anfangszustand herstellen
  lcd_disp_str((uint8_t*)"xx", LCP_STPCT, 2);
  lcd_disp_str((uint8_t*)"....", LCP_ERR, 4);
  stop_ct = 0;
}
```

3.4 Python-Steuerprogramm

Regelungstest mit Python

Genutzte Module:

```
import serial         #serielle Schnittstelle
import matplotlib.pyplot as plt #Plotfunktion
from sys import exit #Fkt. für Programmabbruch
```

Grundeinstellung für die Regelung und Kommunikation:

```
kp = 1000; ki=500;    # Regelungskoeffizienten
ts = 5                # Regelschr. je Nachricht
```

- Bei ts = 5 wird alle 100 ms vom Fahrzeug eine Nachricht erwartet und eine zurückgeschickt.
- Wenn der PC zu langsam ist, stoppt die PWM.
- Wenn der Stoppzähler auf dem LCD der Zähler mehr als eins pro Bewegung hochzählt bzw. die Regelung ruckt, ts hochsetzen[3].

Beschreibung der Bewegung als Liste von Tupeln aus Sollgeschwindigkeit und Zeitdauer:

```
#Bewegungsablauftupel(speed, count)
#speed: Sollgeschwindigkeit in WS*256 je 20ms
#count Anzahl von Schritten der Dauer ts*20ms
trajList = [(2000, 20), (1000, 20), (-3000, 20)]
```

[3] Für Messungen im kürzeren Zeitabstand ist das Programm so umzuschreiben, dass die Zeittoleranzen und Datenpakete größer sind.

Funktion zur Erzeugung einer Sendenachricht:

```
def smsg(v):    # Sendenachricht erzeugen
  bv = v.to_bytes(2, byteorder='big', signed=True)
  bts = ts.to_bytes(2, byteorder='big')
  bkp = kp.to_bytes(2, byteorder='big')
  bki = ki.to_bytes(2, byteorder='big')
  return bv + bv + bts + bkp + bki
```

(Vereinbarung ts, ki und kp siehe Folie zuvor).

Serielle Schnittstelle öffnen. COM anpassen. Timeout so setzen, dass Leseoperationen nach etwa der doppelten Zeit, in der der µP geantwortet haben muss, mit weniger gelesenen Bytes abbrechen:

```
serial.Serial("COM9", timeout=ts*0.04)
```

Anfangspunkt graphische Ausgabe, Tabellenkopf Textausgabe:

```
t =[0]; dA=[0]; dB=[0]; s=[0]
print('␣t␣|speed|diff_R|intg_R|diff_L|intg_L|')
```

Damit der µP nach Abschluss jeder Teilbewegung die nächste Nachricht hat, müssen zum Bewegungsbeginn vor dem Warten auf die erste Antwort zwei Nachrichten gesendet werden:

```
ser.write(smsg(trajList[0][0]))
```

Wiederhole für jedes Tupel der Trajektorliste »count« mal:

```
for (speed, count) in trajList:
 for idx in range(count):
  ser.write(smsg(speed)) #Nachricht senden
  rmsg = ser.read(8)     #auf 8 Antwortbytes warten
  if len(rmsg)<8:        #werden weniger empfangen[4]
   ser.close();          #Schnittstelle schliessen
   exit()                #Script beenden
```

Sonst die 8 Bytes aufspalten. Zeit, Sollgeschwindigkeit, ... tabellarisch ausgeben und für graphische Ausgabe an Listen hängen:

```
diff_R=int.from_bytes(rmsg[0:2],byteorder='big',signed=True)
intg_R=int.from_bytes(rmsg[2:4],byteorder='big',signed=True)
diff_L=int.from_bytes(rmsg[4:6],byteorder='big',signed=True)
intg_L=int.from_bytes(rmsg[6:8],byteorder='big',signed=True)
print('%3.1f|'%t[-1] + '%5i|'%speed, '%5i|'%diff_R,
    '%5i|'%intg_R, '%5i|'%diff_L, '%5i|'%intg_L)
t  += [t[-1]+0.1]
dR += [diff_R]
dL += [diff_L]
s  += [speed*(10.0/256)] #Größenanpassung an Diff. im Bild
```

[4] Das passiert, wenn eine Taste am Versuchsboard gedrückt wird.

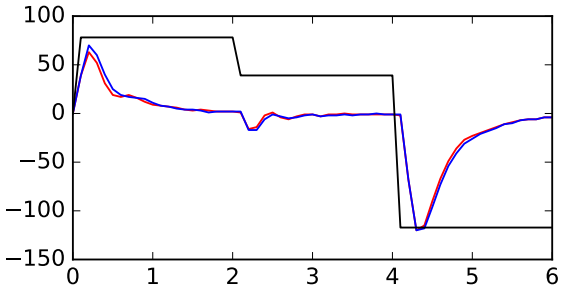
Beispielhaft erzeugte Textausgabe:

```
 t  |speed|diff_R|intg_R|diff_L|intg_L|
0.0|  2000|    38|   224|    39|   228|
0.1|  2000|    67|   798|    68|   806|
0.2|  2000|    55|  1407|    59|  1436|
...
```

Serielle Schnittstelle schließen und Graphik erzeugen:

```
ser.close()
plt.plot(t, dR, 'r', t, dL, 'b', t, s, 'k')
plt.show()
```

Sollposition und Fehler in Winkelschritten in Abhängigkeit von der Bewegungsdauer in Sekunden:



(schwarz – Sollgeschwindigkeit (skaliert); rot / blau – Positionsabweichung Motor R / L in Winkelschritten).

Ergebnisdiskussion

```
kp = 1000; ki=500;          #Regelungskoeffizienten
ts = 5                      #Regelschritte je Nachricht
trajList = [(2000, 20), (1000, 20), (-3000, 20)]
```

Bei jedem Geschwindigkeitssprung schwingt die Soll/Ist-Abweichung, bevor sie gegen null strebt. Das Schwingen lässt sich unterbinden,

- indem die Soll-Geschwindigkeit in kleinen Schritten oder stetig geändert wird.
- Durch bessere Wahl von k_p und k_i.

4 Aufgaben

Aufgabe 12.1: Ausprobieren der Motorregelung

- Taster- oder Schalter-Modul an JA (Not-Aus)
- LC-Display an JD (LCD Monitor), JDX »gekreuzt (=)«.

- H-Brücken mit Motoren an JL, JLX »gekreuzt (=)«.
- PModUSBUSART an JH und PC, JHX »gekreuzt (=)«.
- Projekt »F12-regelung\regelung« öffnen, übersetzen, starten.
- Konsole (cmd) öffnen.
- Wechsel in das Programmverzeichnis »...\P12\Python«.
- »regelung.py« + Enter.

Experimentelle Bestimmung von k_p und k_i:

- $k_\mathrm{i} = 0$ setzen k_p so lange erhöhen, bis die Regelung schwingt. Davon auf 60% reduzieren.
- k_i soweit erhöhen, dass die Regelung schwingt und davon auch auf 60% reduzieren.

Aufgabe 12.2: Aufgabe Abschlussprojekt

Sie haben

- eine Einführung in C und Python mit Beispielen bekommen,
- die Hardware in ihrer Kiste kennen gelernt.
- Für die Hardware wurden Treiber durchgesprochen, ...
- Es folgen morgen als letzte HW-Bausteine: Joystick, IR-Abstandssensor und Bodensensor für Linienverfolgung.

Überlegen Sie sich eine Aufgabe für ihr Abschlussprojekt, die mit der Hardware und ihren Kenntnissen als Team-Arbeit realisierbar ist.

- Besprechung der Realisierbarkeit mit dem Betreuer[5].
- Abschluss des Programmierprojekts bis zur letzten Vorlesungswoche.
- Abschlusspräsentation mit BBB-Vortrag, und Vorführung am letzten Veranstaltungstermin.

[5] Die Suche einer vernünftigen und realisierbaren Zielstellung ist anspruchsvoll. Oft werden Zielstellungen am Anfang unrealistisch groß gewählt ...