1 Drehzahlregelung

1.1 Drehzahlregelung: Michael Wörner (30137), Christian Silfang (30147)

1.2 Aufgabenstellung und -ziel

Entwickelt werden soll eine Drehzahlregelung mit Hilfe von Interrupts. Dazu sollen drei verschiedene Aufgaben gelöst werden.

1. Der Motor soll so geregelt werden, dass die Drehzahl möglichst konstant bleibt
2. Es sollen vier Positionen im 90° Winkel angefahren werden
3. Es soll eine serielle Ausgabe der Drehrichtung und der Drehzahl, bei Fremdantrieb, erfolgen

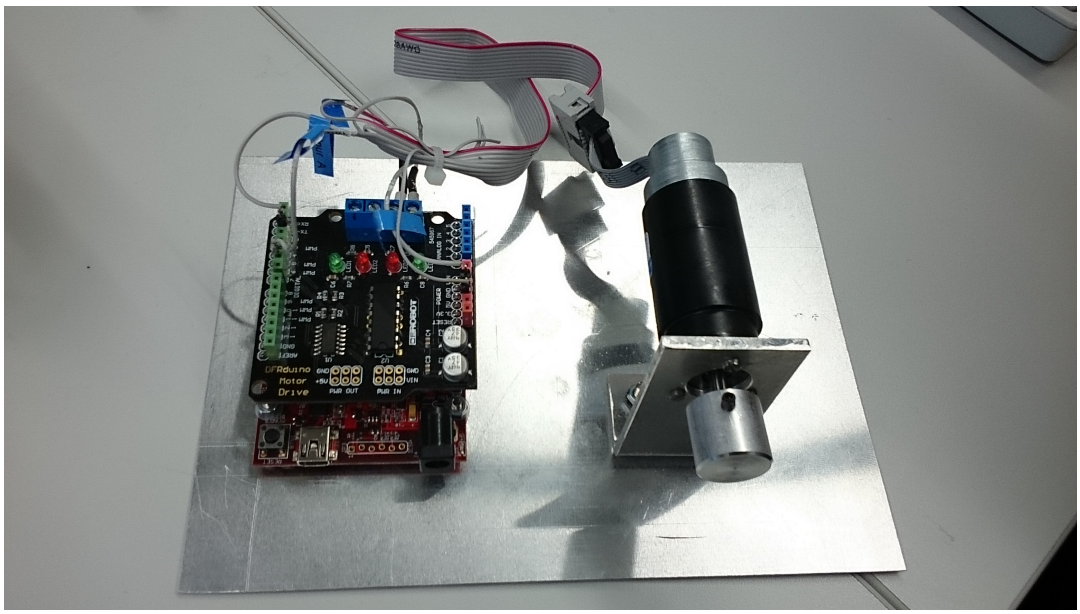

Abbildung 1.1: „Beweisfoto“ des Versuchs Drehzahlregelung

1.3 Versuchsaufbau

Der Aufbau des Versuchs ist in in Abb. 1.1 zu sehen.

Die Abb. 1.2 zeigt den aufgesetzten Motor-Shield der Arduino Plattform. dieser wird zur Ansteuerung des Motors benötigt, da dieser das PWM-Signal für den Motor in die entsprechende Drehzahl umwandelt.

Die Tab. 1.1 zeigt die nötigen Belegungen des Shields.

D4	Motor 1 direction control
D5	Motor 1 PWM control
D6	Motor 2 PWM control
D7	Motor 2 direction control

Tabelle 1.1: Belegung des Arduino-Motorshields

Im Folgenden soll nun auf die konkrete Umsetzung eingegangen werden.

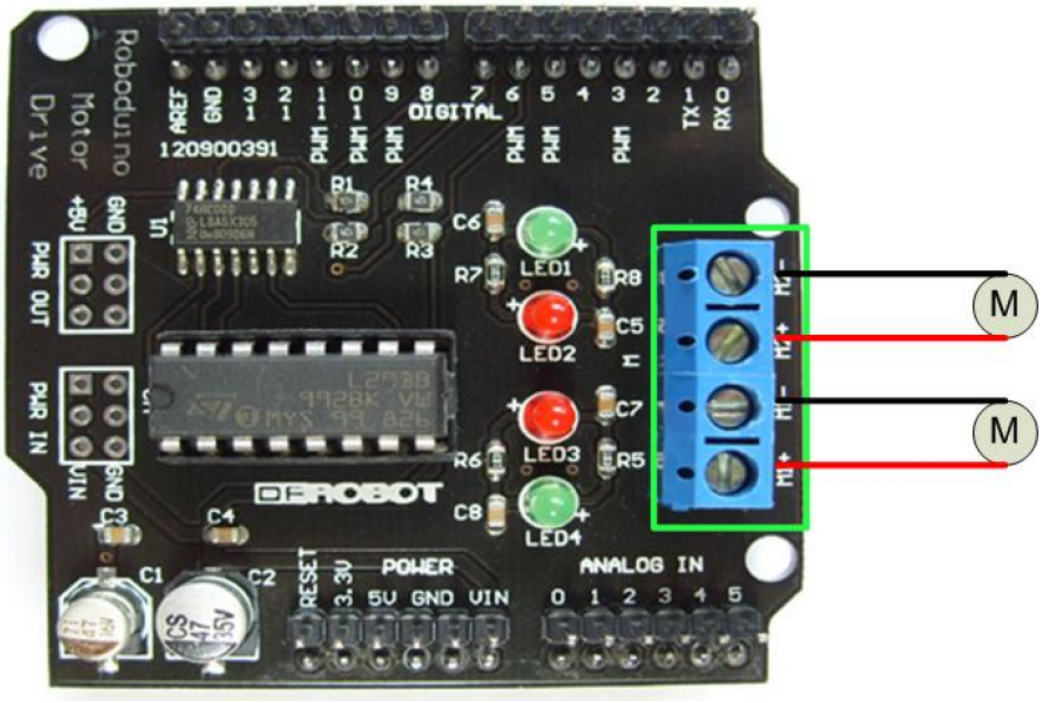

Abbildung 1.2: Arduino Motor-Shield

1.4 Modellierung

1.4.1 Messen der Drehzahl und der Position des Rotors

Damit die Drehzahl geregelt werden kann, muss diese gemessen werden. Dies geschieht auf folgende Art und Weise. Die Impulsgeber des Motors werden an zwei Anschlüsse des Mikrocontroller angeschlossen. Einer davon löst bei einer steigenden Flanke einen Interrupt aus. In der dazugehörigen Interrupt Service Routine wird überprüft ob der Pin, an dem der andere Impulsgeber angeschlossen ist, High oder Low ist. Dadurch kann bestimmt werden ob der Motor sich links- oder rechtsherum dreht. Abhängig davon wie die Drehrichtung ist, werden zwei Variablen inkrementiert oder dekrementiert.

In einem Interrupt der von einem Timer generiert wird, wird die Anzahl der Erhöhungen bzw. Erniedrigungen der Variable überprüft. Durch die Überprüfung der Impulse pro

Zeiteinheit wird die Drehzahl bestimmt. Nachdem die Überprüfung stattgefunden hat wird einer der Variablen wieder auf 0 zurückgesetzt. In der zweiten Variable ist dann die absolute Position des Rotors abgespeichert.

1.4.2 Setzen der Drehzahl und der Drehrichtung

Die Drehzahl des Motors wird mittels eines PWM-Signals eingestellt. Um dieses zu generieren wird ein Timer Interrupt verwendet. Bei jedem Aufrufen der Interrupt Service Routine wird eine Variable inkrementiert. Anschließend wird der Wert dieser Variable mit einem anderen Wert verglichen. Ist der Wert größer, wird ein digitaler Ausgang des Controllers auf High gesetzt. Ist er kleiner, wird dieser Ausgang auf Low gesetzt. Durch das Vergrößern bzw. Verkleinern des Vergleichswertes wird die Pulsbreite des PWM-Signals vergrößert bzw. verkleinert.

Um die Drehrichtung einzustellen, steht ein Digitaler Eingang am Motor zur Verfügung. Wird an diesem ein Low-Pegel angelegt dreht der Motor sich im Uhrzeigersinn. Wird ein High-Pegel angelegt dreht er sich im Gegenuhrzeigersinn.

1.4.3 Drehzahlregelung

Um die Drehzahl des Motors zu regeln wird in regelmäßigen Abständen der Istwert mit dem Sollwert verglichen. Ist der Istwert zu klein, wird die Drehzahl um einen bestimmten Wert erhöht. Ist der Istwert zu groß wird die Drehzahl um einen gewissen Wert erniedrigt.

1.4.4 Positionsregelung

Ein Teil der Aufgabe ist es, mit dem Motor eine bestimmte Position anzufahren. Um dies zu realisieren, wird ein abgewandelter Proportional Regler implementiert. Die Stellgröße des Positionsreglers ist der Sollwert der Drehzahlregelung.

Der Regler wird folgendermaßen entworfen. Ist die Abweichung sehr groß, wird die Geschwindigkeit des Motors auf den maximalen Wert eingestellt. Für eine mittlere Abweichung der Position, wird die Drehzahl durch eine lineare Funktion der Abweichung abgebildet

$$\mathrm{rpm}(x) = m \cdot x \tag{1.1}$$

Ist die Abweichung nahe 0 wird der Sollwert, für die Drehzahlregelung auf 0 gestellt.

Die Werte für die maximale Geschwindigkeit, die Steigung m aus der Gleichung 1.1 sowie die Abweichung, ab der die Drehzahl auf 0 gesetzt wird, werden experimentell ermittelt.

1.4.5 User-Interface

Damit für die einzelnen Teilaufgaben überprüft werden können, wird ein User-Interface implementiert. Die Interaktion mit dem Benutzer findet über die serielle Schnittstelle statt.

Der Benutzer hat folgende Möglichkeiten

- Motor soll von Hand angetrieben werden. In diesem Fall wird die Positions- und die Drehzahlregelung mittels eines Flags außer Kraft gesetzt. Die Messung der Drehzahl bleibt aber aktiv. Alle 0,5s wird die gemessene Drehzahl über die serielle Schnittstelle ausgegeben. Ist der Wert negativ, dreht wird der Motor im Gegenuhrzeigersinn bewegt.

- Der Motor soll einen 90°-, einen 180°-, einen 270°- und einen 360°-Winkel anfahren. In diesem Fall bleiben die Drehzahl- und die Positionsregelung aktiv. Zu beginn wird die Absolute Position auf 0 gesetzt. Anschließend wird die Sollposition auf 1350 gesetzt. Dies entspricht einer Drehung um 90°. Nachdem die Position eingestellt wurde, wird mittels der Arduino-Funktion **delay()** für 5 Sekunden gewartet. Anschließend wird die Sollposition um 1350 erhöht. Dieser Vorgang wird wiederholt bis die Position 5400 angefahren wurde.

- Die Drehzahl des Motors soll konstant gehalten werden. In diesem Fall wird die Positionsregelung mittels eines Flags auser kraft gesetzt. Die Drehzahlregelung bleibt aktiv. Hierbei stehen drei geschwindigkeiten - fast, medium und slow - dem Benutzer zur auswahl. Alle 0,5s wird die Soll- und die Istfrequenz der Impulsgeber ausgegeben.

1.5 Implementierung

Dieser Abschnitt geht detailiert auf die Implementierung des Versuchs ein. Dazu wurden auch die Listings zur Demonstration des Quellcodes abgebildet.

1.5.1 Setup des Boards

Das Listing 1.1 zeigt das Setup des Boards zur Drehzahlregelung. Die Belegung der Pins (Zeile 2-5) ist ebenfalls in Tabelle 1.2 abgebildet.

```
1 /* Definitionen */
2 #define PWM_OUT  5
3 #define DIR_OUT  4
4 #define SPUR_A   9
5 #define SPUR_B   2
```

PWM OUT	5	OUTPUT
DIR OUT	4	OUTPUT
SPUR A	9	INPUT
SPUR B	2	INPUT

Tabelle 1.2: Pinbelegung des ChipKit Uno32 für die Drehzahlregelung

```
6
7  #define LINKS  true
8  #define RECHTS false
9
10 #define MAX_INT          INT16_MAX
11 #define MAX_LONG         INT32_MAX
12 #define MAX_I_TERM       (MAX_LONG / 2)
13
14
15 /* Globale Variablen */
16 uint16_t pwm;
17 int16_t impuls_cnt;
18 int16_t positionIst;
19 int16_t positionSoll;
20 int16_t fImpulsIst;
21 int16_t fImpulsSoll;
22 bool b_position;
23 bool b_impuls;
24 int16_t task_var;
25
26
27 /* Funktionsprototypen */
28 uint32_t pwm_callback(uint32_t current_time);
```

```
29 void set_rpm(int16_t rpm);
30 uint32_t getRpm_callback(uint32_t currentTime);
31 void spur_interrupt(void);
32 uint32_t position_callback(uint32_t currentTime);
33 void constant_drive(uint8_t swi);
34
35
36
37 /* Setup Funktion */
38 void setup(void)
39 {
40     /* Variablen initialisieren */
41     pwm = 0;
42     impuls_cnt = 0;
43     positionIst = 0;
44     positionSoll = 0;
45     fImpulsIst = 0;
46     fImpulsSoll = 0;
47     b_position = false;
48     b_impuls = false;
49
50     /* Peripherie Konfigurieren */
51     pinMode(PWM_OUT,OUTPUT);
52     pinMode(DIR_OUT,OUTPUT);
53     pinMode(SPUR_A, INPUT);
54     pinMode(SPUR_B, INPUT);
55     Serial.begin(9600);
56
57     /* Tasks erzeugen */
58     createTask(position_callback, 100, TASK_ENABLE, &task_var);
59
```

```
60      /* Interrupts einstellen */
61      attachCoreTimerService(pwm_callback);
62      attachCoreTimerService(getRpm_callback);
63      attachInterrupt(1, spur_interrupt, RISING);
64
65      delay(1000);
66      Serial.println("press␣m␣for␣menue");
67  }
```

Listing 1.1: Setup des Boards für die Drehzahlregelung

Die verwendeten Variablen haben dabei folgenden Bedeutung:

- **pwm**: aktueller Wert der PWM
- **impuls_cnt**: Anzahl der Impulse bei der Drehung des Motors (360°≙5400
- **positionIst**: der Istwert der aktuellen Position
- **positionSoll**: der Sollwert der Position
- **fImpulsIst**: der Istwert der Drehzahl in Hertz
- **fImpulsSoll**: der Sollwert der Drehzahl in Hertz
- **b_position**: Flag um Positionsregelung ein- bzw. auszuschalten
- **b_impuls**: Flag um die Drehzahlregelung ein- bzw. auszuschalten
- **task_var**: Variable zum Austausch von Daten in einem Task (wird nicht benötigt)

Zusätzlich wird ein Task zur Positionsregelung angelegt, zwei Interrupt-Service-Routinen und ein Interrupt. Der erste Interrupt-Service wird dabei für die Generierung des PWM-

Signals angelegt der zweite für die Bestimmung der Drehzahl. Der dritte Interrupt ist zur Erkennung eines Impulses des Motors.

1.5.2 User-Interface

Das User-Interface wird in der `main loop()`-Funktion implementiert. Dabei wird auf die Eingabe eines Benutzers gewartet um eine vordefinierte Funktion auszuführen. Das Listing 1.2 zeigt die genaue Implementierung. Über die switch-case Anweisungen kann auf die unterschiedlichen Eingaben reagiert werden.

```
1  void loop(void)
2  {
3      uint8_t u8_recv, temp;
4      uint8_t i;
5
6      if(Serial.available())
7          u8_recv = Serial.read();
8
9
10     switch(u8_recv)
11     {
12         case 'm':   Serial.println("Menu:");
13                     Serial.println("External␣Drive:␣␣␣␣␣␣␣␣␣␣1");
14                     Serial.println("Four␣Positions:␣␣␣␣␣␣␣␣␣␣2");
15                     Serial.println("Constant␣drive␣(slow):␣␣␣3");
16                     Serial.println("Constant␣drive␣(medium):␣4");
17                     Serial.println("Constant␣drive␣(fast):␣␣␣5");
18         break;
19
20         case '1':   Serial.println("External␣Drive");
21                     b_position = false; /* Positionsregelung
```

```
                    abschalten */
22                  b_impuls = false;   /* Drehzahlregelung
                    abschalten */
23
24                  do
25                  {
26                      Serial.print(((float)fImpulsIst)*60.0 /
                        5400.0);
27                      Serial.println("␣rpm");
28                      delay(500);
29                  }while(!Serial.available());
30                  temp = Serial.read();   /* Serial Buffer
                    leeren */
31          break;
32
33          case '2':   Serial.println("Four␣Positions");
34                      positionIst = 0;
35                      positionSoll = 1350;
36                      b_position = true;  /* Positionsregelung
                        aktivieren */
37                      b_impuls = true;    /* Drehzahlregelung
                        aktivieren */
38                      delay(5000);
39
40                      for(i=0; i<3; i++)
41                      {
42                          positionSoll += 1350;   /* Erhoeung um 90
                            Grad */
43                          delay(5000);
44                          Serial.print("positionSoll:␣");
45                          Serial.print(positionSoll);
```

```
46                          Serial.print("␣␣␣positionIst:␣");
47                          Serial.println(positionIst);
48                      }
49          break;
50
51          case '3':   Serial.println("Constant␣Drive␣(slow)");
52                      constant_drive(0);
53          break;
54
55          case '4':   Serial.println("Constant␣Drive␣(medium)");
56                      constant_drive(1);
57          break;
58
59          case '5':   Serial.println("Constant␣Drive␣(fast)");
60                      constant_drive(2);
61          break;
62
63          default:
64          break;
65      }
66  }
```

Listing 1.2: Das User-Interface - main loop()

1.5.3 Messung und Regelung der Drehzahl und Messung der Position des Rotors

Die Listings 1.3 und 1.7 zeigen die konkrete Implementierung der Funktionalität wie sie bereits in Abschnitt 1.4.1 beschrieben wurde.

```
1 uint32_t getRpm_callback(uint32_t currentTime)
```

```
{
    int16_t i16_i;
    static int16_t i16_temp = 0;

    /* Messen der Frequenz der Impulse von Spur B */
    /* Es wird die Anzahl der Impuls pro 0,01s gezaehlt */
    /* Dies wird um 100 Multipliziert um die Frequenz in Hz zu
       erhalten */
    fImpulsIst = impuls_cnt * 100; /* In Hz */
    impuls_cnt = 0;

    if(b_impuls)
    {
        /* Einfache Regelung der Drehzahl */
        /* Ist der Wert der soll Frequenz zu klein wird die
           Pulsbreite erhoeht */
        /* Ist der Wert zu gross wird die Pulsbreite verringert
           */
        if(fImpulsIst < fImpulsSoll)
            i16_temp += 10;

        if(fImpulsIst > fImpulsSoll)
            i16_temp -= 10;

        /* Ueberpruefen auf Ueber- oder Unterlauf der Variable */
        i16_i = 0x07ff;
        if( i16_temp > i16_i )
            i16_temp = i16_i;
        else if( i16_temp < -i16_i )
            i16_temp = -i16_i;

```

```
30          /* Drehzahl setzten */
31          set_rpm(i16_temp);
32      }
33      return (currentTime + 400000);
34  }
```

Listing 1.3: Messung der Drehzahl

```
1   void position_callback(int a, void * b)
2   {
3       int16_t i16_posDiff;
4       if(b_position)
5       {
6           i16_posDiff = positionSoll - positionIst;
7
8           if( (i16_posDiff < 5) && (i16_posDiff > -5) )
9               fImpulsSoll = 0;
10          else if( (i16_posDiff < 1400) && (i16_posDiff > -1400) )
11              fImpulsSoll = 11 * i16_posDiff / 7;
12          else if( i16_posDiff >= 1400 )
13              fImpulsSoll = 2200;
14          else
15              fImpulsSoll = -2200;
16      }
17  }
```

Listing 1.4: Messung der Position des Rotors

1.5.4 Setzen der Drehzahl

Die Listings 1.5 und 1.6 wurden bereits in Abschnitt 1.4.2 und 1.4.3 näher beschrieben.

```
uint32_t pwm_callback(uint32_t current_time)
{
    static uint16_t cnt = 0;
    uint32_t u32_return;

    if( cnt >= pwm )
    {
        digitalWrite(PWM_OUT, LOW);
        flags |= (1<<0);
    }
    else
    {
        digitalWrite(PWM_OUT, HIGH);
        flags |= (1<<1);
    }

    cnt ++;
    cnt &= 0x07ff;

    u32_return = current_time + 200;

    return u32_return;
}
```

Listing 1.5: Berechung des PWM-Signals

```
void set_rpm(int16_t rpm)
{
    uint16_t u16_temp;

    pwm = abs(rpm) & 0x07ff;

```

```
7       if(rpm >= 0)
8           digitalWrite(DIR_OUT,RECHTS);
9       else
10          digitalWrite(DIR_OUT, LINKS);
11  }
```

Listing 1.6: Setzen der Drehzahl

1.5.5 Positionsregelung

Das Listing 1.7 wurde bereits in Abschnitt 1.4.4 detailiert beschrieben.

```
1   void position_callback(int a, void * b)
2   {
3       int16_t i16_posDiff;
4       if(b_position)
5       {
6           i16_posDiff = positionSoll - positionIst;
7
8           if( (i16_posDiff < 5) && (i16_posDiff > -5) )
9               fImpulsSoll = 0;
10          else if( (i16_posDiff < 1400) && (i16_posDiff > -1400) )
11              fImpulsSoll = 11 * i16_posDiff / 7;
12          else if( i16_posDiff >= 1400 )
13              fImpulsSoll = 2200;
14          else
15              fImpulsSoll = -2200;
16      }
17  }
```

Listing 1.7: Positionsregelung

1.5.6 Hilfsfunktionen

In den Listings 1.8 und 1.9 sind Funktionen dargestellt die im Programm der Drehzahlregelung als Hilfsfunktionen benötigt werden.

Die Funktion `spur_interrupt()` wird als Interrupt Service Routine verwendet. Sie wird aufgerufen, wenn einer der Impulsgeber einen Interrupt auslöst.

```
1  void spur_interrupt(void)
2  {
3      /* Bestimmen der Drehrichtung, Anazhl der Impulse und die
           Absolute Positions */
4      if( digitalRead(SPUR_A) )
5      {
6          impuls_cnt ++;
7          positionIst ++;      /* Absolute Position erhoehen */
8      }
9      else
10     {
11         impuls_cnt --;
12         positionIst --;      /* Absolute Position erniedirigen */
13     }
14 }
```

Listing 1.8: Funktion die bei externem Interrupt ausgelöst wird

Die Funktion `constant_drive()` wird für die einfachere Bedienung des Programms bzw. zur Unterstützung des User-Interfaces benötigt und funktioniert wie bereits in Abschnitt 1.4.5 beschrieben wurde.

```
1  void constant_drive(uint8_t swi)
2  {
3      uint8_t temp;
```

```
4     if(swi < 3)
5     {
6         fImpulsSoll = 1000 + swi * 500;
7         b_position = false; /* Positionsregelung abschalten */
8         b_impuls = true;    /* Drehzahlregelung anschalten */
9         do
10        {
11            Serial.print("fImpulsSoll␣");
12            Serial.println(fImpulsSoll);
13            Serial.print("fImpulsIst␣");
14            Serial.println(fImpulsIst);
15            delay(500);
16        }while(!Serial.available());    /* So lange widerholen
              bis Benutzer eine Eingabe taetigt */
17        fImpulsSoll = 0;
18        temp = Serial.read();
19    }
20    else
21        fImpulsSoll = 0;
22    delay(3000);    /* Warten bis auf 0 Hz geregelt wurde */
23 }
```

Listing 1.9: Hilfsfunktion für das User-Interface

1.6 Quellcode

```
1 #include <Arduino.h>
2 #include <stdlib.h>
3
4
```

```
5  /* Definitionen */
6  #define PWM_OUT  5
7  #define DIR_OUT  4
8  #define SPUR_A  9
9  #define SPUR_B  2
10
11 #define LINKS  true
12 #define RECHTS false
13
14 #define MAX_INT          INT16_MAX
15 #define MAX_LONG         INT32_MAX
16 #define MAX_I_TERM       (MAX_LONG / 2)
17
18
19 /* Globale Variablen */
20 uint16_t pwm;
21 int16_t impuls_cnt;
22 int16_t positionIst;
23 int16_t positionSoll;
24 int16_t fImpulsIst;
25 int16_t fImpulsSoll;
26 bool b_position;
27 bool b_impuls;
28 int16_t task_var;
29
30
31 /* Funktionsprototypen */
32 uint32_t pwm_callback(uint32_t current_time);
33 void set_rpm(int16_t rpm);
34 uint32_t getRpm_callback(uint32_t currentTime);
35 void spur_interrupt(void);
```

```
36 uint32_t position_callback(uint32_t currentTime);
37 void constant_drive(uint8_t swi);
38
39
40
41 /* Setup Funktion */
42 void setup(void)
43 {
44     /* Variablen initalisieren */
45     pwm = 0;
46     impuls_cnt = 0;
47     positionIst = 0;
48     positionSoll = 0;
49     fImpulsIst = 0;
50     fImpulsSoll = 0;
51     b_position = false;
52     b_impuls = false;
53
54     /* Peripherie Konfigurieren */
55     pinMode(PWM_OUT,OUTPUT);
56     pinMode(DIR_OUT,OUTPUT);
57     pinMode(SPUR_A, INPUT);
58     pinMode(SPUR_B, INPUT);
59     Serial.begin(9600);
60
61     /* Tasks erzeugen */
62     createTask(position_callback, 100, TASK_ENABLE, &task_var);
63
64     /* Interrupts einstellen */
65     attachCoreTimerService(pwm_callback);
66     attachCoreTimerService(getRpm_callback);
```

```
67      attachInterrupt(1, spur_interrupt, RISING);
68
69      delay(1000);
70      Serial.println("press␣m␣for␣menue");
71  }
72
73  /* Main Funktion */
74  void loop(void)
75  {
76      uint8_t u8_recv, temp;
77      uint8_t i;
78
79      if(Serial.available())
80          u8_recv = Serial.read();
81
82
83      switch(u8_recv)
84      {
85          case 'm':   Serial.println("Menu:");
86                      Serial.println("External␣Drive:␣␣␣␣␣␣␣␣␣␣1");
87                      Serial.println("Four␣Positions:␣␣␣␣␣␣␣␣␣␣2");
88                      Serial.println("Constant␣drive␣(slow):␣␣␣3");
89                      Serial.println("Constant␣drive␣(medium):␣4");
90                      Serial.println("Constant␣drive␣(fast):␣␣␣5");
91          break;
92
93          case '1':   Serial.println("External␣Drive");
94                      b_position = false; /* Positionsregelung
                           abschalten */
95                      b_impuls = false;   /* Drehzahlregelung
                           abschalten */
```

```
96
97                      do
98                      {
99                          Serial.print(((float)fImpulsIst)*60.0 /
                               5400.0);
100                         Serial.println("␣rpm");
101                         delay(500);
102                     }while(!Serial.available());
103                     temp = Serial.read();   /* Serial Buffer
                           leeren */
104
105         break;
106
107         case '2':   Serial.println("Four␣Positions");
108                     positionIst = 0;
109                     positionSoll = 1350;
110                     b_position = true;  /* Positionsregelung
                           aktivieren */
111                     b_impuls = true;    /* Drehzahlregelung
                           aktivieren */
112                     delay(5000);
113
114                     for(i=0; i<3; i++)
115                     {
116                         positionSoll += 1350;   /* Erhoeung um 90
                               Grad */
117                         delay(5000);
118                         Serial.print("positionSoll:␣");
119                         Serial.print(positionSoll);
120                         Serial.print("␣␣␣positionIst:␣");
121                         Serial.println(positionIst);
```

```
122
123                     }
124
125         break;
126
127         case '3':   Serial.println("Constant Drive (slow)");
128                     constant_drive(0);
129
130         break;
131
132         case '4':   Serial.println("Constant Drive (medium)");
133                     constant_drive(1);
134         break;
135
136         case '5':   Serial.println("Constant Drive (fast)");
137                     constant_drive(2);
138
139         break;
140
141         default:
142         break;
143
144     }
145
146
147 }
148
149
150 /* Funktion um das PWM-Signal zu generieren */
151 /* Diese Funktion wird als CoreTimer Callback Funktion verwendet
    */
```

```
152 uint32_t pwm_callback(uint32_t current_time)
153 {
154     static uint16_t cnt = 0;
155     uint32_t u32_return;
156 
157     if( cnt >= pwm )
158     {
159         digitalWrite(PWM_OUT, LOW);
160     }
161     else
162     {
163         digitalWrite(PWM_OUT, HIGH);
164     }
165 
166     cnt ++;
167     cnt &= 0x07ff;
168 
169     u32_return = current_time + 200;
170 
171     return u32_return;
172 }
173 
174 
175 /* Funktion um die Drehzahl des Motors einzustellen */
176 void set_rpm(int16_t rpm)
177 {
178     uint16_t u16_temp;
179 
180     pwm = abs(rpm) & 0x07ff;
181 
182     if(rpm >= 0)
```

```
183         digitalWrite(DIR_OUT,RECHTS);
184     else
185         digitalWrite(DIR_OUT, LINKS);
186 }
187
188 /* Funktion um die Frequenz der Impulsgeber zu ermitteln */
189 uint32_t getRpm_callback(uint32_t currentTime)
190 {
191     int16_t i16_i;
192     static int16_t i16_temp = 0;
193
194     /* Messen der Frequenz der Impulse von Spur B */
195     /* Es wird die Anzahl der Impuls pro 0,01s gezaehlt */
196     /* Dies wird um 100 Multipliziert um die Frequenz in Hz zu
           erhalten */
197     fImpulsIst = impuls_cnt * 100; /* In Hz */
198     impuls_cnt = 0;
199
200     if(b_impuls)
201     {
202         /* Einfache Regelung der Drehzahl */
203         /* Ist der Wert der soll Frequenz zu klein wird die
               Pulsbreite erhoeht */
204         /* Ist der Wert zu gross wird die Pulsbreite verringert
               */
205         if(fImpulsIst < fImpulsSoll)
206             i16_temp += 10;
207
208         if(fImpulsIst > fImpulsSoll)
209             i16_temp -= 10;
210
```

```
211         /* Ueberpruefen auf Ueber- oder Unterlauf der Variable */
212         i16_i = 0x07ff;
213         if( i16_temp > i16_i )
214             i16_temp = i16_i;
215         else if( i16_temp < -i16_i )
216             i16_temp = -i16_i;
217 
218         /* Drehzahl setzten */
219         set_rpm(i16_temp);
220     }
221     return (currentTime + 400000);
222 }
223 
224 /* Funktion die bei Impuls-Interrupt aufgerufen wird */
225 void spur_interrupt(void)
226 {
227     /* Bestimmen der Drehrichtung, Anazhl der Impulse und die
            Absolute Positions */
228     if( digitalRead(SPUR_A) )
229     {
230         impuls_cnt ++;
231         positionIst ++;      /* Absolute Position erhoehen */
232     }
233     else
234     {
235         impuls_cnt --;
236         positionIst --;      /* Absolute Position erniedirigen */
237     }
238 }
239 
240 /* Funktion um die Stellung des Motors zu regeln */
```

```
241 /* Diese wird als Callback-Funktion in einem Coretimer verwendet
    */
242 void position_callback(int a, void * b)
243 {
244     int16_t i16_posDiff;
245     if(b_position)
246     {
247         i16_posDiff = positionSoll - positionIst;
248 
249         if( (i16_posDiff < 5) && (i16_posDiff > -5) )
250             fImpulsSoll = 0;
251         else if( (i16_posDiff < 1400) && (i16_posDiff > -1400) )
252             fImpulsSoll = 11 * i16_posDiff / 7;
253         else if( i16_posDiff >= 1400 )
254             fImpulsSoll = 2200;
255         else
256             fImpulsSoll = -2200;
257     }
258 }
259 
260 /* Funktion um eine Konstante Drehzahl einzustellen */
261 /* Der Funktion muss ein Wert zwischen 0 und 2 uebergeben werden
    */
262 /* Dabei steht 0 fuer slow, 1 fuer medium und 2 fuer fast */
263 /* Ist der Uebergabeparameter groesser als 2 wird die Drehzahl
    auf 0 gestellt */
264 void constant_drive(uint8_t swi)
265 {
266     uint8_t temp;
267     if(swi < 3)
268     {
```

```
269         fImpulsSoll = 1000 + swi * 500;
270         b_position = false; /* Positionsregelung abschalten */
271         b_impuls = true;    /* Drehzahlregelung anschalten */
272         do
273         {
274             Serial.print("fImpulsSoll␣");
275             Serial.println(fImpulsSoll);
276             Serial.print("fImpulsIst␣");
277             Serial.println(fImpulsIst);
278             delay(500);
279         }while(!Serial.available());    /* So lange widerholen
                bis Benutzer eine Eingabe taetigt */
280         fImpulsSoll = 0;
281         temp = Serial.read();
282     }
283     else
284         fImpulsSoll = 0;
285     delay(3000);    /* Warten bis auf 0 Hz geregelt wurde */
286 }
```