1 Wippe

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

1.2 Aufgabenstellung und -ziel

Das Ziel dieser Aufgabe ist es eine Kugel die sich auf einer Wippe befindet, so zu stabilisieren, das sie in der Mitte der Wippe bleibt. Zusätzlich soll die Position der Kugel über eine Serielle Schnittstelle gesendet werden.

1.3 Versuchsaufbau

Der Versuch besteht aus der eigentlichen Wippe. An ihr wurden zwei Fohlienpotentiometer zur Positionsbestimmung angebracht. Die Spannung am Ausgang des Potentiometer kann mit der folgenden Gleichung beschrieben werden

$$U_{\mathrm{A/D}}(s) = 3,3V \cdot \frac{1-s}{1+s} \qquad (0 \leq s \leq 1) \tag{1.1}$$

Die dazugehörige kurve ist in Abb. 1.2 zu sehen.

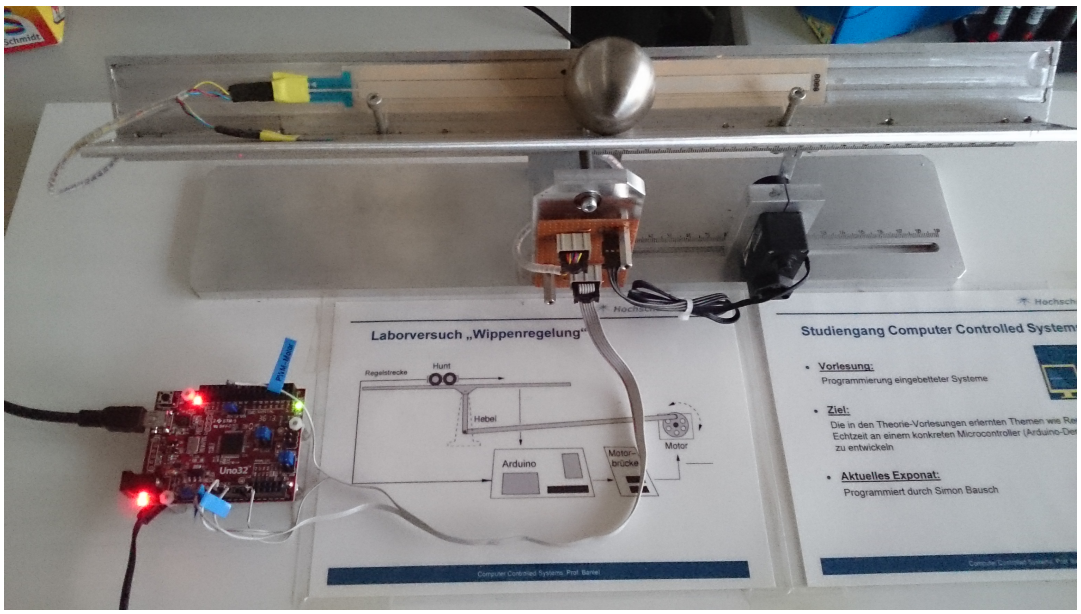

Abbildung 1.1: „Beweisfoto" des Versuchs Wippe

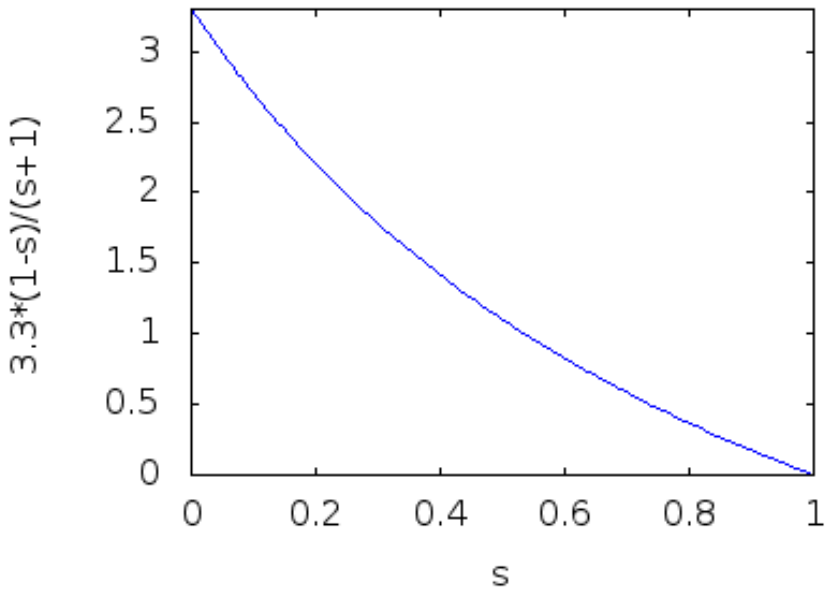

Abbildung 1.2: Uad(s)

Die Wippe wird mittels eines Servomotors geneigt. Die Neigung der Wippe ist abhängig von der Pulsbreite des Rechtecksignal, mit dem der Servo gesteuert wird.

1.4 Modellierung

1.4.1 Messung der Position

Damit die Position der Kugel geregelt werden kann, ist eine Positionsmessung zwingend notwendig. Hierfür stehen die Folienpotentiometer und eine Analog-Digital Wandler zur Verfügung. Dieser besitzt eine Auflösung von 10 Bit. Nachdem eine Messung mit dem AD-Wandler durchgeführt wurde, wird aus dem Ergebnis die Position berechnet. Damit die Berechnung schnell durchgeführt werden kann, wird auf die Verwendung von Variablen des Typs float verzichtet. Aus diesem Grund muss die Gl. (1.1) angepasst werden.

Für die Berechnung der Position wird die Gleichung 1.1 so umgeformt, dass deren Eingangsbereich zwischen -1000 und 1000 liegt und der Ausgangsbereich zwischen 0 und 1023 liegt. Dadurch ergibt sich folgende Gleichung

$$U_{\text{A/D-neu}}(s) = -\frac{1024(s-1000)}{x+3000} \qquad (-1000 \leq s \leq 1000) \tag{1.2}$$

Für die eigentliche Berechnung muss diese Gleichung nach s aufgelöst werden.

$$s(U_{\text{A/D-neu}}) = -\frac{2000(3 \cdot U_{\text{A/D-neu}} - 1024)}{2 \cdot U_{\text{A/D-neu}} + 2048} \tag{1.3}$$

1.4.2 Einstellung der Neigung

Damit die Neigung der Wippe verändert werden kann, steht ein Servomotor zur Verfügung. Die Stellung des Motors wird durch die Pulsbreite des Rechtecksignal verändert. Die Pulsbreite muss zwischen 1ms und 2ms liegen. Dabei bedeutet eine Pulsbreite von 1ms eine komplette Auslenkung nach links. Bedingt durch den Aufbau der Wippe, entspricht ein kompletter Ausschlag nicht gleichzeitig die steilste Neigung. Aus diesem

Grund wird die kleinste und größte (sinnvolle) Pulsbreite experimentell ermittelt. Zusätzlich wird noch ermittelt bei welcher Pulsbreite die Wippe Waagerecht steht.

1.4.3 Regelung der Position

Für die Durchführung der Aufgabe wird noch ein Regler benötigt. Dieser wird als Proportional Regler implementiert. Dies bedeutet aus dem Sollwert und dem Istwert wird die Differenz berechnet. Dies entspricht der Abweichung von Sollwert. Diese Abweichung wird mit einem Konstanten Faktor P multipliziert. Das Ergebnis dieser Berechnung wird dazu verwendet die Neigung der Wippe einzustellen. In Abb. 1.3 ist die Struktur des Regelkreises zu sehen. Dabei ist R der Regler, Rs die Regelstrecke und M die Positionsmessung. Die Konstante P des Reglers wird experimentell ermittelt.

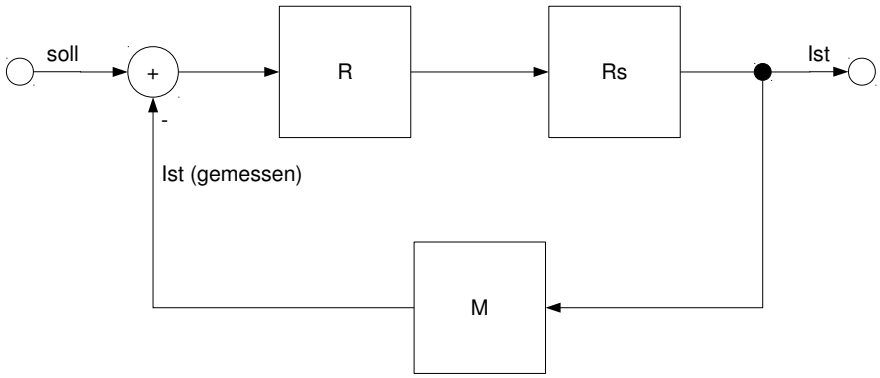

Abbildung 1.3: Reglerstruktur der Wippe

1.4.4 Anzeige der Position

Eine weitere Aufgabe ist es die Position über eine Serielle Schnittstelle auszugeben. Dies wird realisiert, indem alle 0,5s die gemessene Position in Millimeter umgerechnet und ausgegeben wird.

1.5 Implementierung

1.5.1 Setup des Boards

Die unten stehende Tabelle zeigen die Belegung der Pins die für die Ansteuerung des Servomotors und das Messen der Spannung verwendet wird.

PWM_OUT (PWM-Signal Ausgang)	9
ANALOG_IN (Analog-Digital Wandler Eingang)	14

Tabelle 1.1: Pinbelegung der Schieberegister für die LEDs

Die Implementierung des Setups ist in Listing 1.1 zu sehen.

```
1  /* Defines */
2  #define T_PWM            280000       /* Alle 7 ms */
3  #define T_PWM_MIN        27000
4  #define T_PWM_MAX        73000
5  #define T_PWM_MIDDLE     48000
6  #define T_CTRL           4000        /* Alle 100us */
7
8  #define ANALOG_IN   14
9  #define PWM_OUT 9
10
11
12 /* Globale Variablen */
13 uint32_t u32_pw;         /* Pulsbreite  */
14 int32_t i32_position;    /* Position der Kugel */
15 int32_t p_value;         /* P-Wert fuer den Regler */
16
17 void setup(void)
```

```
18 {
19     u32_pw = T_PWM_MIN;  /* Mittelstellung */
20
21     pinMode(PWM_OUT, OUTPUT);
22     pinMode(ANALOG_IN, INPUT);
23     Serial.begin(9600);
24
25     attachCoreTimerService(pwm_callback);
26     attachCoreTimerService(ctrl_callback);
27     /* Experimentel ermittelter P-Wert fuer den Regler */
28     p_value = 30;
29
30 }
```

Listing 1.1: Setup Funktion

Neben der Initalisierung der Variablen und der verwendeten Peripherie werden in der Funktion **setup()** zwei Timer-Interrupts eingerichtet.

Die erste Interrupt-Funktion ist die **pwm_callback()**. Mittels dieser Funktion wird das PWM-Signal generiert. Der Zeitabstand zwischen zwei Interrupts ist abhängig von der eingestellten Pulsbreite.

Die zweite Interrupt-Funktion ist die **ctrl_callback()**. In dieser Funktion ist der Regler implementiert. Diese Funktion wird all 100μs aufgerufen.

1.5.2 Einstellen und Generieren des PWM-Signals

Für die Generierung des PWM-Signals, wurde die Funktion **pwm_callback()** geschrieben. Diese wird als Callback Funktion eines Coretimer verwendet. Übergeben wird der

Funktion die Aktuelle Zeit. Der Rückgabewert ist die Zeit in der die Funktion das nächste mal aufgerufen wird.

In dieser Funktion wird der digitale Ausgang **PWM_OUT** abwechselnd auf High oder Low geschaltet. Für den Fall, das der Pin auf High geschaltet wird, ist der Rückgabewert der Übergabewert addiert mit dem Wert der Variable **u32_pw**. Dies ist eine Globale Variable mit der die Pulsbreite eingestellt wird.

Für den Fall, das der Ausgang **PWM_OUT** auf Low geschaltet wird, ist der Rückgabewert der Funktion der Übergabewert addiert mit der Differenz der gesamten Periodendauer **T_PWM** und dem Wert der Variable **u32_pw**. Dies bewirkt das die Periodendauer des PWM-Signals immer gleich ist.

Da der Wertebereich der Variable **u32_pw** nicht voll ausgenutzt wird, ist für die Einstellung eines Wertes eine Funktion implementiert worden. Die Funktion **set_pw()** sorgt lediglich dafür das ein gewisser Wert nicht über bzw. unterschritten wird.

Die Implementierung dieser Funktionen ist im Listing 1.2 zu sehen.

```
1  /* Funktion fuer die Generierung eines Pulsbreiten Moduliertes
       Signals */
2  /* Sie wird als Callback-Funktion eines Core-Timer verwendet */
3  uint32_t pwm_callback(uint32_t currentTime)
4  {
5      static bool last = true;
6      uint32_t u32_return;
7
8      digitalWrite(PWM_OUT, last);
9
10     if(last)
11         u32_return = u32_pw + currentTime;
12     else
```

```
13          u32_return = currentTime + (T_PWM - u32_pw);
14
15      last = !last;
16
17      return u32_return;
18  }
19
20
21  /* Funktion um die Pulsbreite einzustellen */
22  /* Wird ein Wert der groesser ist als das ermittetlte Maximum
        oder kleiner als das Minimum */
23  /* wird dieser begraenzt */
24  void set_pw(uint32_t pw)
25  {
26      if(pw > T_PWM_MAX)
27          u32_pw = T_PWM_MAX;
28      else if(pw < T_PWM_MIN)
29          u32_pw = T_PWM_MIN;
30      else
31          u32_pw = pw;
32  }
```

Listing 1.2: Funktionen für die Generierung und Einstellung des PWM-Signals

1.5.3 Messen der Position

Für die Positionsbestimmung der Kugel gibt es die Funktion **get_position()**. In dieser wird als erstes der AD-Wandler ausgelesen und dadurch die Position berechnet. Die Berechnung erfolgt mittels der Gleichung 1.3. Die Implementierung dieser Funktion ist in Listing 1.3 zu sehen.

```
1  /* Funktion um die Position der Kugel zu bestimmen */
2  /* Sie liesst den AD-Wandler aus und berechnet daraus die
       Position */
3  void get_position(void)
4  {
5      uint16_t u16_adc;
6
7      /* Aktuellen Wert Messen */
8      u16_adc = analogRead(ANALOG_IN);
9
10     /* Position berechnen */
11     i32_position = 2000 * (3*u16_adc-1024) /(2*u16_adc+2048);
12 }
```

Listing 1.3: Funktionen für die Bestimmung der Kugelposition

1.5.4 Positionsregelung

Um die Position der Kugel zu regeln wurde die Funktion **ctrl_callback()** implementiert. Diese wird als callback Funktion eines Coretimer verwendet. Der Übergabewert ist die Zeit in der die Funktion aufgerufen ist. Der Rückgabewert ist die Zeit in der die Funktion das nächste mal aufgerufen werden soll.

Der Regler ist folgendermaßen implementiert.

1. Ermitteln der aktuellen Position der Kugel mittels **get_position()**. Aus dieser wird die Abweichung zu der Position 0 berechnet.

2. Die Abweichung wird mit der Variable **p_value** multipliziert und mit der Konstante **T_PWM_MIDDLE**. In dieser Konstante ist die Pulsbreite hinterlegt in

der die Wippe waagerecht liegt. Die Addition mit dieser Konstante ist notwendig, da die Pulsbreite positiv sein muss.

3. Als letztes wird der berechnete Wert mittels **set_pw()** als Pulsbreite gesetzt.

Die Implementierung dieser Funktion ist in Listing 1.4 zu sehen.

```
1  /* Funktion die einen Regler beinhaltet */
2  /* Sie wird als Callback-Funktion eines Core-Timer verwendet */
3  uint32_t ctrl_callback(uint32_t currentTime)
4  {
5      int32_t i32_Pterm, i32_error;
6
7      /* Position bestimmen */
8      get_position();
9
10     /* Abweichung berechnen */
11     i32_error = 0 - i32_position;
12
13     i32_Pterm = i32_error * p_value;
14
15     i32_Pterm += T_PWM_MIDDLE;
16     set_pw((uint32_t) i32_Pterm);
17
18     return (currentTime + T_CTRL);
19 }
```

Listing 1.4: Funktionen für die Regelung der Kugelposition

1.5.5 Ausgabe der Kugelposition

Die Position der Kugel soll über eine Serielle Schnittstelle ausgeben werden. Dies wurde in der Funktion **loop()** realisiert. Durch das regelmäßige aufrufen der Funktion **ctrl_callback()**, liegt in der Variable **u32_pw** immer ein nahezu aktueller Wert der Position. Die Ausgabe sieht folgendermaßen aus:

1. Berechnung der Position in Millimeter. Dies geschieht indem der gemessene Wert durch zehn dividiert wird.

2. Anschließend wird dieser Wert mittels der Funktion **Serial.print()** ausgegeben.

3. Zuletzt wird, ebenfalls mit **Serial.print()**, der Text " mm \n". Nachdem dieser Text ausgeben gesendet wurde, wird mit der Arduino-Funktion **delay()** für 500ms gewartet.

Die Implementierung der Positionsausgabe ist in Listing 1.5 zu sehen.

```
1 void loop(void)
2 {
3     Serial.print( ((float)i32_position) /10);
4     Serial.print("␣mm\n");
5     delay(500);
6 }
```

Listing 1.5: loop-Funktion mit der Positionsausgabe