Universität Siegen

Embedded Control Laboratory

Ball on inclined plane

Report of lab excercise 3

WS 2016/17

Group 11

901510 David Schlosser
918781 Stefan Schmidt

Inhaltsverzeichnis

1 Bericht

1.1 Einleitung

In diesem Versuch haben wir vor, die in Lab 2 errechneten Werte für den PID-Regler in den vorhandenen Code einzufügen und so ein funktionierendes System zu erlangen. Zusätzlich müssen wir die Werte für $MIDPOS$ und $DELTA$ berechnet werden und den Willkommenstext auf der LCD-Anzeige geändert werden.

1.1.1 Verwendung von OCR3A

OCR3A ist das Timer Output Compare Register, ...
TODO

1.1.2 Berechnung von MIDPOS

Der richtige MIDPOS Wert gibt an, wann die Fläche vollkommen horizontal steht. Zur Berechnung von MIDPOS benutzten wir folgende Formel:

$$MIDPOS = Frequenz/Prescaler/1000 * Pulslaenge \tag{1}$$

Die Frequenz ist mit $16Mhz$ gegeben. Der Prescaler-Wert ist auf den Wert $0b00000011$ am Port $TCCR3B$ gesetzt. Mit Hilfe des Datenblattes kann diese Bitfolge auf den Wert 64 dekodiert werden (...). Die Pulslänge der PWM für eine horizontaler Lage ist in der Aufgabenstellung erwähnt und liegt bei $1,5ms$.
Das Ergebnis unserer Berechnung ergibt $MIDPOS = 375$. Allerdings ist die Fläche nach dem Sertzen von $MIDPOS$ auf den errechneten Wert nicht vollständig horizontal. Der Wer muss auf $MIDPOS = 355$ angepasst werden, damit unser Modell in der Waage ist.
Zu beachten ist, dass der Wert von $MIDPOS$ eventuell vor jeder Versuchsdurchführung neu angepasst werden muss, um zum Beispiel ein Schifstehen des Tisches oder ähnliches auszugleichen.

1.1.3 Berechnung von DELTA

$DELTA$ ist die Differenz zwischen der maximalen linken oder rechten Position der Fläche und der mittleren Position ($MIDPOS$). Für die Berechnung des Wertes von $DELTA$ verwenden wir die folgende Formel:

$$DELTA = (Frequenz/Prescaler/1000 * Pulslaenge) - MIDPOS \quad (2)$$

Für den Wert der Pulslänge muss diesmal allerdings $2ms$ verwendet werden, da wir den Wert für die maximale rechte Position der Fläche benötigten. Von diesem Wert ziehen wir $MIDPOS$ ab, um die Differenz zu erhalten. Nun müssen wir wieder den Wert leicht anpassen und kommen auf das Ergebnis $DELTA = 125$.

1.1.4 Willkommens-Anzeige

Die Willkommensnachricht auf der LCD Anzeige ist in der Datei $HMI.c$ in Zeile 260 festgelegt und kann einfach mit der richtigen Gruppennummer angepasst werden.

1.1.5 PID Implementation

Die Implementierung des PID-Reglers erfolgt in der Datei $PID.c$. Wir verwenden folgende Summenformel für den PID-Regler:

$$y = (proportional * error) + (integral * errorSum * sampleTime) + (derivative * (error * errorLast)/sampleTime) \quad (3)$$

Die Werte für *proportional*, *integral* und *derivative* sind die Faktoren für den P-, I- und D-Teil des Reglers. *error* beschreibt den aktuellen Fehler (Sollwert - Istwert). *errorSum* ist die Summer aller bisherigen Fehler. *errorLast* beschreibt den letzten Fehler (nicht den aktuellen).
Die *sampleTime* haben wir auf $18ms$ $(0,018s)$ gesetzt, da die Ansteuerung des Servo-Motors alle $18ms$ erfolgt.
Im $Versuch2$ haben wir den PID-Regler simuliert. Leider können die Werte

aus der Simulation nicht übernommen werden. Ein Grund dafür ist, dass in der Simulation ein kontinuierlicher PID simuliert wird, wir am Modell jedoch einen diskreten PID vorfinden. Außerdem werden in der Simulation Einflüsse wie zum Beispiel die nicht perfekte Rundheit des Balls, die Trägheit des Servo-Motors und so weiter ignoriert.
Nach längerem Kalibrieren kommen wir auf folgende Werte:
$proportional = 0,19$, $integral = 0,018$ und $derivative = 0,51$

1.1.6 Vergleich der Ergebnisse mit Versuch 2

Im Vergleich zu Versuch 2 benötigten wir beim eigentlichen Modell vollkommen andere PID-Werte. Ein möglicher Grund dafür ist, dass wir in Versuch 2 ein kontinuierliches System simuliert haben, in Versuch 3 aber mit einem diskreten System arbeiten haben. Der Unterschied ist, dass bei einem diskreten PID die *sampleTime*, also eine Verzögerungszeit mit eingerechnet werden muss. Dadurch ändert sich die Formel des Reglers.
Außerdem werden, wie oben schon genannt, äußere Einflüsse ignoriert (Rundheit des Balls, Trägheit des Motors, etc).

1.1.7 Queues und Tasks

Es gibt drei Queues *QueueTaster*, *QueueSensor* und *QueueServo*, sowie sechs Tasks $vSensor$, $vIO\_SRAM\_to\_LCD$, $vTaster$, $vHMI$, $vServo$ und $vPID$.

QueueTaster
Diese Queue erhält Input-Daten von $vTaster$. Diese werden dann weiter gegeben an $vHMI$ und an $change\_para$.
QueueSensor
QueueSensor empfängt die Positionsdaten des Balls von $vSensor$ und sendet diese an $vPID$.
QueueServo
QueueServo verwaltet den PID-Wert der in $vPID$ berechnet wurde und gibt sie an $vServo$ weiter.

vSensor
Dieser Task berechnet die aktuelle Ballposition mit Hilfe der Sensoren und sendet die Position an *QueueSensor*.
vIO_SRAM_to_LCD
Dieser Task kopiert Daten vom RAM zum LCD und positioniert den Cursor im Menü.
vTaster
vTaster liest Input Daten an den Pins und sendet diese Parameter an *QueueTaster*.
vHMI
Dieser Task steuert/verwaltet die Menüs und Funktionen, welche auf dem LCD dargestellt werden, überprüft welcher Parameter zur Zeit in *Queuetaster* ist und verwendet diesen zur Menüsteuerung.
vServo
Dieser Task berechnet den Vergleichswert des Servosignals und setzt diesen Wert auf den Port *OCR3A*. *vServo* erhält den Wert *BETA* der für die Berechnung benötigt wird aus *QueueServo*.
vPID
Dieser Task holt sich die aktuelle Ballposition aus *QueueSensor* und berechnet damit den aktuellen PID-Wert.

1.1.8 Inter Task Kommunikation

TODO

1.2 Ergebnis

Mit unseren aktualisierten PID-Werten ist es uns gelungen den Ball in weniger als zehn Sekunden an der gewünschten Stelle stehen bleiben zu lassen.

1.3 Fazit

TODO