2Wheeler

Embedded Software Engineering

Supervisor:
Univ.-Prof. Dr Christoph Kirsch

Autoren:
Anton Pölzleitner
Stefan Lukesch
York Keyser

10. Februar 2011

Inhaltsverzeichnis

1 Einführung

Im Rahmen der Lehrveranstaltung „Embedded Software Engineering“ bestand die Aufgabe ein überschaubares Projekt auf einem Embedded Device zu lösen. Die Entscheidung fiel auf den Nachbau eines *Segways*^{TM} . Als geeignete Plattform wurde *Lego Mindstorms*^{TM} gewählt.

1.1 *Segway*™

Ein *Segway*^{TM} ist ein Einpersonen-Transportmittel, das seit 2001 von der Firma Segway Inc. hergestellt wird. Es handelt sich im Wesentlichen um eine Plattform mit zwei, auf einer Achse liegenden Rädern, angetrieben durch einen Elektromotor. Der *Segway*^{TM} ist in der Lage sich selbst auszubalancieren. Die Steuerung für Vorwärts-/Rückwärtsfahren und das Lenken erfolgt ausnahmlos über die Lageveränderung des Fahrers, der auf der Plattform steht. Es gibt den *Segway*^{TM} inzwischen in vielen verschiedenen Ausführungen für unterschiedlichste Anwendungsgebiete, angefangen bei rein privaten Anwendungen bis hin zu Einsätzen bei Polizei und Rettung.

Abbildung 1: Der 2Wheeler und ein original *Segway*^{TM} in Straßenausführung

1.2 *Lego Mindstorms*™

1998 veröffentlichte die Firma Lego einen Bausatz unter dem Namen RIS (Robotics Invention System) mit einem programmierbaren Baustein und verschiedenen Sensoren und Motoren. Im Jahr 2009 folgte die dritte Generation unter dem Namen *Lego Mindstorm NXT 2.0*, die auch bei diesem Projekt Verwendung fand.

Den Kern des Systems bildet der so genannte NXT-Brick, ein Multi-Sensor-Baustein mit folgenden technischen Spezifikationen:

- 32-bit ARM7 microcontroller (56 Kbytes FLASH, 64 Kbytes RAM)
- 8-bit AVR microcontroller (4 Kbytes FLASH, 512 Byte RAM)
- 4 Input Ports
- 3 Output Ports
- 100 x 64 pixel LCD Bildschirm
- Lautsprecher
- Bluetooth Schnittstelle
- USB 1.0 port (12 Mbit/s)

Für die Stromversorgung werden sechs AA-Batterien benötigt. Für dieses Projekt wurde jedoch eine externe Stromquelle verwendet, um zu vermeiden, dass schwächer werdende Batterien die Fahr-Eigenschaften des 2Wheelers während der Tuning-Phase (siehe Abschnitt 3.4) verändern.
Lego bietet für die Programmierung des NXT-Bricks eine graphische Programmiersprache namens NXT-G, basierend auf einer LabView Plattform an. Da es jedoch für das Projekt wesentlich interessanter war, den Source-Code selbst zu schreiben und zu optimieren, wurde die Programmiersprache NXC eingesetzt.

1.3 Not Exactly C - NXC

NXC ist eine frei verfügbare höhere Programmiersprache mit C-ähnlicher Syntax. Sie basiert auf dem Next Byte Code (NBC), einer Assembler-ähnlichen Programmiersprache für die Programmierung des NXT-Bricks. Die Sprache hat zwar einige Einschränkungen, so wird zum Beispiel keine Rekursion unterstützt und es gab bis zur letzten Version keine Float-Datentypen. Die Einschränkungen der Sprache kamen bei diesem Projekt jedoch kaum zum Tragen.
Als IDE wurde das BrixcCommandCenter unter Windows verwendet, da relativ früh in der Entwicklung klar wurde, dass die Tools, die für MacOSX angeboten werden, noch relativ schlecht funktionierten. So war zum Beispiel das Auslesen der Variablen-Belegungen und Sensorwerte für Debugging-Zwecke unter MacOSX mit den NeXTTools nicht möglich.

2 Der 2Wheeler

2.1 Der Aufbau

Das eigentlich recht einfache Design des *Segways*^{TM} ließ sich auch mit Lego relativ einfach nachbauen. Sämtliche Teile, die benötigt wurden, sind Bestandteil des *Lego Mindstorms*^{TM} Basispaketes. Benötigt wurden für das Modell neben diversen Steckteilen:

- ein NXT-Brick,
- ein Farbsensor,
- ein Berührungssensor,
- zwei Motoren.

2.2 Programmablauf

Nach dem Start des Programms wird zuerst die Neutrallage mit Hilfe des Farbsensors ermittelt. Der Sensor misst die Helligkeit des von der Standfläche reflektierten Lichtes. Der Berührungssensor dient als Start/Stop-Schalter für den Roboter. In der Hauptschleife des Programms wird periodisch die aktuelle Lage über den Farbsensor gemessen und dann durch den Kontroller (siehe Abschnitt 3) die nötige Ausgleichsbewegung berechnet, um den Roboter wieder in eine neutrale Lage zu bekommen.

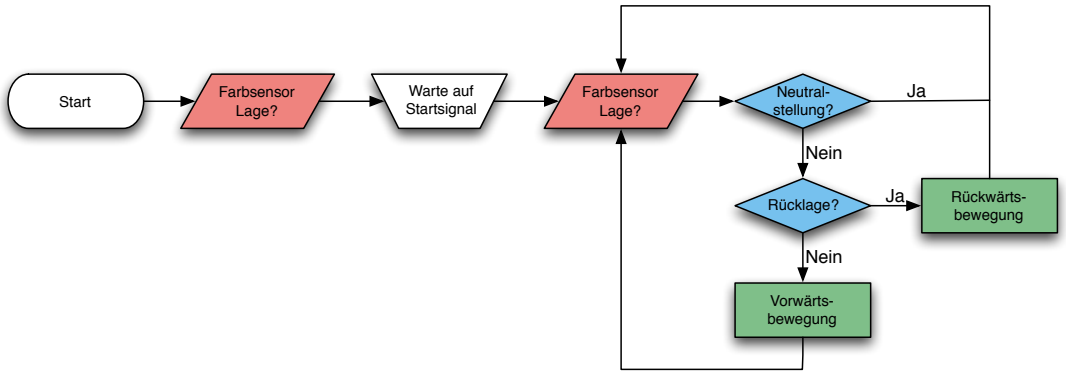

Abbildung 2: Das Flussdiagramm des fertigen Lego-Segways

In den folgenden Abschnitten soll noch kurz der Kontroller, den wir verwendet haben etwas näher beschrieben werden und einige Schwierigkeiten, auf die wir gestoßen sind aufgezählt werden.

3 PID Kontroller

Kurz für Proportional Integral Derivate Kontroller, ist ein klassischer stetiger Regler. Stetige Regler liefern am Ausgang ein kontinuierliches Signal, dass jeden Wert innerhalb eines bestimmten definierten Intervalls annehmen kann. PID Regler berechnen einen so genannten *error*-Wert, welcher die Abweichung zwischen einem bestimmten Sollwert und dem aktuell gemessenen Sensorwert darstellt, und versuchen dann diese Abweichung zu minimieren. Für diesen Schritt ist der PID Regler in drei Elemente aufgeteilt, die zusätzlich einzeln parametrisiert werden, um ein ideales Ergebnis für die jeweiligen Anforderungen zu erzielen. Die Auswahl des richtigen Reglers für das jeweils vorliegende Problem ist entscheidend. Alternativen bzw. Abwandlungen des PID Reglers sind, der P, I, PI, PD Regler, diese unterscheiden sich hauptsächlich in Geschwindigkeit und Genauigkeit, bezogen auf das Ausgleichen des error-Wertes.
Das Schema eines PID Reglers sieht nun folgendermaßen aus:

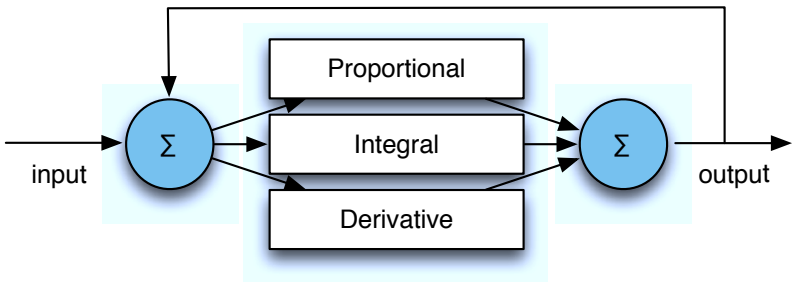

Abbildung 3: Proportional Integral Derivate Kontroller Diagramm

$$\mathrm{PID_{out}(t)} = P_{\mathrm{out}} + I_{\mathrm{out}} + D_{\mathrm{out}}$$

mit $P_{\mathrm{out}}, I_{\mathrm{out}}, D_{\mathrm{out}}$ als die jeweiligen Outputs der einzelnen drei Elemente des Reglers.

3.1 Proportional

Der Proportional Teil des PID Reglers spiegelt die aktuelle Abweichung des definierten Sollwertes proportional verändert durch den Faktor Kp. Genauer gesagt

wird der errechnete aktuelle *error*-Wert durch Kp verstärkt. Wird dieser Faktor zu hoch oder zu klein gewählt, kann das System instabil werden, den optimalen Wert Kp zu finden ist äußerst wichtig.

$$P_{\text{out}} = K_p\, e(t)$$

P_{out}	...	Proportional Output
K_p	...	Parameter zur Feineinstellung
e	...	*error*-Wert, also die Differenz von Istwert - Sollwert
t	...	aktuelle Zeit

3.2 Integral

Das so genannte Gedächtnis des PID Reglers stellt der Integral Teil dar. Hier werden alle vorangegangenen gemessenen *error*-Werte aufsummiert und mit einem Faktor Ki multipliziert. Es wird also nicht nur der Wert sondern auch die Dauer der Abweichung berücksichtigt. Dadurch können auch sehr kleine Abweichung, über längere Zeit aufsummiert, eine gewisse Größe erlangen, die schlussendlich ausgeglichen wird. Dieses Element des Reglers reagiert langsamer als der Proportional Teil, dafür aber exakter und es ist möglich auch kleinere Fehler zu beheben.

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

I_{out}	...	Integral Output
K_i	...	Parameter zur Feineinstellung
e	...	*error*-Wert, also die Differenz von Istwert - Sollwert
t	...	aktuelle Zeit
τ	...	Integrationsvariable

Ein Effekt der in diesem Element auftreten kann, ist der so genannte "Wind-Up-Effekt". Dabei wird nach längeren, großen einseitigen error-Werten, beim Rücklauf der Wert des Integral Teiles nicht schnell genug abgebaut und damit der Sollwert überschritten bis genug große *error*-Werte auf der anderen Seite erzielt wurden. Man kann diesen Effekt auf zwei Arten umgehen, zum Einen kam man den Integralwert, jedes Mal wenn $error = 0$ ist, ebenfalls auf 0 setzen. Oder man versucht den Integralwert zu dämpfen, indem man dem Parameter Ki einen Wert kleiner 0 zuweist.

3.3 Derivate

Im dritten, dem so genannten Derivate Element, wird nicht der *error*-Wert an sich, sondern die Stärke der Veränderung zum letzten *error*-Wert, also die Differenz

zwischen dem alten und dem aktuellen *error*-Wert, betrachtet. Es wird hier also versucht, die nächste Änderung vorherzusehen und in die aktuelle Berechnung mit einzubeziehen. Somit wirkt sich das Ganze als eine gewisse Dämpfung auf die gesamte Regelung aus, die sich vor allem um den Nullpunkt, sprich den Sollwert, am meisten auswirkt. Im Fall unseres 2Wheelers sorgt das Derivate Element für eine geschmeidigere Bewegung im nahezu ausbalancierten Zustand.

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

D_{out}	...	Derivate Output
K_d	...	Parameter zur Feineinstellung
e	...	*error*-Wert, also die Differenz von Istwert - Sollwert
t	...	aktuelle Zeit

3.4 Schleifen Tuning

Der letzte, aber mit Sicherheit zeitaufwändigste Teil bei der Erstellung eines PID Reglers ist schlussendlich das Optimieren der Parameter (Kp, Ki, Kd) der einzelnen Elemente. Dazu gibt es einige Methoden die diesen Prozess erleichtern bzw. vereinfachen können, namentlich Ziegler-Nichols - oder Cohen-Coon Methode, diverse Software Tools, ... Leider gibt aber keine dieser Methoden eine Garantie für ein konkretes Problem eine optimale Lösung zu finden. Somit entschieden wir uns für manuelles Tuning, was in unserem Fall zu einer zeitintensiven Trial-and-Error Methode wurde.

4 Infinite Impulse Response Filter

Da der Roboter nicht nur auf der Stelle stehen sollte sondern sich eigentlich auch vor- und rückwärts bewegen können sollte, wurde noch versucht den 2Wheeler mit einem zweiten, vielleicht effizienteren Filter laufen zu lassen. Das sollte es auch ermöglichen den 2Wheeler in eine bestimmte Richtung fahren zu lassen, was mit dem PID-Kontroller aus Abschnitt 3 bisher nicht gelang. Wir haben uns für einen zeitdiskreten, linearen, verschiebungsinvarianten Filter namens *Infinite Impulse Response Filter* oder auch IIR genannt, entschieden. Unter einem zeitdiskreten, linearen, verschiebungsinvarianten Filter versteht man ein System, welches eine Linearität aufweist und zusätzlich unabhängig von zeitlichen Verschiebungen ist.
Dieser Filter hat die Möglichkeit Spitzenwerte, die der Sensor übermittelt, auszufiltern und daher nicht zu ruckartig auf diveres Inputwerte zu reagieren. Die Hoffnung hierbei war, dass der 2Wheeler ruhiger steht oder ruhigere Bewegungen machen würde. Dieses würde uns die Vorwärtsbewegung vielleicht erleichtern.

4.1 Der Algorithmus

$$y[n] = \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] + \cdots + b_P x[n-P] - a_1 y[n-1] - a_2 y[n-2] - \cdots - a_Q y[n-Q])$$

P	...	Output Filter Grenze
b_i	...	Koeffizient b des Output Filters am Index i.
Q	...	Input Filter Grenze
a_i	...	Koeffizient a des Output Filters am Index i.
$x[n]$	...	Signal Input
$y[n]$	...	Signal Output

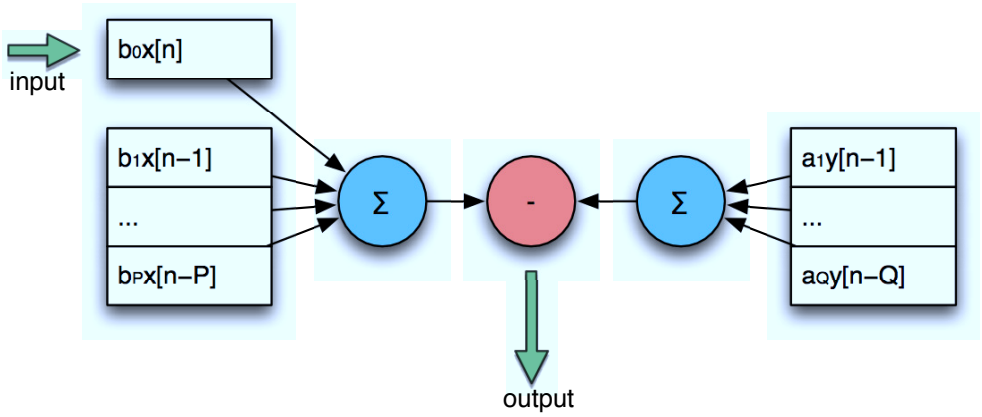

Abbildung 4: Infinite impulse response filter Diagramm

Es gibt zwei Fenster, die eine individuelle Länge haben können. Durch die individuelle Länge besteht die Möglichkeit, den Output über längerer Distanz in die Auswertung des neuen Input einfließen zu lassen. Die Koeffizienten ermöglichen eine Wichtung der einzelnen Punkte. Durch diese Wichtung hat jeder einzelne Punkt einen individuellen Einfluss auf den neuen Outputwert.

4.2 Warum er nicht eingesetzt wurde

Das schwierigste bei der Umsetzung dieses Algorithmus ist, wie bei dem PID Regler, die richtige Wichtung zu finden. Um die richtigen Koeffizienten zu finden müsste man zB. einen Datensatz im Mathematica bearbeiten. Der NXT-Brick bietet leider keine einfache und sinnvolle Möglichkeit die nötigen Daten zu sammeln und auszuwerten. Somit muss man die Koeffizienten per Hand testen.

Diese sanfte Ausbalancierung unserer Inputwerte brachte aber das Problem mit sich, dass sollte der *Lego Mindstorms Brick*^{TM} zu Fallen beginnen, er teilweise zu langsam oder überhaupt nicht reagiert. Das Problem sollte mit der richtigen Wichtung noch besser gelöst werden können, war uns aber aus Zeitgründen nicht mehr möglich.

5 Konklusion

Abschließend soll noch einige Probleme hingewiesen werden die im Laufe des Projekts zu Tage kamen.
Der Farbsensor bietet zwar einige Möglichkeiten zur Erkennung von Farben und Helligkeitsunterschiede, in unserem Fall waren die Ergebnisse jedoch sehr ungenau und teilweise wirkten sie eher zufällig. Auch die Oberfläche auf der der 2Wheeler stand hatte großen Einfluss auf die Sensorwerte, so waren zum Beispiel leichte Maserungen in der Tischplatte schon genug um ihn aus der Balance zu bringen.
Auch die Motoren weisen eine nicht unerhebliche mechanische Toleranz auf, was gerade bei kleinen Ausgleichsbewegungen zu ruckartigen Bewegungen und stellenweisem Über- oder Untersteuern geführt hat.
Beim IIR Filter (4) machte sich auch die Rechenleistung des Kontrollers bemerkbar.

6 Arbeitsaufteilung

Der Großteil der Arbeit, vor allem das Tuning der Kontroller Parameter wurde in gemeinsamer Arbeit ausgeführt, die Aufgabenbereiche waren ungefähr wie folgt verteilt:

Anton Pölzleitner:

- Bautechnische Optimierung des 2Wheelers
- Tuning und Verbesserung des PID Kontrollers
- Implementierung des IIR Filter
- Kapitel 3 in der vorliegenden Arbeit

Stefan Lukesch:

- Installation Basissystem und IDE
- Implementierung des Basisprogrammes und einfacher PID Kontroller

- Kapitel 1 - 2, 5 in der vorliegenden Arbeit

York Keyser:

- Zusammenbau des Grundmodels
- Tuning und Implementierung des IIR Filters
- Kapitel 4 in der vorliegenden Arbeit

Quellenverzeichnis

[1] *Segway Personal Transporter.*
http://de.wikipedia.org/wiki/Segway

[2] *Segway Personal Transporter.*
http://www.segway.at/type-i2.php

[3] *NXT Sensors.*
http://www.legoengineering.com/nxt-sensors-2.html

[4] *Next Byte Codes & Not eXactly C.*
http://bricxcc.sourceforge.net/nbc/

[5] *Regelungstechnik.*
http://www.rn-wissen.de/index.php/Regelungstechnik

[6] *PID Controller.*
http://en.wikipedia.org/wiki/PID_controller

[7] *Filter mit unendlicher Impulsantwort.*
http://de.wikipedia.org/wiki/IIR-Filter

[8] *Infinite impulse response.*
http://en.wikipedia.org/wiki/Infinite_impulse_response

[9] Dan Lavry, Lavry Engineering *Understanding IIR (Infinite Impulse Response) Filters - An Intuitive Approach*, 1997
http://www.lavryengineering.com/white_papers/iir.pdf