# Positionsregelung mit AVR/Arduino und Python

Marius Unsel

7. April 2014

# Inhaltsverzeichnis


**1 Vorwort** **2**

**2 Aufbau** **3**
2.0.1 Bauteile . . . 3
2.0.2 Blockschaltbild . . . 4

**3 SHARP low-cost Distanzsensor auswerten** **5**
3.1 Kennlinie des Sensors . . . 6
3.2 Lookuptable mit linearer Interpolation . . . 6

**4 Avago Drehencoder auslesen** **9**

**5 Regler-Algoritmus** **10**
5.1 Beschreibung der Funktionsweise . . . 10
5.2 Bestromung des Motors mit H-Brücke . . . 11
5.3 Anmerkung zur praktischen Umsetzung . . . 11
5.4 Simulink-Modell . . . 11

**6 Python Interface** **13**
6.1 Warum Python? . . . 13
6.2 Oberfläche . . . 13
6.3 Serielle Kommunikation . . . 14

**7 Quellen** **15**

# Kapitel 1

# Vorwort

In den folgenden Kapiteln werde ich meinen Fortschritt bei dem Entwurf einer Positionsregelung dokumentieren. Die Umsetzung soll mit dem Opensource Mikrocontrollersystem Arduino stattfinden. Für Debugzwecke, zur Parametrierung des Reglers und zum Anzeigen der Messergebnisse am Computer dient die serielle Schnittstelle, die zur Datenübertragung zum Computer benutzt wird. Der $\mu$C ATmega2560, der auf dem Arduino Mega verbaut ist wird in C mit Hilfe von der Arduino IDE zur Verfügung gestellten Bibliotheken programmiert. Viele Einstellungen im Mikrocontroller erfolgen über Wertzuweisungen in Register, die sogenannten "special function register". Die notwendigen Werte und Registerbezeichnungen, sowie Codebeispiele zur Anwendung finden sich im Datenblatt des Microcontrollers.

Die seriellen Daten vom Arduino sollen von einem Python-Programm ausgewertet werden. Dieses stellt eine Visualisierung der Messdaten in Echtzeit zur Verfügung und stellt ein Interface bereit, mit dem sich Einstellungen der Regelung bequem per GUI(Graphical User Interface) vornehmen lassen. Die Umsetzung dieses Auswert-Programms wird mit dem GUI-Framework "Qt" von Nokia und der Plotbibliothek "PyQtGraph" erfolgen. Näheres dazu im Kapitel "Python Interface"

Hiermit möchte ich meinem Professor Roustaim Chakirov danken, da dieser mir die Freiräume gelassen hat, in diesem Semster Dinge zu lernen, die ich mir grßtenteils selbst ausgesucht habe. Dazu gehören die Grundlagen von Linux, Python und weiterführende Kenntnisse über Atmel Mikrocontroller sowie die absoluten Grundlagen des Satzsystems LATEX.

# Kapitel 2

# Aufbau

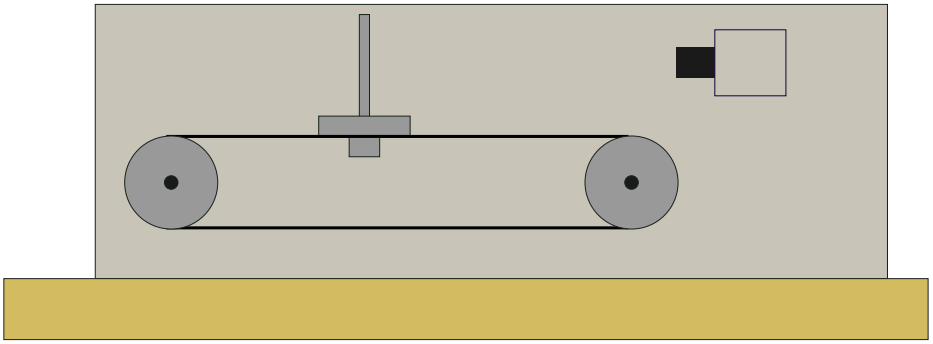


Abbildung 2.1: Skizze des Modells

Die Regelstrecke besteht aus einem Keilriemen, der zwischen zwei Umlenkrollen gespannt ist. Der Riemen wird von einem DC-Motor an einer der Rollen angetrieben und treibt einen Schlitten, der auf einer Linearführung sitzt an. An diesem Schlitten ist eine Metallplatte befestigt, mit dessen Hilfe der Sensor den Abstand des Schlittens zum Sensor bestimmen kann. Die Metallplatte reflektiert dabei die Infrarotstrahlung, die vom Sensor ausgestrahlt wird.

### 2.0.1 Bauteile

- 12V DC Motor
- SHARP Sensor
- Avago Inkrementalgeber
- Arduino Mega2560

- DFRobot Moto Shield
- igus Linearführung und Schlitten
- Keilriemen
- zwei Umlenkrollen
- Grundplatte mit Spannvorrichtung

### 2.0.2 Blockschaltbild

# Kapitel 3

# SHARP low-cost Distanzsensor auswerten

Der in diesem Versuch verwendete Infrarot-Distanzsensor von SHARP liefert eine analoge Spannung, die umgerechnet werden kann in die entsprechende Distanz. Da diese Kennlinie nicht linear ist und die entsprechende Funktion nicht bekannt ist, wird ein Verfahren zur linearen Interpolation verwendet, um aus einem Messwert und zwei Vektoren mit bekannten x-, sowie y-Koordinaten die entsprechende Distanz anzunähern.

## 3.1 Kennlinie des Sensors

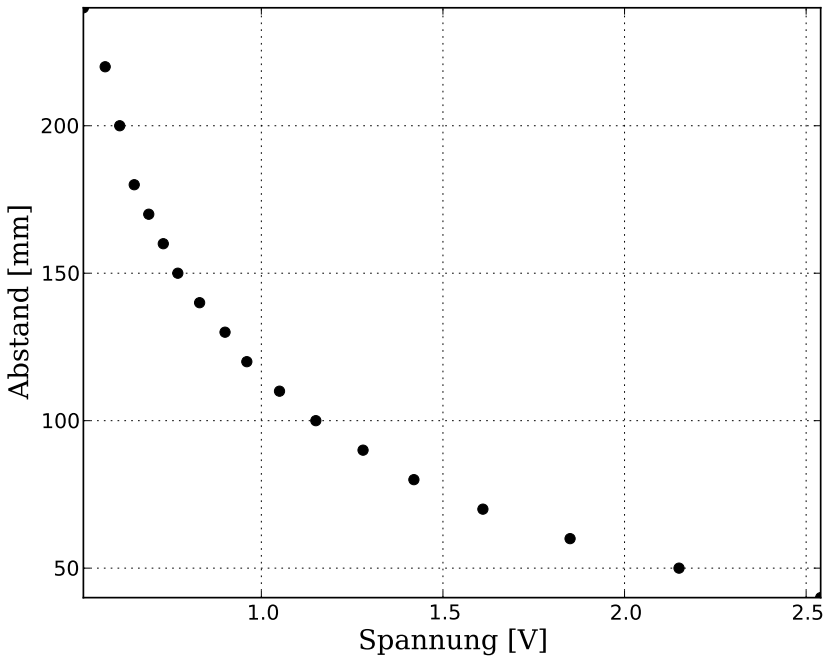



Abbildung 3.1: Kennline $x(u)$

Die Kennlinie des Sensors wurde mit Hilfe eines Multimeters und einem Maßband Gemessen. Der Abstand zwischen zwei Messungen beträgt 10mm.

## 3.2 Lookuptable mit linearer Interpolation

Die Spannungswerte $U_1, U_2, ..., U_n$ des Sensors in der Einheit Volt lauten:

$[0.51, 0.57, 0.61, 0.65, 0.69, 0.73, 0.77, 0.83, 0.9, 0.96, 1.05, 1.15, 1.28, 1.42, 1.61, 1.85, 2.15, 2.54]$

Diese sind in aufsteigender Grösse sortiert, damit der Algorithmus richtig funktioniert. für die entsprechenden Diszanzwerte $x_1, x_2, ..., x_n$ in der Einheit Millimeter gilt diese Sortierung nicht, da jeder Wert $x_i$ von einem entsprechenden Wert $U_i$ abhängt. Die Distanzwerte lauten:

$[240, 220, 200, 180, 170, 160, 150, 140, 130, 120, 110, 100, 90, 80, 70, 60, 50, 40]$

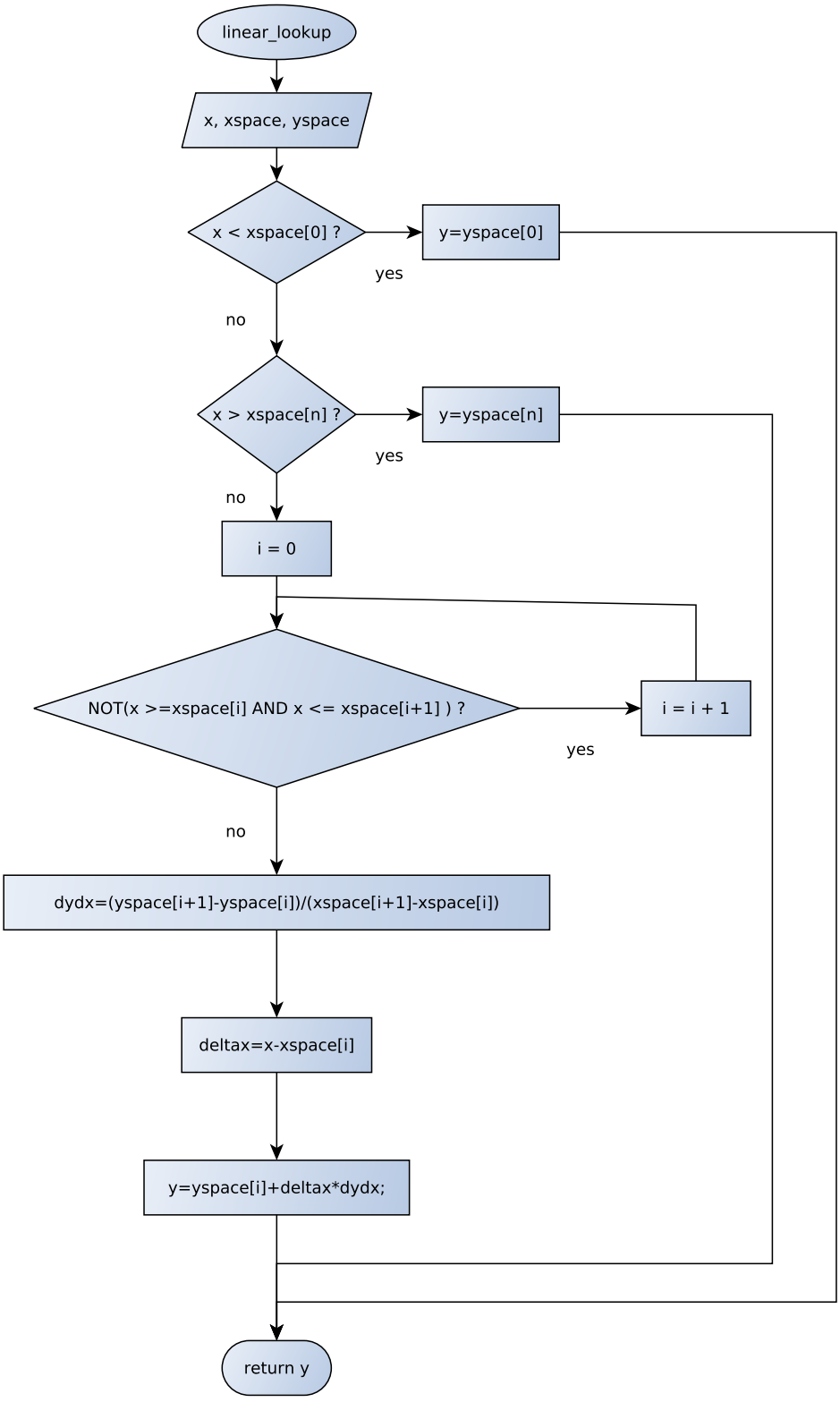



Abbildung 3.2: Algorithmus einer linearen Interpolation

Die Interpolationsroutine prüft zuerst, ob, und zwischen welchen Spannungswerten die gemessene Spannung liegt. Liegt die Spannung ausserhalb, so wird das entsprechende Randelement aus der Distanztabelle($x_1$ oder $x_n$) ausgegeben. Eine Extrapolation findet nicht statt. Liegt der Wert $U_{mess}$ nun zwischen $U_i$ und $U_{i+1}$, so wird der Differenzenquotient $\frac{\Delta x}{\Delta U} = \frac{x_{i+1}-x_i}{U_{i+1}-U_i}$ und der Abstand zwischen $U_{mess}$ sowie $U_i$ gebildet. Der interpolierte Distantwert ergibt sich dann durch $x = U_i + \frac{\Delta x}{\Delta U} * (U_{mess} - U_i)$. Die Abbildung 3.2 zeigt das Flussdiagramm des implementierten Algorithmus.

# Kapitel 4

# Avago Drehencoder auslesen

Beim Drehencoder von Avago handelt es sich um einen optischen 3-Kanal Encoder. Für dessen Auswertung wurde eine Interrupt Rooutine so eingestellt, dass diese jedes mal, wenn auf dem ersten Kanal eine positive Flanke erkannt wird aufgerufen wird. In dieser Routine wird dann gepüft, ob der aktuelle Digitalwert auf dem zweiten Kanal HIGH oder LOW ist und dementsprechend wird eine Zählervarible hoch, bzw runter gezählt. Aus diesem Zählerstand kann in regelmäßigen Abständen durch den Regleralgorithmus eine Umrechnung in eine entsprechende Winkelgeschwindigkeit, einen Winkel oder die entsprechende Position des Schlittens erfolgen. Der Encoder liefert pro Kanal 500 Impulse, es werden pro Umdrehung also 500 positive Flanken von Kanal A ausgewertet.

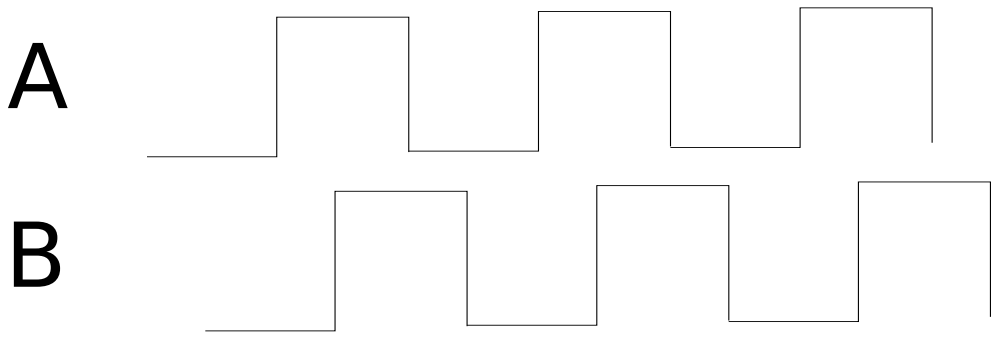



Abbildung 4.1: Kanal A und B des Encoders

# Kapitel 5

# Regler-Algoritmus

## 5.1 Beschreibung der Funktionsweise

Der Regler auf dem AtMega2560 ist als diskreter PID Regler realisiert. Dieser nimmt eine Struktur aus Zustandsvariablen und Reglerparametern entgegen und berechnet daraus die Stellgröße. Außerdem lässt sich durch Senden von Befehlen über die serielle Schnittstelle einstellen, welchen Zustand(Position oder Geschwindigkeit) als Grundlage für den Algorithmus verwendet wird. Eine neue Stellgrösse wird 100 mal pro sekunde berechnet. Der Kehrwert von 10ms fliesst als Konstante mit in die Berechnung ein, damit die Zustände in den Einheiten Sekunde und Millimeter abgespeichert sind. Die Zeitparameter $\frac{1}{T_a}$ und $T_v$ sind dann ebenfalls in der Einheit $\frac{1}{s}$ bzw. $[s]$ und der Verstärkungsfaktor $[K_p] = [\frac{1}{mm}]$. Somit lassen sich die gleichen Parameter, wie aus dem Simulink Modell in dieser Regelung übernhemen.
Die Standardabweichung ist:

$$e(t) = w(t) - x(t)$$

Dabei ist $x$ der Istwert und $w$ die Führungsgröße. Die Formel eines PID-Reglers ist:

$$y(t) = K_p \left( e(t) + \frac{1}{T_a} * \int_0^t e(t)\, dt + T_v * \frac{de}{dt} \right)$$

Da wir allerdings nur in festgelegten Abständen $T = 10ms = 0,01s$ eine neue Stellgrösse berechnen können, müssen die kontinuierlichen Anteile $\int_0^t e(t)dt$ und $\frac{de(t)}{dt}$ der Gleichung durch diskret berechenbare Formelteile ersetzt werden. Das Integral lässt sich numerisch durch eine Summe annähern:

$$\int_0^t e(t)\, dt \approx \sum e(kT) * T \qquad \text{für} \qquad k \in \mathbb{N}$$

Die Ableitung der Standardabweichung lässt sich auch als Differenzenquotient darstellen. Man wechselt dabei von der Tangentensteigung zu einer Sekantensteigung:

$$\frac{de(t)}{dt} \approx \frac{e(kT) - e((k-1)T)}{T}$$

Somit ergibt sich die Reglergleichung:

$$y(kT) = K_p \left( e(kT) + \frac{T}{T_a} * \sum e(kT) + T_v * \frac{e(kT) - e((k-1)T)}{T} \right)$$

Damit der integrierende Anteil der Gleichung nicht ins Unermessliche anwachsen kann, wird dieser ab einem bestimmten Wert festgelegt. Diese Grenze lässt sich durch ausprobieren ermitteln.

## 5.2 Bestromung des Motors mit H-Brücke

Das Motoshield ist in der Lage, zwei Gleichstrommotoren in beide Drehrichtungen zu treiben. Es enthält als zwei H-Brücken. Da wir nur einen Motor in unserer Regelung verwenden, benötigen wir nur eine dieser Brücken. Für die Ansteuerung eines Motors benötigt man zwei Ausgänge des Arduino. Einen zur Drehrichtungsbestimmung und einen zur Steuerung der gemittelten Spannung durch eine PWM. Liegt auf dem Richtungsausgang ein HIGH-Pegel an, so dreht sich der Motor die eine Rihtung, liegt ein LOW-Pegel an, so dreht er sich anders herum. Die PWM wurde durch Registerbefehle des AtMega2560 so verändert, dass Sie mit einer Trägerfrequenz von $\approx 31kHz$ arbeitet, sodass die Ansteuerung des Motors nicht mehr fï das menschliche Auge hörbar ist. Dessen Duty-Cycle hat eine Auflösung von 8bit, also lassen sich mit dem Arduino-Befehl `analogWrite(PWM-pin,Wert)` die Werte 0...255 verarbeiten. Also muss durch die Routine der Motorbestomung sichergestellt werden, dass der Stellgrad bei negativem Wert die H-Brücke anders bestromt als bei positivem Wert und dessen Betrag bei 255 begrenzt wird.

## 5.3 Anmerkung zur praktischen Umsetzung

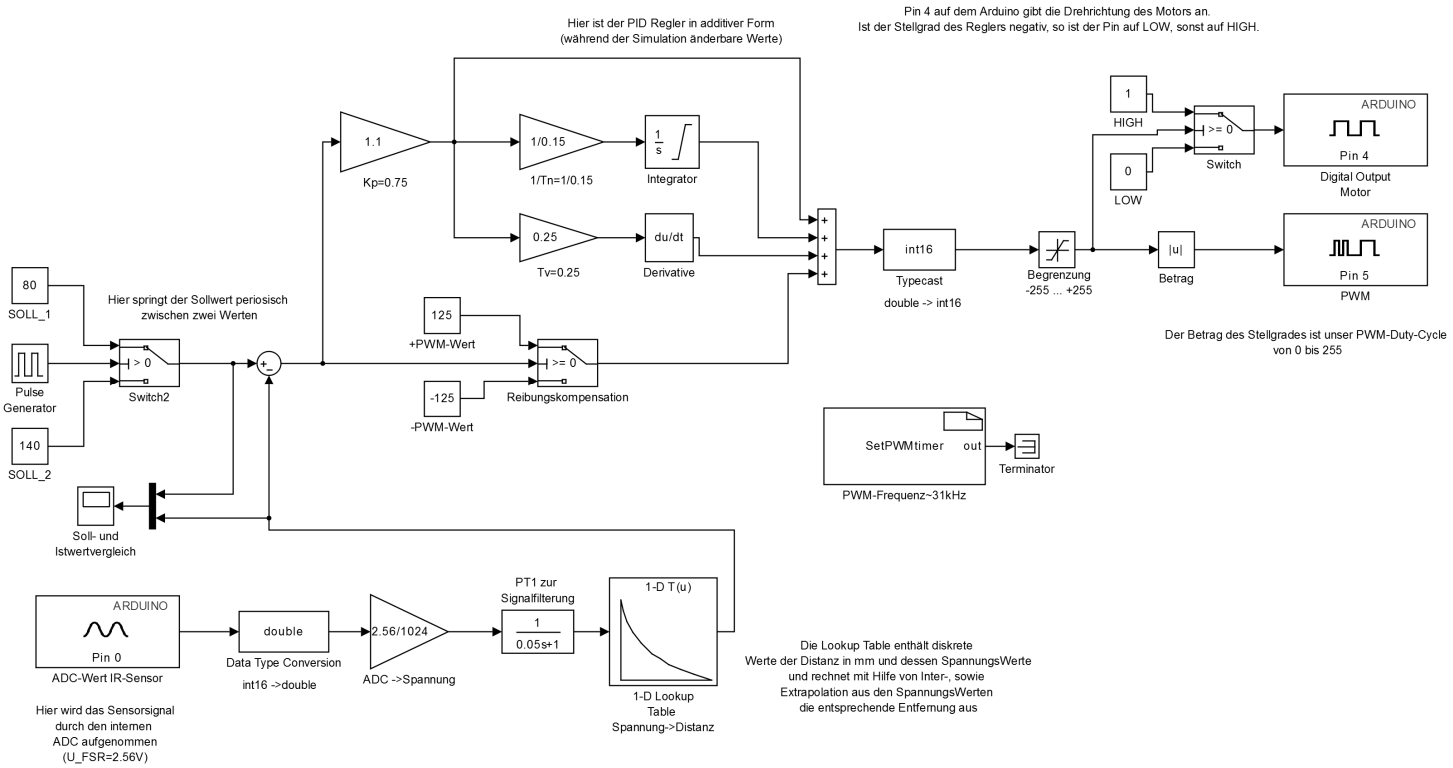
Der Regleralgorithmus wurde komplett mit Fließkommazahlen berechnet. Das liegt daran, dass das rapid-prototyping-Simulink-Modell ebenfalls mit floats rechnet und die Lookuptable mit floats rechnet. Dies ist auf einer kleinen Maschine wie einem Mikrocontroller keine praxistaugliche Lösung, da dessen Ressourcen stark begrenzt sind und die Berechnung einer neuen Stellgröße mit diesem Datentyp viel zu lange dauert. Außerdem sind die Messdaten des Infrarotsensors und des Drehencoders, sowie dem Stellwert der H-Brücke integer-Werte. Somit wäre es wesentlich effizienter, die Berechnung des Stellgrades ebenfalls mit integer-Werten durch zu führen. Da der Versuchsaufbau nichts anderes macht, als zu regeln, lässt sich dennoch eine Frequenz von $100Hz$ erreichen, was für diese träge mechanische Regelstrecke ausreicht. Ist die Regelung allerdings Teil eines komplexeren Systems, muss eine effizientere Methode verwendet werden. Eine Methode wäre, die obere Reglergleichung zu vereinfachen:

$$y(kT) = K_p \left( e(kT) + \frac{1}{T_a} * \sum e(kT) + T_v * (e(kT) - e((k-1)T) \right)$$

Außerdem ist es auch nicht nötig, im Mikrocontrollerprogramm mit korrekten Einheiten zu arbeiten. Man rechnet einfach mit den direkt aufgenommenen Messdaten, statt eine Umwandlung vor zu nehmen. Möchte man dann einen Sollwert in einer Einheit wie z.B. $\frac{mm}{s}$ einstellen, so muss man außerhalb des Programms eine Umrechnung durchführen und dem Mikrocontroller diesen Wert mitteilen.

## 5.4 Simulink-Modell

Dem folgenden Simulink-Modell ist das C-Programm nachempfunden. Die dort ermittelten Reglerparameter funktionieren genau so im selbst geschriebenen Programm.


Hier ist der PID Regler in additiver Form
(während der Simulation änderbare Werte)
Pin 4 auf dem Arduino gibt die Drehrichtung des Motors an.
Ist der Stellgrad des Reglers negativ, so ist der Pin auf LOW, sonst auf HIGH.
1.1
Kp=0.75
1/0.15
1/Tn=1/0.15
1/s
Integrator
0.25
Tv=0.25
du/dt
Derivative
1
HIGH
0
LOW
>= 0
Switch
ARDUINO
Pin 4
Digital Output
Motor
int16
Typecast
double -> int16
Begrenzung
-255 ... +255
|u|
Betrag
ARDUINO
Pin 5
PWM
Der Betrag des Stellgrades ist unser PWM-Duty-Cycle
von 0 bis 255
80
SOLL_1
Hier springt der Sollwert periosisch
zwischen zwei Werten
Pulse
Generator
> 0
Switch2
140
SOLL_2
125
+PWM-Wert
-125
-PWM-Wert
>= 0
Reibungskompensation
SetPWMtimer
out
Terminator
PWM-Frequenz~31kHz
Soll- und
Istwertvergleich
ARDUINO
Pin 0
ADC-Wert IR-Sensor
Hier wird das Sensorsignal
durch den internen
ADC aufgenommen
(U_FSR=2.56V)
double
Data Type Conversion
int16 ->double
2.56/1024
ADC ->Spannung
PT1 zur
Signalfilterung
1
0.05s+1
1-D T(u)
1-D Lookup
Table
Spannung->Distanz
Die Lookup Table enthält diskrete
Werte der Distanz in mm und dessen SpannungsWerte
und rechnet mit Hilfe von Inter-, sowie
Extrapolation aus den SpannungsWerten
die entsprechende Entfernung aus

# Kapitel 6

# Python Interface

## 6.1 Warum Python?

Um den Regler bequem einstellen zu können sowie die Messdaten in Echtzeit in einem Graphen dar zu stellen, wird ein kleines Python-Proramm erstellt, dass als Interface für den regler dienen soll. Die Wahl fiel auf Python, da sie eine sehr leicht zu erlernende Programmiersprache ist und nichts kostet, da Open Source. Somit lassen sich auf einem System geschriebene Programme auf vielen Sytemen laufen, ohne Lizenzgebühren für einen Interpreter oder eine Runtime-Umgebung für jedes System neu zu kaufen. Die Anwendung soll sowohl auf einem PC, als auch auf einem eingebetteten System, wie dem Raspberry Pi oder dem Beaglebone laufen. Außerdem gibt es mit den Erweiterungen "Numpy", "Scipy" und "Matplotlib" sehr mächtige numerische sowie wissenschaftliche Funktionen, die die häufigsten MATLAB-Funktionen abdecken und von dessen Syntax her nahezu identisch sind.

## 6.2 Oberfläche

Die Grafische Oberfläche des Interfaces wurde mit dem Qt Designer erstellt. Mit diesem freien Editor lassen sich GUIs nach dem Prinzip WYSIWYG(what you see is what you get) erstellen. Er ist bei der Installation des Qt Frameworks mit enthalten. Da es sich bei Qt eigentlich um ein C++ Framework handelt, muss die mit dem Designer erstellte *.ui-Datei mit Hilfe eines Übersetzers in ein Python Skript konvertiert werden. Das Übersetzerprogramm heißt "pyuic4" und erstellt ein Skript mit einer Window-Klasse, die recht einfach in ein Hauptprogramm eingebunden werden kann. Neben dem Python-Interpreter sind folgende Zusatzbibliotheken nötig:

- PyQt4
- PySerial
- PyQtGraph
- numpy
- scipy und matplotlib(optional)

## 6.3 Serielle Kommunikation

Für die Kommunikation über die serielle Schnittstelle mit dem Arduino wird die "Serial"-Bibliothek verwendet. Außerdem kommt die "Threading"-Bibliothek zum Einsatz, mit der in einem seperaten Thread permanent den Eingang der Schnittstelle abgefragt wird und ein Buffer dementsprchend gefüllt wird. Dieser Buffer wird in regelmäßigen Abständen ausgelesen, geleert und dessen Inhalt in einem Array gespeichert, dass als Grundlage für den live-Plot dient. Die serielle Kommunikation sollte

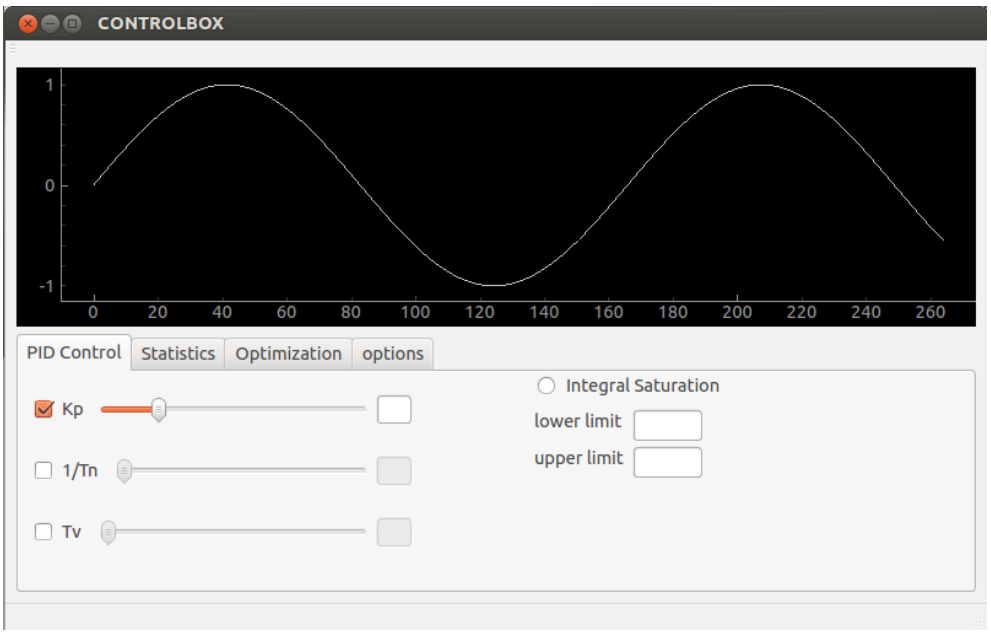



Abbildung 6.1: Aussehen des Interfaces

mit Zeitstempel und folgendem 16 bit Integer-Wert vom Mokrocontroller zum Python-Interface erfolgen. Außerdem soll vom Interface einzelne character zur Erkennung mit folgenem Eintsellwert als Integer zum Microcontroller gesendet werden, um Einstellungen der Ausgabe oder der Parameter durch das Interface vorgenommen werden können. Diese Schritte sind aus Mangel an Zeit und fehlender Erfahrung im Umgang mit dem Signal-Slot-System, mit dem sich Events mit GUI-Elementen verknüpfen lassen nicht implementiert. Eine einfache Plotanwendung, die Strings gefolgt von einem `linefeed` als `float` interpretiert und plottet funktioniert hingegen und liegt den Anlagen ebenfalls bei.

# Kapitel 7

# Quellen


- www.arduino.cc
- http://www.rn-wissen.de/index.php/Regelungstechnik
- PyQt und PySide - Peter Bouda ISBN 978-3-941841-50-5
- http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial
- Datenblatt des Atmega2560
- MATLAB/Simulink Hilfe