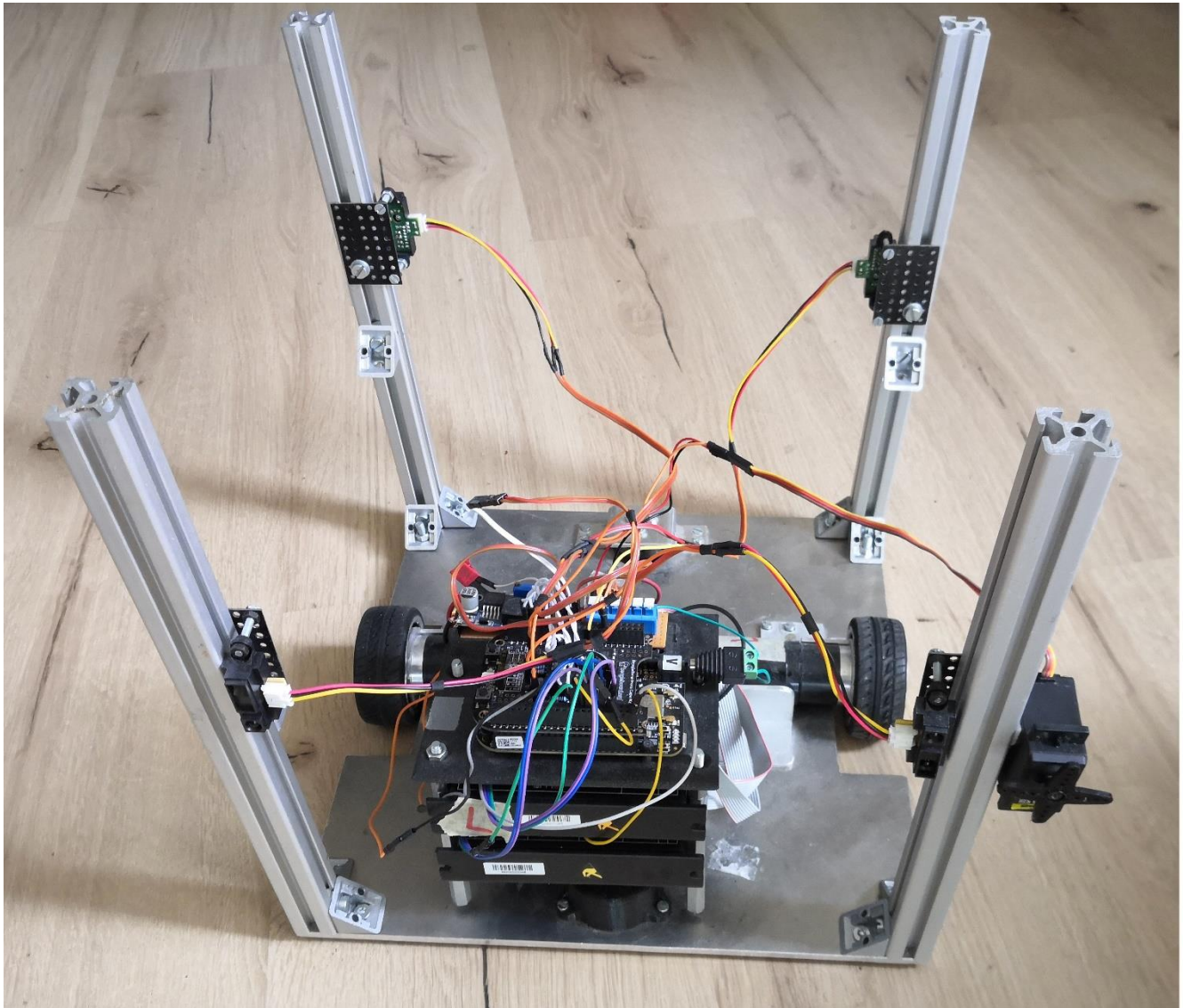


Weiterentwicklung der Eurobot Fahrbasis



Bearbeiter: Christopher Sauer

Betreuer: Prof. Dr.-Ing. habil. Thomas Glotzbach

Inhaltsverzeichnis

1. Einleitung	1
1.1 Beschreibung des Systems	1
1.2 Überblick der vorherigen Projektarbeit	1
1.3 Entstandene Probleme	2
1.4 Ziel dieser Projektarbeit	2
2. Übertragung des Linux Systems in den EMMC-Speicher	2
3. Programmierung Teil 1	3
3.1 Einrichten der Entwicklungsumgebung	3
3.2 Bisherige Programmierung	5
3.3 Veränderungen des Codes	5
3.4 Ausbau der Basisklassen (GPIO, PWM)	6
4. ADC Schutzschaltung	9
5. Regler	11
5.1 Controllereinstellung	11
5.2 Motormessung	13
5.3 Softwareregler	18
5.3.1 Streckenmessung	19
5.5 Messtechnische Bestimmung des Übersetzungsverhältnis der ADCs	21
5.6 Messtechnische Bestimmung des Radabstandes	25
5.7 Fahren bei erhöhter Geschwindigkeit	25
6. Programmierung Teil 2	26
6.1 Übersicht der Programmierung	26
6.1.1 UML - Diagramm	27
6.2 Beschreibung der Klassen & ihrer Funktionalitäten	29
6.2.1 PWM	29
6.2.2 AnalogInput	29
6.2.3 GPIO	30
6.2.4 Timer	30
6.2.5 Motor	30
6.2.6 Servo	31
6.2.7 InfraredSensor	31
6.2.8 Odometry	33
6.2.9 Ecu	33
7. Fazit	36



8.1	Abbildungsverzeichnis	37
8.2	Tabellenverzeichnis	38
9	Konzept.....	39
9.1	Individueller Teil.....	39
9.1.1	Beschreibung des Themas.....	39
9.2	Allgemeiner Teil	41
9.2.1	Dokumentation	41
9.2.2	Durchführung und Abgabe.....	41
9.2.3	Bewertung.....	42
9.3	Meilensteine	42

1. Einleitung

Diese Projektarbeit wurde auf Grundlage einer vorherigen Projektarbeit ausgearbeitet. Um an dieser weiterarbeiten zu können sind folgende Kenntnisse erforderlich gewesen:

- Umgang mit Linux Betriebssystemen
- Aufbau von Makefiles
- Programmiersprache C++ (Klassenprogrammierung)

1.1 Beschreibung des Systems

Die verwendeten Maxon Motoren der Fahrbasis werden über die Maxon Servokontroller angesteuert. Diese werden mit dem Beaglebone Black verbunden. Dabei gibt es einen Pin zur Freigabe, eine zur Drehrichtung und ein PWM-Eingang.

1.2 Überblick der vorherigen Projektarbeit

Es empfiehlt sich die Dokumentation „Entwicklung der Eurobot Fahrbasis“ durchzulesen, um den weiteren Ablauf dieser Projektarbeit besser verstehen zu können.

Beschreibung der Ausgangslage:

Die Fahrbasis wurde schon so weit entwickelt, dass der Roboter Fahrbefehle entgegennehmen konnte und diese auch umsetzte. Dazu gehörten die Befehle Forward, Backward, Left, Right.

Der Codeumfang beinhaltete zu diesem Zeitpunkt die Basisklassen GPIO, PWM. Die Klassen Motor und ECU wurden mit Hilfe einer Statemachine¹ umgesetzt. Die entworfene Statemachine wurde dabei von einem erfahrenen Programmierer übernommen. Eine genauere Beschreibung kann [hier](#) entnommen werden.

¹<https://www.codeproject.com/Articles/1087619/State-Machine-Design-in-Cplusplus-2>

1.3 Entstandene Probleme

Der übermittelte Parameter der zu fahrenden Wegstrecke wurde um etliche Zentimeter verfehlt. Ein Fahren auf bestimmte Positionen auf dem Spielfeld war noch nicht möglich.

Ebenfalls war ein reproduzierbarer Fehler festgestellt worden. Nachdem mehrere Fahrbefehle dem Roboter übergeben wurden, hat sich eines der Motoren nicht mehr gedreht.

Ein etwas kleineres Problem bestand in dem Bootvorgang des BeagleBones. Die verwendete Distribution (Debian-Linux) wurde auf eine SD-Karte gespielt und in den Slot des Beaglebones gesteckt. Um diese zu laden, musste beim Hochfahren ein Button gedrückt werden. Wurde dies nicht getan, konnten die PWM-Chips des Kernels nicht geladen werden.

1.4 Ziel dieser Projektarbeit

Ziel des Projekts war es, die eben beschriebenen Fehler, zu beheben. Ebenfalls müssen diverse Motortypen (Servo- und Schrittmotoren) angesteuert werden, um entsprechende Mechaniken steuern zu können. Diese werden genutzt, um Aufgaben auf dem Spielfeld durchzuführen. Zu den Aufgaben gehören unter anderem: Aufheben der Cups, Ermittlung von Norden und Süden sowie Flagge hissen.

Ein anderes Problem bestand in den ADCs des Beaglebones. In diesem wurde hardwaretechnisch keine Schutzschaltung integriert. Die maximale Spannung, die die ADCs abkönnen liegt bei 1,8V.

Damit diese Spannung nicht überschritten wird, wurde eine Schutzschaltung gelötet, die die Eingänge von zu hohen Spannungen schützt.

2. Übertragung des Linux Systems in den EMMC-Speicher

Zunächst muss ein passendes Image auf die SD-Karte geflasht werden. Daraufhin wird die SD-Karte in den Beaglebone gesteckt und dieser über die SD-Karte gebootet.

Dabei sollte man wie folgt vorgehen:

1. Drücken des S2-Buttons und anschließend Strom zuführen (Button nicht loslassen!)
2. Sobald alle LEDs anfangen gleichzeitig zu blinken, kann der Button losgelassen werden
3. Der Beaglebone wird nun von der SD-Karte gebootet

Nachdem der Beaglebone erfolgreich von der SD-Karte gebootet wurde, muss nun das auf der SD-Karte liegende Betriebssystem in den EMMC Speicher übertragen werden.

Dafür muss ein Skript mit folgendem Befehl ausgeführt werden¹:

```
sudo bash /opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh
```

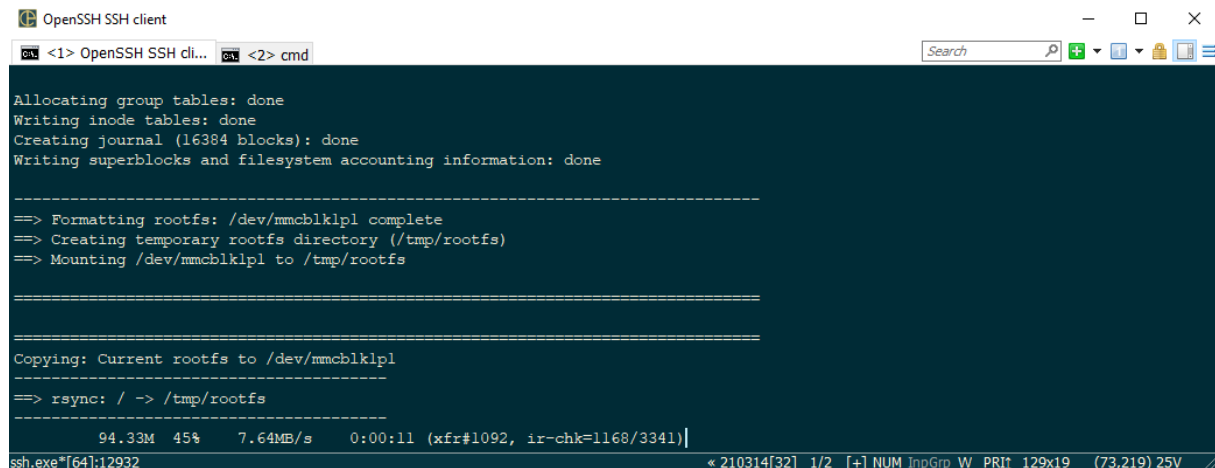
¹https://wiki.fhem.de/wiki/BeagleBone_Black

Das auf der SD-Karte liegende Betriebssystem wird daraufhin in den internen Speicher übertragen. Die SD-Karte wird danach nicht mehr benötigt.

Bei dem zurzeit benutzten Betriebssystem handelt es sich um „Debian Image 2019-08-03“.

Diese kann mit folgendem Befehl nachgesehen werden:

```
cat /etc/dogtag
```



```

OpenSSH SSH client
<1> OpenSSH SSH cli... <2> cmd

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

=====
=> Formatting rootfs: /dev/mmcblkpl complete
=> Creating temporary rootfs directory (/tmp/rootfs)
=> Mounting /dev/mmcblkpl to /tmp/rootfs

=====

Copying: Current rootfs to /dev/mmcblkpl
=====
=> rsync: / -> /tmp/rootfs

94.33M 45% 7.64MB/s 0:00:11 (xfr#1092, ir-chk=1168/3341)
ssh.exe*[64]:12932 < 210314[32] 1/2 [+] NUM InpGrp W PRI 129x19 (73,219) 25V

```

Abb.1: Übertragung in den eMMC Speicher

3. Programmierung Teil 1

3.1 Einrichten der Entwicklungsumgebung

Damit von einem Windows Rechner aus, der Beaglebone programmiert werden kann, wird folgende Software empfohlen: ConEmu¹, Visual Studio Code², WinSCP⁴

ConEmu ist ein Open-Source-Terminalemulator und eine bequeme alternative zur Windows Power Shell. Hier ist es möglich mehrere Terminals in einem zu vereinigen. Das bietet mit dem Umgang des BeagleBones viele Vorteile, da bei der Entwicklung des Programmcodes mehrere Files in unterschiedlichen Ordnern des BeagleBones überwacht werden müssen.

Visual Studio Code ist ein Quelltexteditor welches viele verschiedenen Programmiersprachen unterstützt. In diesem Projekt wurde überwiegend mit der Programmiersprache C++ gearbeitet.

Hier empfiehlt es sich die C/C++ Extension³ herunterzuladen. Diese unterstützt IntelliSense und bietet einen erheblichen Vorteil beim Programmieren. IntelliSense wird zur automatischen Vervollständigung beim Bearbeiten des Quellcodes genutzt.

Bei WinSCP handelt es sich um eine freie SFTP- und FTP-Client-Software für Windows. Diese kann Dateien zwischen einem lokalen und entfernten Computer mit diversen Übertragungsprotokollen kopieren.

¹<https://conemu.github.io/en>

²<https://code.visualstudio.com>

³<https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>

⁴<https://winscp.net/eng/docs/lang:de>

Die meisten Netzwerkrouter arbeiten mit einem DHCP-Server. Damit sich die IP-Adresse des Beaglebones nach einer gewissen Zeit nicht mehr ändert, wird eine statische IP-Adresse über den Router eingestellt.

Für die Fritzbox¹ muss man dafür nur unter Heimnetz->Netzwerk den gewünschten Teilnehmer raussuchen und durch Klicken der Bearbeiten-Schaltfläche, die gewünschte Option aktivieren.

Zunächst wird eine Verbindung über WinSCP mit dem eigenen Rechner und dem BeagleBone hergestellt. Dazu öffnet man das Programm und erstellt ein neues Verbindungsziel.

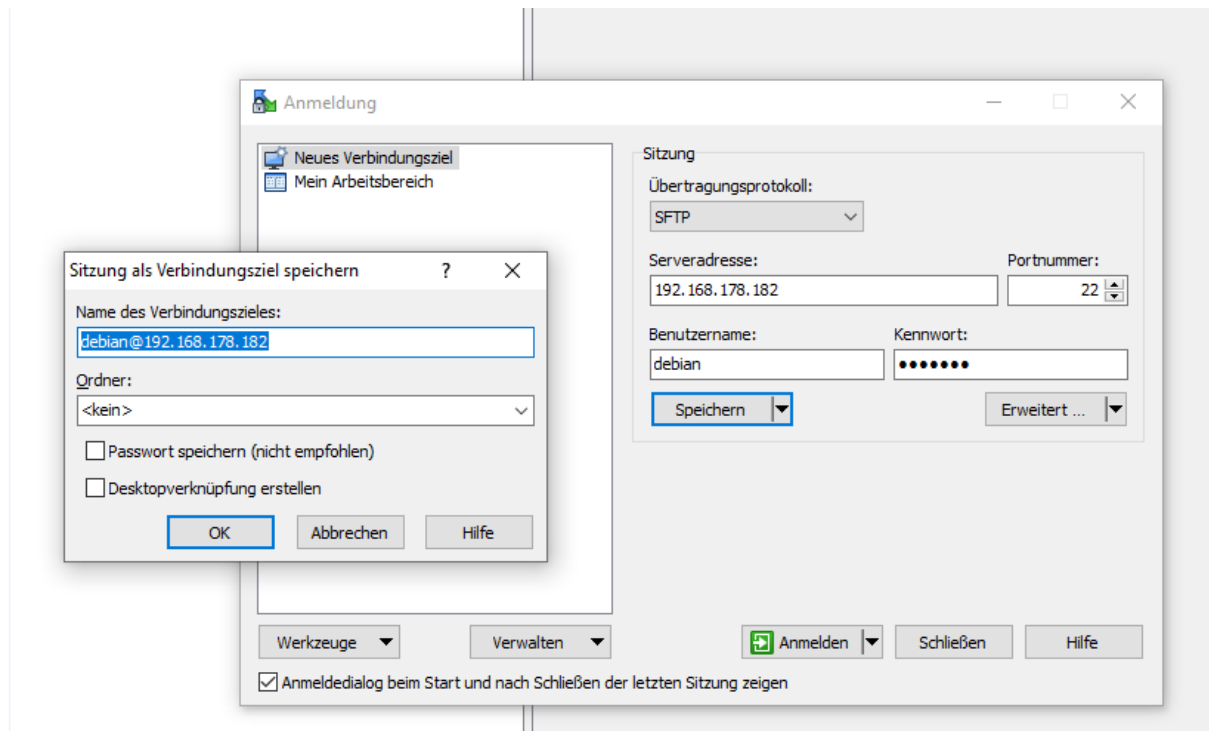


Abb.3.1: Einstellen des Verbindungsziels in WinSCP

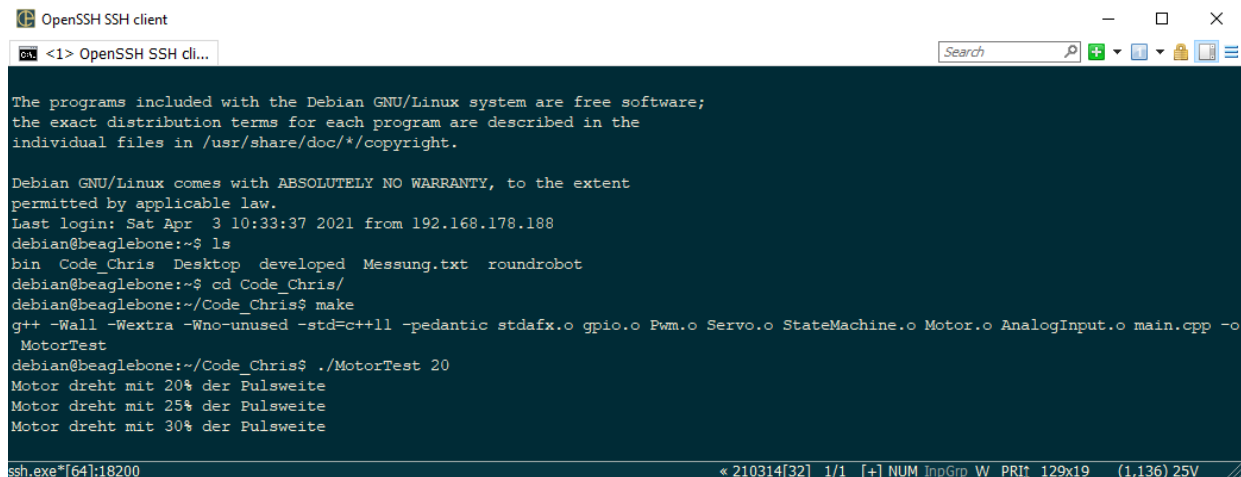
Um ein ständiges Eingeben der Adressdaten zu vermeiden, sollten diese gespeichert werden. Nun besteht die Möglichkeit einzelne Dateien oder ganze Ordner auf bzw. vom BeagleBone zu übertragen. Ein sehr angenehmes Feature ist, dass entfernte Ordner aktuell gehalten werden können.

Dazu wählt man die jeweiligen Ordner, welche aktuell gehalten werden soll, aus und bestätigt.

Wird nun über Visual Studio Code etwas an dem Code auf dem PC verändert und die Files gespeichert, werden diese automatisch über WinSCP auf den Beaglebone übertragen.

Anschließend kann man sich über die PowerShell oder conEmu auf den Beaglebone via SSH einloggen und die Makefile in dem jeweiligen Ordner mit make ausführen. Der Code wird anschließend kompiliert und kann getestet werden.

¹https://avm.de/service/fritzbox/fritzbox-7590/wissensdatenbank/publication/show/201_Netzwerkgerat-immer-die-gleiche-IP-Adresse-von-FRITZ-Box-zuweisen-lassen/



```

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Apr  3 10:33:37 2021 from 192.168.178.188
debian@beaglebone:~$ ls
bin  Code_Chris  Desktop  developed  Messung.txt  roundrobot
debian@beaglebone:~$ cd Code_Chris/
debian@beaglebone:~/Code_Chris$ make
g++ -Wall -Wextra -Wno-unused -std=c++11 -pedantic stdafx.o gpio.o Pwm.o Servo.o StateMachine.o Motor.o AnalogInput.o main.cpp -o
MotorTest
debian@beaglebone:~/Code_Chris$ ./MotorTest 20
Motor dreht mit 20% der Pulsweite
Motor dreht mit 25% der Pulsweite
Motor dreht mit 30% der Pulsweite
  
```

Abb.3.2: Ausführen der Makefile und anschließendes testen

3.2 Bisherige Programmierung

Bestandteil der bisherigen Programmierungen waren die Basisklassen zum Ansteuern der GPIOs und der PWM. Hierbei wurden Veränderungen vorgenommen, um den Code effizienter zu gestalten. Die Klassen des Motors sowie des Controllers wurden mit Hilfe einer StateMachine umgesetzt.

Die StateMachine wurde nicht selbst entwickelt, sondern aus dem Internet übernommen.

Ebenfalls wurde RoboDoc in den Code implementiert, um den Nächsten, die mit dem Code arbeiten wollen, eine genaue Übersicht über die Funktionen des Codes geben zu können.

Die Klasse easylogging++ sollte eine Unterstützung beim Entwickeln des Codes darstellen. Diese sorgte dafür, dass jeglicher Methodenaufruf auf dem Terminal ausgegeben wurde.

3.3 Veränderungen des Codes

Die Klasse easylogging++ und alle damit zusammenhängende Funktionalitäten wurden aus dem Codeumfang entfernt. Hier kam es beim Programmieren zu unzähligen Compilerfehlermeldungen und stellte somit keine Hilfe bei der Weiterentwicklung des Codes da.

Die Header DataTypes.h wurde gelöscht und mit der Standardbibliothek stdint.h ersetzt.

Die Implementierung der StateMachine wurde entfernt, da dieses Projekt auch Einsteigern Umgänglich gemacht werden soll. Ebenfalls darf für den Eurobot nur Code zum Einsatz kommen, welcher auch selbst entwickelt wurde.

3.4 Ausbau der Basisklassen (GPIO, PWM)

GPIO

Die Methoden Export und Unexport wurden aus dem Codeumfang entfernt. Alle zur Verfügung stehenden GPIOs werden beim Bootvorgang, des Beaglebones, standardmäßig vom System exportiert und machen die Funktionalität somit überflüssig.

PWM

Beim Benutzen bestimmter Pins als PWM, besteht ein Problem darin, dass die jeweiligen Pins über das System konfiguriert werden müssen. Dies geschah bis dahin mit einem selbst geschriebenen Bash-Skript. Dieser führte lediglich den Befehl

```
config-pin P8_19 pwm
```

aus.

Diese Anweisung wurde in den Codeumfang implementiert.

Auf dem Beaglebone sind mehrere Pins zu finden, welche als PWM-Pin genutzt werden können.

Dabei ist jedem Pin genau einem PWM-Chip zugeordnet. Ein PWM-Chip besteht wiederum aus 2 Submodulen, die sich ein Periodenregister teilen. Beispielsweise ist der PWM-Chip 7 in pwm-7:0 und pwm-7:1 untergliedert. Hier ist es nur möglich eine Periode für den jeweiligen PWM-Chip einzustellen. Die jeweiligen PWMs der einzelnen Chips sind allerdings mit unterschiedlichen Vergleichsregister ausgestattet und können somit unterschiedlich eingestellt werden.

SPIO_SLCK	P9_22	pwm-1:0	pwmchip1
SPIO_DO	P9_21	pwm-1:1	
EHRPWM1A	P9_14	pwm-4:0	pwmchip4
EHRPWM1B	P9_16	pwm-4:1	
EHRPWM2A	P8_19	pwm-7:0	pwmchip7
EHRPWM2B	P8_13	pwm-7:1	

Tab. 3.1: Lookup-Tabelle der genutzten Chips und den zugeordneten Pins

Die als PWM-Pin konfigurierten Pins müssen daraufhin exportiert werden, um die PWM betreiben zu können.

```
debian@beaglebone:/sys/class/pwm$ ls
pwmchip0 pwmchip1 pwmchip3 pwmchip4 pwmchip6 pwmchip7
```

Abb.3.3: Ordnerpfad zu den jeweiligen pwmchips

```
debian@beaglebone:/sys/class/pwm/pwmchip7$ ls
device export npwm power subsystem uevent unexport
```

Abb.3.4: Ordnerinhalt eines pwmchip

Eine PWM wird mit Schreiben der Value 0 oder 1 in den pwmchip Ordner in die File export exportiert.

```
echo 0 > /sys/class/pwm/pwmchip7/export
```

In diesem Beispiel wurde die pwm-7:0 exportiert, welche auf dem Pin P8_19 ausgegeben wird.

```
debian@beaglebone:/sys/class/pwm/pwmchip7$ echo 0 > export
debian@beaglebone:/sys/class/pwm/pwmchip7$ ls
device export npwm power pwm-7:0 subsystem uevent unexport
```

Abb.3.5: exportieren der PWM am Beispiel pwm-7:0

Um diese Konfigurationseinstellungen nicht immer manuell durchführen zu müssen, wurde die Befehlsabarbeitung in den Code implementiert.

Dabei ist darauf zu achten, dass nicht jeder Pwmchip durch den Code verfügbar ist. Möchte man weitere PWM Signale nutzen, müssen die zugehörigen Pins auf dem Beaglebone ausfindig gemacht und in den Code nachträglich hinzugefügt werden. Eine Auflistung der einzelnen Pins und ihren Einstellungen kann [hier](#) gefunden werden.

```
debian@beaglebone:/sys/class/pwm/pwmchip7/pwm-7:0$ ls
capture device duty_cycle enable period polarity power subsystem uevent
```

Abb.3.6: Ordnerinhalt der fertig exportierten PWM

Die einstellbaren Parameter für die Periode und der Pulsweite werden in Nanosekunden übergeben. Eine genaue Beschreibung, zur Einstellung der PWM, findet sich auf der Seite von TexasInstrument².

Das Exportieren der Chips, geschieht mit dem Übergabeparameter `-export`. **Dieser Befehl muss einmalig nach jeden Bootvorgang ausgeführt werden.**

¹<https://vadl.github.io/beagleboneblack/2016/07/29/setting-up-bbb-gpio>

²https://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/Foundational_Components/Kernel/Kernel_Drivers/Display/PWM.html

Pin-Problem (gelöst):

Anfänglich konnte der Pin P8_13 über die config-pin Anweisung nicht konfiguriert werden. Der zugehörige Ordnerpfad existierte nicht.

Die config-pin Anweisung greift in Abhängigkeit des angegebenen Pins auf einen Ordner zu.

Im Beispiel P8_13:

```
/sys/devices/platform/ocp/ocp:P8_13_pinmux
```

Dieser Ordner konnte vom System, beim Bootvorgang, nicht automatisch erstellt werden. Um Veränderungen am Bootvorgang durchführen zu können, kann auf eine Text-File zugegriffen werden, welche sich unter `/boot/uEnv.txt` befindet.

Eine Bearbeitung der Text-File kann nur dann stattfinden, wenn der Beaglebone über die USB-Schnittstelle verbunden wird. Der Zugriff erfolgt über SSH mit der IP 192.168.7.2.

Eine Veränderung der File kann weitreichende Folgen haben. Hier sollte man genau wissen für was welche Option¹ steht.

Im vorliegenden Fall musste die Option `disable_uboot_overlay_addr3=1` auskommentiert werden, da `uboot`² mit der config-pin Anweisung in Verbindung steht.

```
enable_uboot_cape_universal=1
###
###Debug: disable uboot autoload of Cape
#disable_uboot_overlay_addr0=1
#disable_uboot_overlay_addr1=1
#disable_uboot_overlay_addr2=1
disable_uboot_overlay_addr3=1
```

Abb.3.7: Auskommentierte Option in `/boot/uEnv.txt`

Der fehlende Ordner konnte nach hinzufügen dieser Funktion ausfindig gemacht werden.

Eine Auflistung der zur Verfügung stehenden Pins, für die config-Pin Anweisung, wird mit:

```
ls /sys/devices/platform/ocp | grep pinmux
```

durchgeführt.

¹https://elinux.org/Beagleboard:BeagleBoneBlack_Debian#4.19.x-ti

²<https://github.com/cdsteinkuehler/beaglebone-universal-io>

4. ADC Schutzschaltung

Die maximale Spannung der ADCs ist auf 1,8V begrenzt. Ein Überschreiten dieser Spannung, würde den ADC zerstören.

Damit keine Spannung den Wert von 1,8V überschreitet, wurde eine Schutzschaltung auf einem Prototype Cape, für den Beaglebone, aufgelötet.

Signalpegel, welche höher als 1,8V gehen, werden von der Schutzschaltung bei abgeschnitten. Bei den verwendeten Operationsverstärkern handelt es sich um Rail to Rail Amps, welche als Spannungsfolger agieren.

6 ADC-Eingänge sind in Benutzung: 4 x Infrarotsensoren, 2 x Encoderspannungen zur Drehzahlmessung

Bei den verwendeten Infrarotsensoren handelt es sich um die GP2Y0A21YK0F¹ von Sharp. Diese benötigen eine Versorgungsspannung von 4,5 V – 5,5V und geben eine maximale Spannung von 3,2V aus. Dieser Bereich wurde mittels eines Spannungsteilers auf 1,8V begrenzt.

Die Spannungen der Drehzahlsensoren werden über die Servokontroller² von Maxon ausgegeben. Der Spannungsbereich kann bei der Konfiguration der Controller über ESCON-Studio eingestellt werden.

Die abgebildete Zeichnung zeigt die ADC-Schutzschaltung³. Diese wurde von der Website des Begleitbuches „Exploring the Beaglebone“ übernommen.

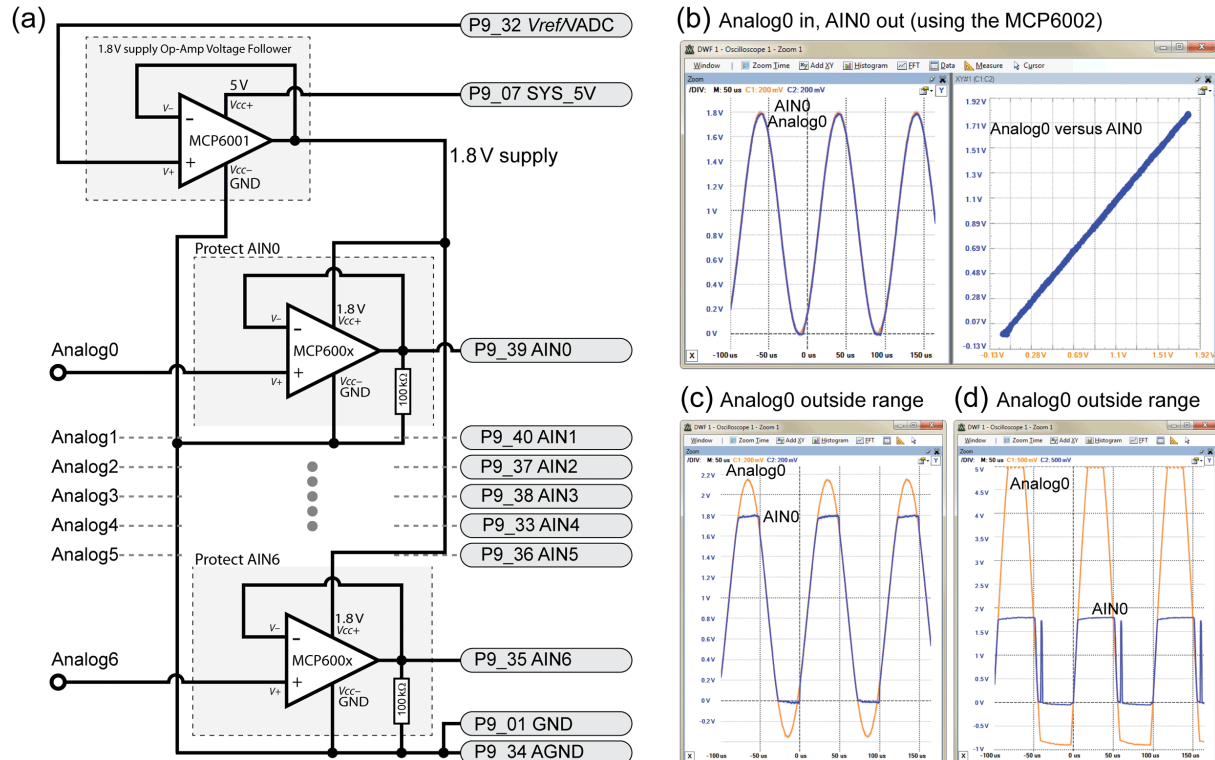


Abb.4.1: Prinzip der ADC-Schutzschaltung³

Die Eingänge der Infrarotsensoren wurden um einen Spannungsteiler erweitert, welche die maximale Ausgangsspannung zusätzlich auf 1,8V begrenzt.

¹https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf

²ESCON 50/5 Servokontroller – 409510

³<http://exploringbeaglebone.com/wp-content/uploads/2014/12/935125-c09f010-abcd.png>

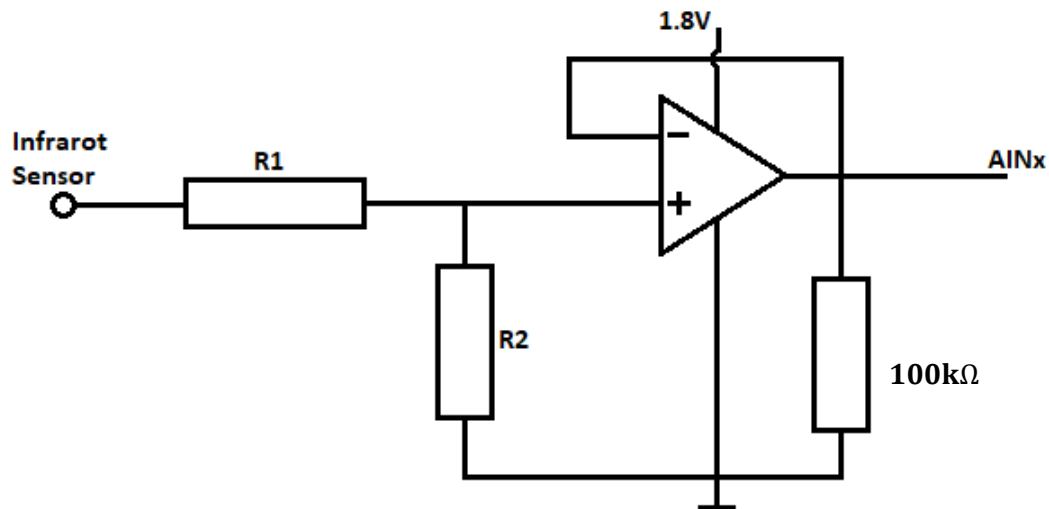


Abb.4.2: Spannungsteiler Eingang Operationsverstärker Infrarotsensoren

Der Widerstandswert für R1 beträgt $27k\Omega$ und für R2 $33k\Omega$.

Bei einer maximalen Eingangsspannung von $3.2V$ liegen, am Eingang des Operationsverstärkers, maximal $1.76V$ an.

$$U_{in} = \frac{3.25V \cdot 33k\Omega}{(33+27)k\Omega} = 1.76V \quad Gl.4.1$$

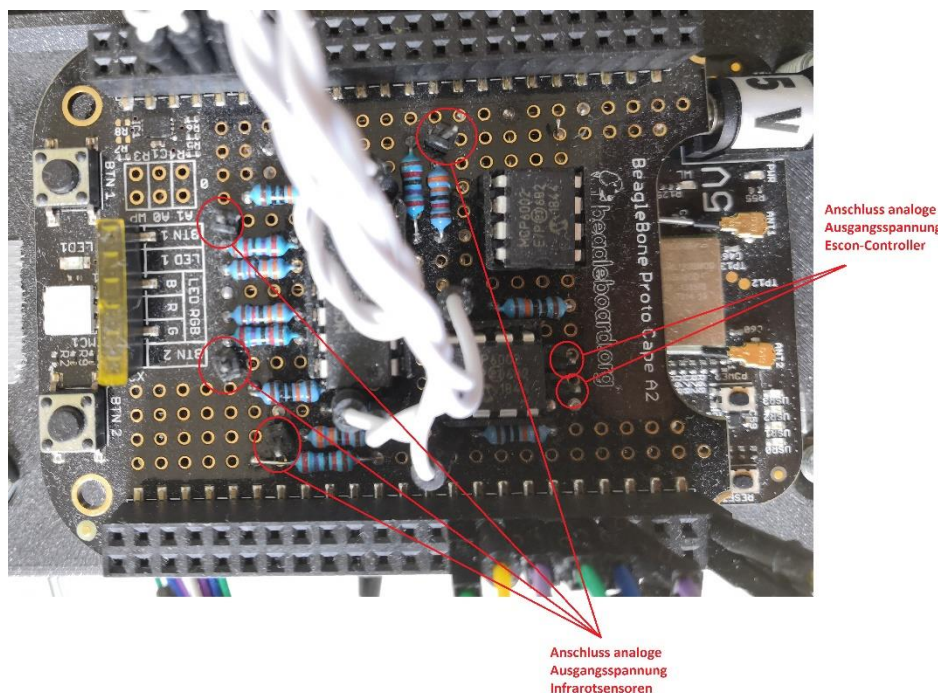


Abb.4.3: Anschluss der analogen Ausgangsspannungen an das Cape

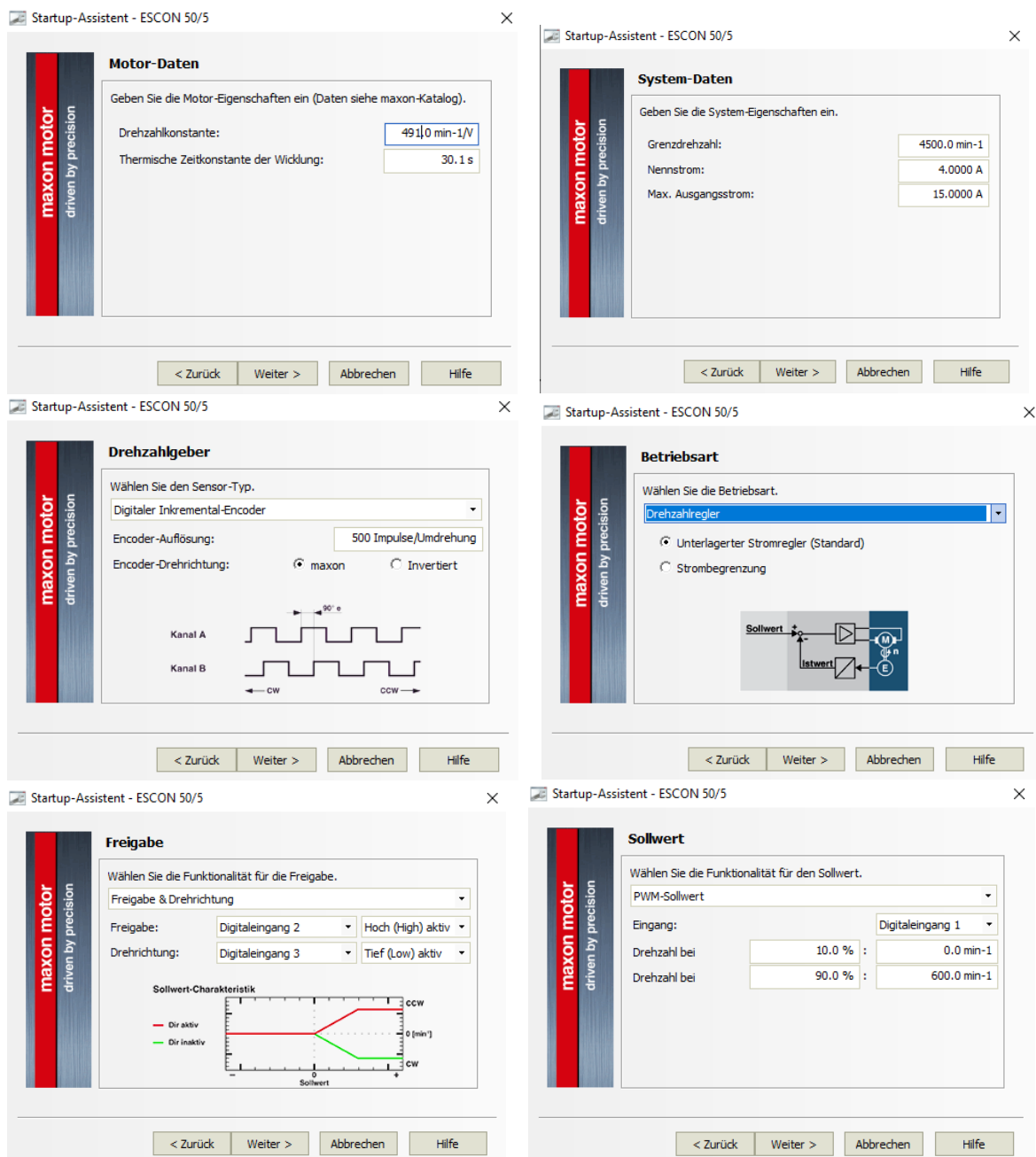
5. Regler

5.1 Controllereinstellung

Damit die Zahlenwerte der ADCs durch die anliegende Spannung mit der von den Motortreibern gemessene Drehzahl übereinstimmen, wurde eine gleichzeitige Messung beider Größen durchgeführt und über eine Mittelwertbildung zusammengeführt.

Zunächst werden die Motortreiber über die lizenzfreie Software, Escon Studio, konfiguriert. Dazu müssen die Motortreiber lediglich mit der USB-Schnittstelle des PCs verbunden und der Startup-Assistent in Escon Studio ausgewählt werden.

Bilderstrecke: Startup-Assistent zur Konfiguration der Motortreiber



The image shows a sequence of six screenshots from the 'Startup-Assistent - ESCON 50/5' software, illustrating the configuration steps for a motor driver.

- Motor-Daten:** The user enters the motor's characteristics. The 'Drehzahlkonstante' (Speed constant) is set to 49.10 min⁻¹/V, and the 'Thermische Zeitkonstante der Wicklung' (Thermal time constant of the winding) is set to 30.1 s.
- System-Daten:** The user enters the system's characteristics. The 'Grenzdrehzahl' (Limit speed) is set to 4500.0 min⁻¹, the 'Nennstrom' (Rated current) is set to 4.0000 A, and the 'Max. Ausgangsstrom' (Max. output current) is set to 15.0000 A.
- Drehzahlgeber:** The user selects the sensor type. 'Digitaler Inkremental-Encoder' is chosen. The 'Encoder-Auflösung' (Encoder resolution) is set to 500 Impulse/Umdrehung. The 'Encoder-Drehrichtung' (Encoder direction) is set to 'maxon'. A diagram shows the pulse sequences for Kanal A and Kanal B, indicating CW and CCW directions.
- Betriebsart:** The user selects the operating mode. 'Drehzahlregler' (Speed controller) is chosen. The 'Unterlagerter Stromregler (Standard)' (Underlying current controller (Standard)) is selected. A diagram shows the control loop with Sollwert (Setpoint) and Istwert (Actual value) inputs.
- Freigabe:** The user selects the functionality for the release. 'Freigabe & Drehrichtung' (Release & Direction) is chosen. The 'Freigabe' (Release) is set to 'Digitaleingang 2' (Digital input 2) and 'Hoch (high) aktiv' (High (high) active). The 'Drehrichtung' (Direction) is set to 'Digitaleingang 3' (Digital input 3) and 'Tief (Low) aktiv' (Low (Low) active). A diagram shows the 'Sollwert-Charakteristik' (Setpoint characteristic) for CW and CCW directions.
- Sollwert:** The user selects the functionality for the setpoint. 'PWM-Sollwert' (PWM setpoint) is chosen. The 'Eingang' (Input) is set to 'Digitaleingang 1' (Digital input 1). The 'Drehzahl bei 10.0 %' (Speed at 10.0 %) is set to 0.0 min⁻¹, and the 'Drehzahl bei 90.0 %' (Speed at 90.0 %) is set to 600.0 min⁻¹.

maxon motor

driven by precision

Strombegrenzung

Wählen Sie die Funktionalität für die Strombegrenzung.

Fixe Strombegrenzung

Strombegrenzung: 15.0000 A

< Zurück

Weiter >

Abbrechen

Hilfe

maxon motor

driven by precision

Digitaleingänge/-ausgänge

Wählen Sie die Funktionalität für die digitalen Ein- und Ausgänge.

Eingang/Ausgang	Funktionalität
Digitaleingang 1	PWM - Sollwert
Digitaleingang 2	Freigabe
Digitaleingang 3	Drehrichtung
Digital I/O 4	keine

< Zurück

Weiter >

Abbrechen

Hilfe

maxon motor

driven by precision

Analogeingänge

Wählen Sie die Funktionalität für die analogen Eingänge.

Eingang	Funktionalität
Analogeingang 1	keine
Analogeingang 2	keine
Potentiometer 1	keine
Potentiometer 2	keine

< Zurück

Weiter >

Abbrechen

Hilfe

maxon motor

driven by precision

Analogausgänge

Wählen Sie die Funktionalität für die analogen Ausgänge.

Ausgang	Funktionalität
Analogausgang 1	Ist-Drehzahl gemittelt
Analogausgang 2	keine

< Zurück

Weiter >

Abbrechen

Hilfe

maxon motor

driven by precision

Analogausgang 1 - Ist-Drehzahl gemittelt

Wählen Sie die Skalierung für den analogen Ausgang.

Drehzahl bei 0.950 V : 0.0 min-1

Drehzahl bei 1.750 V : 600.0 min-1

< Zurück

Weiter >

Abbrechen

Hilfe

Anschließend wird das Auto-Regler-Tuning ausgeführt:

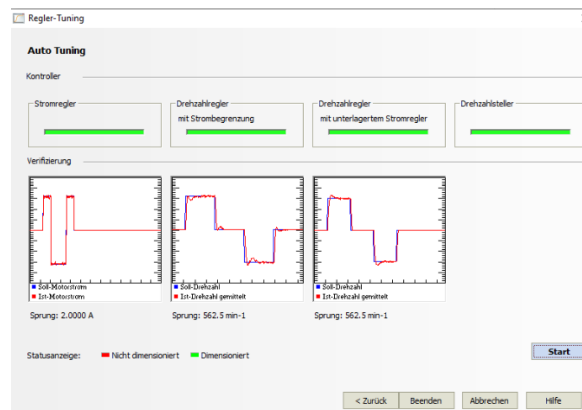


Abb.5.1: Auto-Regler-Tuning abgeschlossen

Sofern die Controller richtig konfiguriert wurden, kann nun eine Datenaufzeichnung stattfinden. Dazu muss man den Reiter Werkzeuge -> Datenaufzeichnung auswählen.

Der in diesem Versuch aufgezeichnete Wert ist der „Ist-Motordrehzahl gemittelt“.

5.2 Motormessung

Auf dem Beaglebone wurde ein Programmrahmen entwickelt, welcher den jeweilig angeschlossenen ADC des Motors 100-mal ausliest und in eine File speichert. Danach wird die Pulsweite des PWM-Moduls um 5% erhöht und die Messung erneut durchgeführt.

Damit eine möglichst Zeit genaue Messung durchgeführt werden kann, wurde der Code zum Ansteuern der Motoren über den Beaglebone wie folgt durchgeführt:

1. Schreiben der PWM-Daten in die File (Pulsweitenverhältnis, Pulsweite)
2. 1 Sekunde Warten
3. Lesen des ADCs
4. 100ms Warten

Schritt 3 und 4 werden über eine for-Schleife 100-mal wiederholt.

In Escon Studio muss dementsprechend, unter den Einstellung zur Datenaufzeichnung, eine Abtastzeit von 100ms gewählt werden.

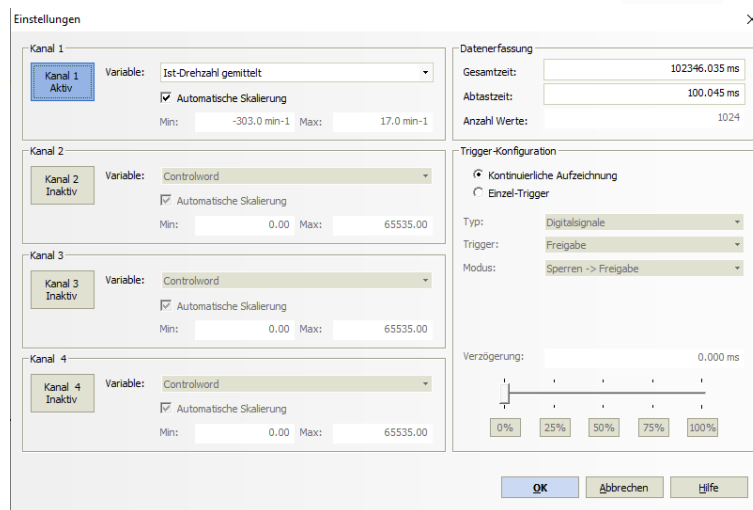


Abb.5.2: Einstellungen Datenaufzeichnung

Die Datenaufzeichnung wird in Escon-Studio durch Drücken des Start-Buttons eingeleitet. Unmittelbar danach wird das Skript über ConEmu aufgerufen.

```
./MotorTest -Param
```

-Param beschreibt das Pulsweitenverhältnis bei dem die Motoren anfangen zu drehen und wird im Bereich von 0-100 angegeben.

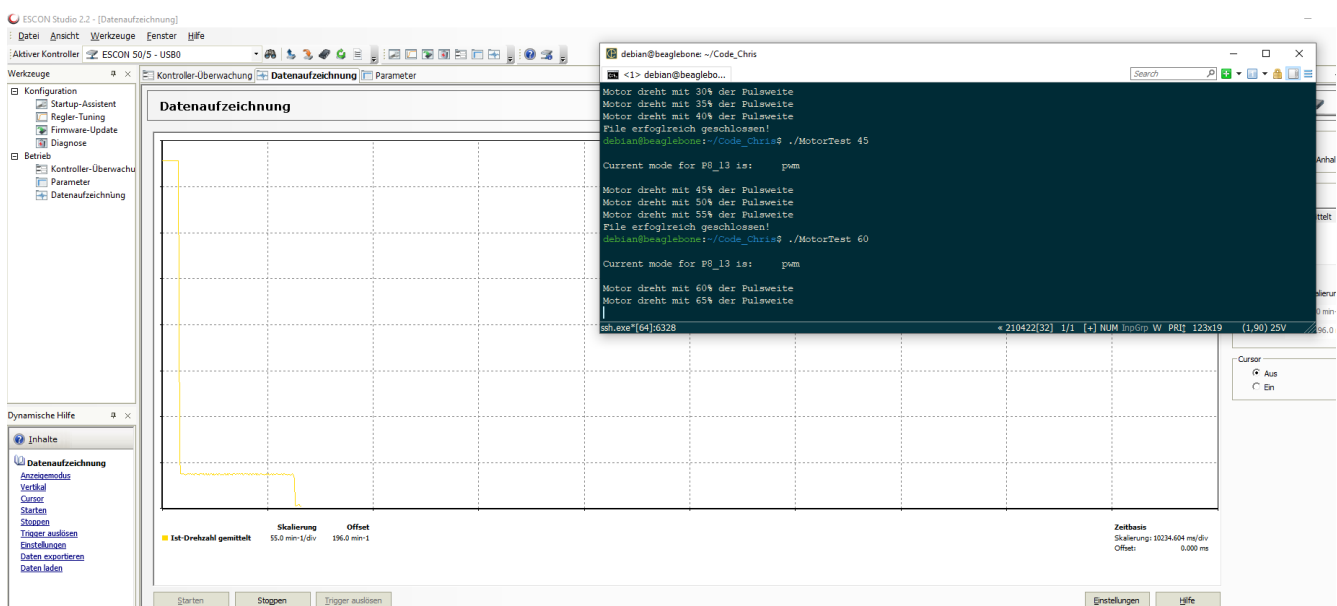


Abb.5.3: Motormessung

Sobald der Code beendet wurde, kann die Datenaufzeichnung unter Escon Studio gestoppt werden. Die aufgezeichneten Parameter lassen sich über Rechtsklick auf die Grafik -> „Aufgezeichnete Daten exportieren“ in einen beliebigen Ordner abspeichern.

Die Daten seitens von Escon Studio werden, über ein Zeitstempel versehen, in eine .csv File gespeichert. Die gemessenen Werte des ADCs wurden dabei manuell in die File hineinkopiert.

44	3301.485	0			
45	3401.53	0			
46	3501.575	-115.2	Start		
47	3601.62	-149.9			
48	3701.665	-150.1			
49	3801.71	-151			
50	3901.755	-149.3			
51	4001.8	-149.6			
52	4101.845	-150.2		Value bei: 30% Pulsweite	
53	4201.89	-149.7		DutyCycle: 600000	
54	4301.935	-150.1			
55	4401.98	-149.1		Drehzahl gemittelt	ADC gemittelt
56	4502.025	-150	1 Sekunde gewartet	-151.521	1611.58
57	4602.07	-149.8	1593		
58	4702.115	-149.2	1595		
59	4802.16	-149.6	1593		

Abb.5.4: gemessene Drehzahl; orange: hinzugefügte ADC-Werte

Allerdings bietet diese Methode keine exakt zeitlich genaue Messung. Es ist nicht sichergestellt, dass zu den angegebenen Zeitpunkten, der eingelesene ADC-Wert genau dieser Drehzahl entsprach. Um dennoch eine möglichst verlässliche Aussage über die Drehzahl und den gemessenen ADC-Wert treffen zu können, wurde eine Mittelwertbildung, beider Größen, durchgeführt und in eine Tabelle zusammengetragen.

	Motor Links			
	Forward		Backward	
Pulsweite	Motordrehzahl	ADC_AIN5	Motordrehzahl	ADC_AIN5
300000	37.908	2206.14	-37.941	1984.01
400000	75.631	2323.67	-75.588	1868.53
500000	113.331	2437.34	-113.355	1751.56
600000	151.142	2555.77	-151.263	1635.2
700000	189.409	2669.33	-189.417	1520.86
800000	227.223	2782.28	-227.139	1403.59
900000	264.942	2898.49	-264.947	1289.31
1000000	302.854	3016.54	-303.223	1174.17
1100000	340.955	3132.74	-340.923	1059.76
1200000	378.526	3248.32	-378.639	939.4
1300000	416.469	3364.69	-416.378	827.91
1400000	454.492	3478.72	-454.573	710.27
1500000	492.436	3597.9	-492.467	595.9
1600000	530.137	3711.6	-530.106	480.1
1700000	567.846	3825.85	-567.877	364.26
1800000	605.942	3941.14	-605.913	247.61

Tab.5.1: Motor Messung Links

Motor Rechts				
Pulsweite	Forward		Backward	
	Motordrehzahl	ADC_AIN4	Motordrehzahl	ADC_AIN4
300000	-37.922	1954.64	38.116	2181.12
400000	-75.805	1839.03	75.839	2295.69
500000	-113.24	1722.93	113.321	2409.29
600000	-151.521	1611.58	151.266	2528.51
700000	-189.355	1494.13	189.365	2640.33
800000	-227.071	1378.41	227.045	2759.42
900000	-264.814	1264.65	264.737	2873.93
1000000	-303.431	1149.99	302.629	2991.28
1100000	-340.885	1035.75	340.938	3105.02
1200000	-378.645	918.88	378.551	3222.87
1300000	-416.277	804.59	416.296	3336.76
1400000	-454.275	686.58	454.351	3451.21
1500000	-492.389	573.98	492.396	3568.2
1600000	-530.105	456.89	530.091	3682.99
1700000	-567.8	343.82	567.748	3798.9
1800000	-605.655	227.09	605.832	3916.8

Tab.5.2: Motor Messung Rechts

Aus der vorherigen Projektarbeit sind mathematische Formeln zur Berechnung der Spannung in den ADC-Wert gegeben.

Eine anliegende Spannung liefert demnach den Zahlenwert nach:

$$D = \frac{V_{in} \cdot (2^{12} - 1)}{V_{ref}} \quad Gl.5.2.1$$

Im Startup-Assistent wurde eine analoge Ausgangsspannung von 0.95 V bei 0 min^{-1} und 1.75V bei 600 min^{-1} eingestellt.

Der anzunehmende ausgegebene Spannungswert, kann über folgende Gleichung berechnet werden:

$$V_{in} = 0,95 \pm \frac{\Delta V}{600} \cdot n_{gemessen} \quad Gl.5.2.2$$

Motor Rechts				
Backward		nach Rechnung		
Motordrehzahl	ADC_AIN4	Spannung	ADC	Abweichung
38.116	2181.12	1.00	2276.87	95.75
75.839	2295.69	1.05	2391.29	95.60
113.321	2409.29	1.10	2504.99	95.70
151.266	2528.51	1.15	2620.09	91.58
189.365	2640.33	1.20	2735.66	95.33
227.045	2759.42	1.25	2849.95	90.53
264.737	2873.93	1.30	2964.29	90.36
302.629	2991.28	1.35	3079.22	87.94
340.938	3105.02	1.40	3195.43	90.41
378.551	3222.87	1.45	3309.52	86.65
416.296	3336.76	1.51	3424.01	87.25
454.351	3451.21	1.56	3539.45	88.24
492.396	3568.2	1.61	3654.85	86.65
530.091	3682.99	1.66	3769.19	86.20
567.748	3798.9	1.71	3883.42	84.52
605.832	3916.8	1.76	3998.94	82.14

Tab.5.3: Gegenüberstellung gemessen und errechnet

Wie man aus der Tabelle entnehmen kann, stimmen die Erwartungswerte mit den der gemessenen Werte, nicht überein.

Der kommentierte Programmrahmen, sowie die Rohdaten, der Aufzeichnung, können der beigelegten CD entnommen werden.

Aus den aufgenommenen Messdaten konnten nun wichtige Parameter für die Regelung bestimmt werden:

$$\text{PulswidthRpmRatio} \quad \left[\frac{\text{pulsweitenverhältnis}}{\text{min}^{-1}} \right]$$

$$\text{AdcRpmRatio} \quad \left[\frac{\text{adcvalue}}{\text{min}^{-1}} \right]$$

$$\text{AdcRpmRatio} = \frac{\sum_{i=1}^{16} \frac{\text{AdcValue}_{i+1} - \text{AdcValue}_i}{n_{i+1} - n_i}}{15} \quad \text{Gl.5.2.3}$$

$$\text{PulswidthRpmRatio} = \frac{\sum_{i=1}^{16} \frac{\text{Pulswidth}_{i+1} - \text{Pulswidth}_i}{n_{i+1} - n_i}}{15} \quad \text{Gl.5.2.4}$$

5.3 Softwareregler

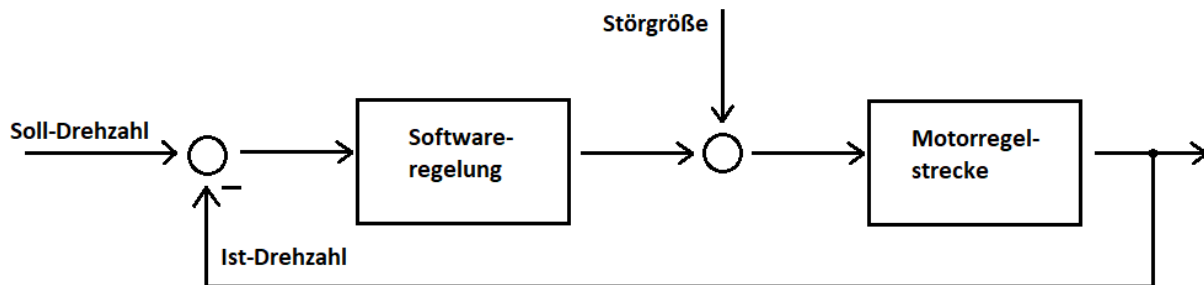


Abb.5.5: Regelstrecke

Der Softwareregler wurde wie folgt konzipiert:

Es wird eine Soll-Drehzahl eingestellt und die erste Pulsweite über den `PulswidthRpmRatio` bestimmt. Daraufhin gibt der Motor die Ist-Drehzahl in Form einer analogen Spannung an das System zurück. Diese Spannung wird zunächst in die Ist-Drehzahl umgerechnet. Die Abweichung zwischen dem Soll- und Ist-Wert bestimmt die zu entgegenregelte Pulsweite.

Bei einer Abweichung von $\pm 2.5 \text{ min}^{-1}$ wird die Pulsweite nicht verändert und stellt somit die Toleranz des Reglers, um den Sollwert, da. Des Weiteren wurden Parameter für unterschiedliche Abweichungen in 5 min^{-1} Schritten eingeführt. Bei höherer Abweichung vom Soll-Wert kann stärker/schwächer entgegen geregelt werden als bei kleineren Abweichungen.

Ebenfalls werden die Reglerparameter, von der vorgegebenen Soll-Drehzahl, um einen konstanten Faktor angepasst.

Des Weiteren kann eine Update-Time eingestellt werden, welche die Mittelwertbildung der gemessenen Drehzahlen bestimmt und die Zeit zwischen den einzelnen Regelsteuerungen setzt.

Daraufhin wurde versucht die Reglerparameter experimentell zu bestimmen. Dabei ist es zu einem nicht reproduzierbaren Fehler gekommen. Der Roboter fährt zunächst gerade aus und weicht dann nach Links oder nach rechts ab. Das Verhalten des Roboters ist somit nicht vorhersehbar, sodass weitere Versuche, die Reglerparameter anzupassen, nicht unternommen wurden.

Nach längerer Recherche¹ ist klar geworden, dass das Messprinzip der Encoder, zu viele Fehler aufweist. Die Encoder arbeiten nach dem Prinzip des Pulszählmessverfahrens. Der Fehler entsteht dadurch, dass in einem Reglertakt knapp die nächste Sensor-Flanke zählt, diese jedoch im nächsten Reglertakt nicht mehr vorhanden ist.

Diese Art der Regelung kann somit nicht genutzt werden.

¹<https://support.maxongroup.com/hc/de/articles/360016541693-Drehzahlmessung-und-Genauigkeit-der-Drehzahlregelung>

5.3.1 Streckenmessung

Damit bestimmt werden kann, wie schnell der Roboter bei welchem Pulsweitenverhältnis fährt, wurde eine Messstrecke aufgebaut. Die jeweiligen Pulsweiten wurden für 3 unterschiedliche Zeiten 3-mal gemessen. Aus diesen Messdaten lässt sich die Geschwindigkeit des Roboters ermitteln.

DutyCycle [%]	Zeit [ms]	Strecke [cm]			Durchschnittsgeschwindigkeit		
		Messung 1	Messung 2	Messung 3			
15	2000	10.3	10.2	10.2	5.2	5.1	5.1
	4000	20.5	20.3	20.4	5.1	5.1	5.1
	6000	30.5	30.4	30.5	5.1	5.1	5.1
20	2000	21	21	21	10.5	10.5	10.5
	4000	41	41.3	41.4	10.3	10.3	10.4
	6000	61.2	61.3	61.3	10.2	10.2	10.2
25	2000	31.9	32	32.2	16.0	16.0	16.1
	4000	62.2	62	62.1	15.6	15.5	15.5
	6000	92.3	92.5	92.5	15.4	15.4	15.4
30	2000	43.2	43.6	43.5	21.6	21.8	21.8
	4000	83.7	83.7	83.9	20.9	20.9	21.0
	6000	124.3	124.2	124	20.7	20.7	20.7
35	2000	54.6	55	55	27.3	27.5	27.5
	4000	104.5	105	104.7	26.1	26.3	26.2
	6000	155.6	156	156	25.9	26.0	26.0
40	2000	66	66	66.1	33.0	33.0	33.1
	4000	127	126.9	126.8	31.8	31.7	31.7
	6000	190	188.7	188.6	31.7	31.5	31.4

Tab.5.4: Streckenmessung

gemittelte Durchschnittsgeschwindigkeit	Abweichung
5.1	
10.34	5.24
15.65	5.31
21.12	5.47
26.53	5.41
32.09	5.56

Tab.5.5 Abweichungen der Messergebnisse

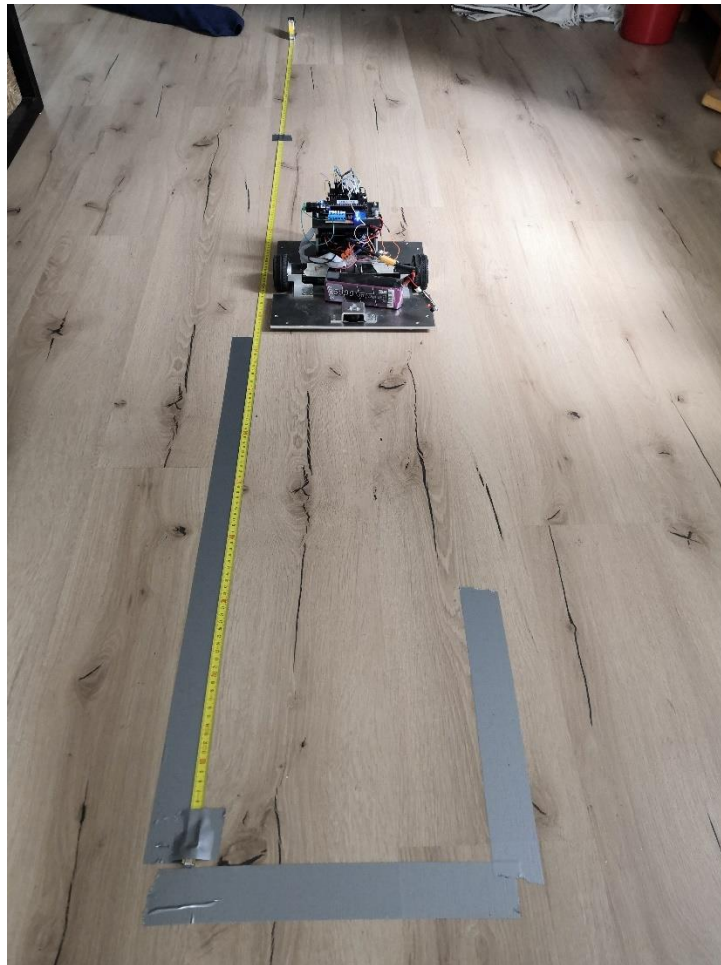


Abb.5.6: Messstreckenaufbau

Wie in Tab.5.8 ersichtlich ist, ist der Geschwindigkeitszuwachs bezogen auf das Pulsweitenverhältnis nicht linear. Somit kann keine Aussage über die zu fahrende Geschwindigkeit bei bestimmten Pulsweitenverhältnissen getroffen werden. Einen funktionellen Zusammenhang zu finden, scheiterte.

Um dennoch über eine Streckeneingabe fahren zu können, müssen die ADCs ausgelesen, umgerechnet und an das System zurückgegeben werden. Die Motoren sollen dann zum Halten kommen, sobald die vorgegebene Strecke erreicht wurde.

5.5 Messtechnische Bestimmung des Übersetzungsverhältnisses der ADCs

Die Motoren drehen in der Drehzahl langsamer, als über die Messung ermittelt wurde. Dies hängt mit der Einstellung der Drehzahlgeber zusammen. Eingestellt wurde eine Encoder-Auflösung von 500 Impulsen/Umdrehung.

Um zu ermitteln in welchem Verhältnis die gemessene Drehzahl und die tatsächliche Drehzahl stehen, wurde wieder eine Messung durchgeführt.

Dabei ist darauf zu achten, dass die bereits gemittelte Ist-Drehzahl von den Encodern softwaretechnisch über eine for-Schleife ebenfalls wieder gemittelt werden. In Abhängigkeit davon, wie oft die Schleife durchlaufen wird, vergeht ebenfalls eine gewisse Zeit. Die Motoren werden über eine do while – Schleife gesteuert. Ist die Zeit beim Eintritt der Schleife so weit fortgeschritten, dass die Motoren bei vorgegebener Zeit fast nicht mehr laufen dürften, könnte, durch die for-Schleife, zeitliche Fehler entstehen. Um diesen Sachverhalt zu umgehen, wurde getestet, wie lange die Schleife ausgeführt wird.

```
do {
    valueCount = 0;
    test.startTimer(); //starten des Timers zur zeitlichen Bestimmung der for-Schleife
    for(int i = 0; i < 200; i++)
    {
        valueCount += analogInput.readValue();
    }
    test.stopTimer(); //stoppen des Timers
    std::cout << test.getTime(msec) << " ms sind nach dem Einsprung in die Schleife vergangen" << std::endl;
    dummy = (double) valueCount / 200; //gemittelter ADC-Wert
    aktuelleRpm = (dummy - zeroPoint) / adcRpmSlope; //Umrechnung in min-1
    rpmGesamt += aktuelleRpm; //Aufsummieren der gemessenen gemittelten Drehzahlen
    count++; //Zählvariable für die Wiederholung der do-while-Schleife
} while(duration.getTime(msec) < _data.time); //Sobald vorgegebene Zeit überschritten wird, werden Motoren gestoppt
```

Abb.5.7: Codeausschnitt zur zeitlichen Erfassung der for-Schleife in *Odometry::superviseDriving*

```
96.264 ms sind nach dem Einsprung in die Schleife vergangen
105.776 ms sind nach dem Einsprung in die Schleife vergangen
99.627 ms sind nach dem Einsprung in die Schleife vergangen
96.93 ms sind nach dem Einsprung in die Schleife vergangen
95.479 ms sind nach dem Einsprung in die Schleife vergangen
99.545 ms sind nach dem Einsprung in die Schleife vergangen
96.502 ms sind nach dem Einsprung in die Schleife vergangen
97.597 ms sind nach dem Einsprung in die Schleife vergangen
96.657 ms sind nach dem Einsprung in die Schleife vergangen
101.476 ms sind nach dem Einsprung in die Schleife vergangen
95.785 ms sind nach dem Einsprung in die Schleife vergangen
96.285 ms sind nach dem Einsprung in die Schleife vergangen
99.851 ms sind nach dem Einsprung in die Schleife vergangen
98.72 ms sind nach dem Einsprung in die Schleife vergangen
100.059 ms sind nach dem Einsprung in die Schleife vergangen
96.78 ms sind nach dem Einsprung in die Schleife vergangen
```

Abb.5.8: Output Zeitmessung

Demnach werden maximal 105ms für 200 Durchläufe benötigt. Dies bedeutet, dass in etwa 0.525ms pro Durchlauf benötigt werden. Die Anzahl der Durchläufe wird mit dem Parameter `repetitionRate` übergeben.

Vor dem for-Schleifen-Eintritt wird nun überprüft, ob diese Zeit eingehalten werden kann.

```
do{
    valueCount = 0;

    if( _data.time - duration.getTime(msec) - repetitionRate * 0.49 > 0)
    {
        for(int i = 0; i< repetitionRate; i++)
        {
            valueCount += analogInput.readValue();
        }

        dummy = (double) valueCount / repetitionRate;           //gemittelter ADC-Wert
        aktuelleRpm = (dummy - zeroPoint) / adcRpmSlope;         //Umrechnung in min-1
        rpmGesamt += aktuelleRpm;                                //Aufsummieren der gemessenen gemittelten Drehzahlen
        count++;                                                  //Zählvariable für die Abschnittsausführung
    }
}while(duration.getTime(msec)<_data.time);                        //Sobald vorgegebene Zeit überschritten wird, werden Motoren gestoppt
```

Abb.5.9: Verbesserung des zeitlichen Verhaltens

Nach jeder Mittelwertbildung muss die gefahrene Strecke berechnet werden. Die Umdrehungsgeschwindigkeit und der gemessene ADC-Wert stehen in einem bestimmten Verhältnis zueinander.

```
do{
    deltaTimer.startTimer();                                     //start des Timers für Streckenrechnung
    valueCount = 0;
    if( Odometry_data.time - duration.getTime(msec) - repetitionRate * 0.49 > 0)
    {
        for(int i = 0; i< repetitionRate; i++)
        {
            valueCount += analogInput.readValue();
        }

        averageRpm = (double) valueCount / repetitionRate;       //gemittelter ADC-Wert
        currentRpm = (averageRpm - zeroPoint) * Odometry_data.RpmAdcRatio; //Umrechnung in min-1
        rpmGesamt += currentRpm;                                //Aufsummieren der gemessenen gemittelten Drehzahlen
        count++;                                                  //Zählvariable für die Abschnittsausführung
    }
    currentRpm = abs(currentRpm);
    Odometry_data.strecke += currentRpm * (deltaTimer.getTime(sec)/60) * 3.14159 * 6.2; //Umrechnungen in cm
    deltaTimer.stopTimer();                                       //stoppen des Timers für Streckenrechnung
}while(duration.getTime(msec) < Odometry_data.time);             //Sobald vorgegebene Zeit überschritten wird, werden Motoren gestoppt
```

Abb.5.10: Funktion zur Bestimmung des `RpmAdcRatio`

Dabei muss auch bekannt sein, wie viel Zeit seit dem letzten Mal Messen und berechnen der gefahrenen Strecke vergangen ist. Diese Zeit wird über den Timer `deltaTimer` ermittelt.

Um nun das Verhältnis messtechnisch bestimmen zu können, wird dieser Wert zunächst willkürlich gewählt. Da die ADCs einen Messwert zwischen 0 und 4096 liegt dieser Wert auf jeden Fall unter 1.

Somit wurde das `RpmAdcRatio` als erstes auf 1 gesetzt.

Danach wird der Roboter wieder auf die Messtrecke gestellt und bei einem Pulsweitenverhältnis von 25% für 8 Sekunden laufen gelassen.

```
rechts      Gefahrene Strecke[cm]: 578.776
Time passed: 8003.02
links      Gefahrene Strecke[cm]: 584.917
Time passed: 8000.8
```

Abb.5.11 Output für RpmAdcRatio von 1

Der Roboter ist nach den ADCs in etwa 580cm gefahren. Auf der Messtrecke konnte die tatsächlich gefahrene Strecke gemessen werden. Der entstandene Fehler kann nun entgegengerechnet werden.

$$RpmAdcRatio_{neu} = \frac{s_{gemessen}}{s_{Adc}} \cdot RpmAdcRatio_{alt} \quad \text{für} \quad s_{ADC} > s_{gemessen} \quad Gl.5.5.1$$

$$RpmAdcRatio_{neu} = \frac{s_{Adc}}{s_{gemessen}} \cdot RpmAdcRatio_{alt} \quad \text{für} \quad s_{ADC} < s_{gemessen} \quad Gl.5.5.2$$

Im Beispiel ist der Roboter tatsächlich 41.2 cm gefahren. Der neue RpmAdcRatio Wert beträgt somit:

$$RpmAdcRatio_{neu} = \frac{s_{gemessen}}{s_{Adc}} \cdot RpmAdcRatio_{alt} = \frac{41.2}{580} \cdot 1 = 0.07103 \quad Gl.5.5.3$$

```
rechts      Gefahrene Strecke[cm]: 41.0619
Time passed: 8002.08
links      Gefahrene Strecke[cm]: 41.5842
Time passed: 8001.49
```

Abb.5.12: Output nach Fehleranpassung

Der zurückgegebene Streckenwert stimmt nun mit dem tatsächlich gemessenen Wert bis auf eine geringe Abweichung überein. Die Abweichung nimmt bei höheren Geschwindigkeiten zu. Da mit diesem Test bewiesen werden konnte, dass die Messwerte der ADCs hinreichend genaue Messungen liefern, wurde die Odometrie-Klasse so umgebaut, dass die Räder zum Halten kommen, sobald die vorgegebene Strecke erreicht wird.

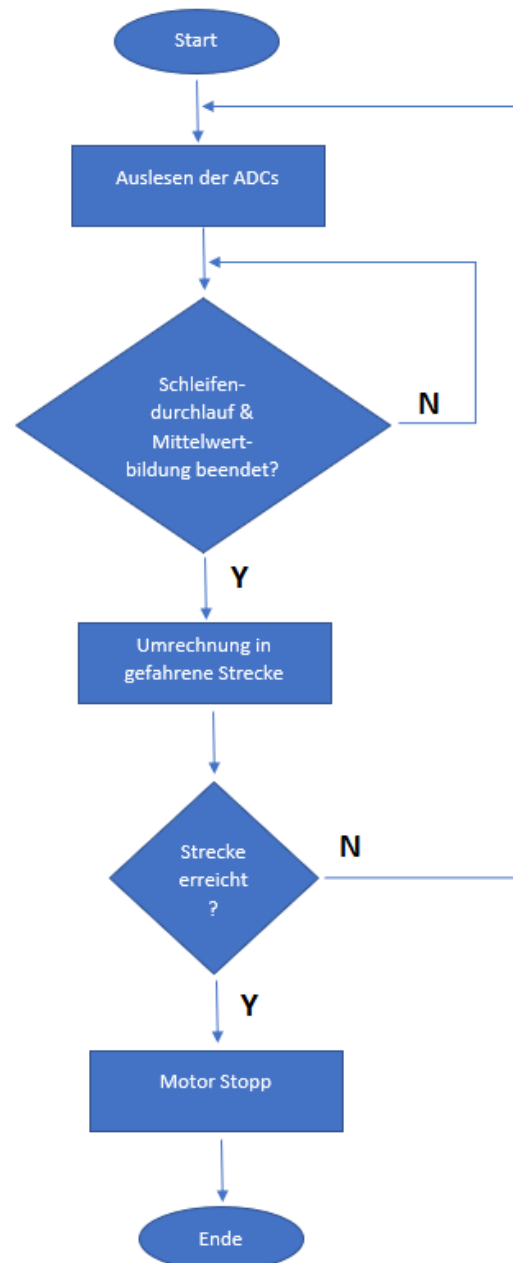


Abb.5.13: Flussdiagramm des Softwarereglers

Da der Regler nun zeitunabhängig agiert, wird der neue Fehler wie folgt entgegengerechnet:

$$RpmAdcRatio_{neu} = \frac{s_{gemessen}}{s_{Input}} \cdot RpmAdcRatio_{alt} \quad Gl.5.5.4$$

$s_{gemessen}$ beschreibt die auf der Teststrecke gemessene Strecke und s_{Input} die zu fahrende Strecke, welche an das System übergeben wird.

5.6 Messtechnische Bestimmung des Radabstandes

Um die Winkelstellung des Roboters auf dem Spielfeld zu verändern, muss dieser um sich selbst drehen, indem die Motoren gegenläufig angesteuert werden.

Die zu fahrende Strecke des Roboters berechnet sich aus:

$$s = \frac{\alpha}{360^\circ} \cdot \pi \cdot wheelSpace \quad Gl.5.6.1$$

Der gemessene Radabstand beträgt 26.2 cm. Um den gefahrenen Fehler zu bestimmen, wurde der Roboter auf ein Whiteboard gestellt und bei geringer Drehzahl um 360° drehen gelassen.

Daraufhin wurde der Winkel gemessen, den der Roboter wirklich gefahren ist. Der angegebene Winkel und der gemessene Winkel werden ins Verhältnis gesetzt und dem wheelSpace entgegengerechnet:

$$wheelSpace_{neu} = \frac{\alpha_{Input}}{\alpha_{gemessen}} \cdot wheelSpace_{alt} \quad Gl.5.6.2$$

5.7 Fahren bei erhöhter Geschwindigkeit

Der Roboter sollte auch bei höherer Drehzahl möglichst genau fahren können. Da die Abweichung bei einem plötzlichen Sprung auf eine höhere Geschwindigkeit Fehler verursacht, wurde eine Be- und Entschleunigungsphase in den Codeumfang implementiert.

Demnach wird das Pulsweitenverhältnis nach jedem Messen und anschließender Mittelwertbildung um 1% erhöht. Eine Veränderung des Parameters zum Schleifendurchlauf verändert somit auch das Be- und Entschleunigungsverhalten.

Entsprechend der Controller-Einstellungen fährt der Roboter bei 10% Pulsweitenverhältnis 0 min-1 und bei 90% 600 min-1. 1% der Pulsweite entsprechen somit 7.5 min-1.

Für 80 Schleifendurchläufe vergehen in etwa 70 ms. Somit benötigt ein Schleifendurchlauf ca. 0.875 ms.

Die Winkelbeschleunigung kann mit Gl.5.7.1 angenähert werden.

$$\alpha [s^{-2}] = \frac{0.125}{repetitionRate \cdot 0.875 \cdot 10^{-3}} \quad Gl.5.7.1$$

6. Programmierung Teil 2

6.1 Übersicht der Programmierung

Um eine bessere Übersicht über die Klassen zu geben und wie diese untereinander verknüpft sind, wurde ein UML-Klassendiagramm erstellt. Dieses ist auf der nächsten Seite zu finden.

Das Diagramm wird dabei wie folgt gelesen¹:

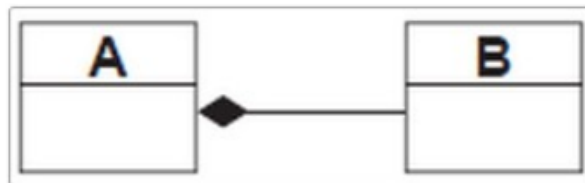


Abb. 6.1: Komposition: A besteht aus B

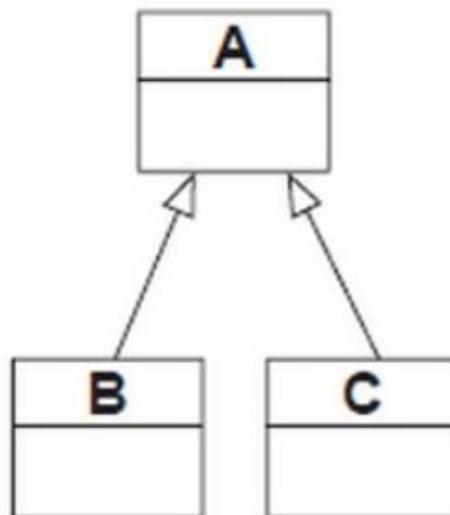


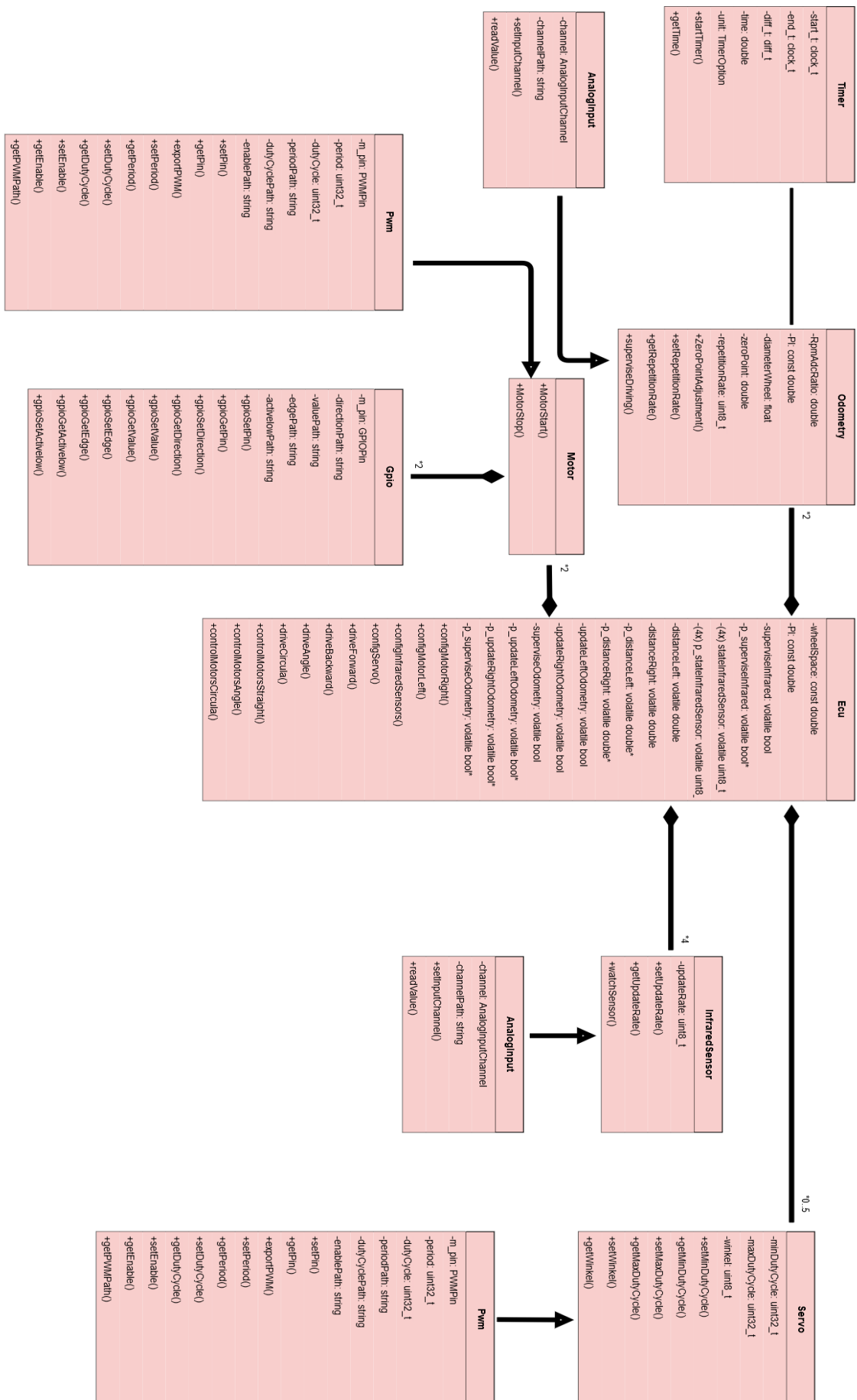
Abb.6.2: Vererbung: Jede Instanz von A besteht aus B und C



Abb.6.3: Assoziation; A benutzt ein B

¹<https://www.informatik.uni-leipzig.de/~stjaenicke/mup1/s2.pdf>

6.1.1 UML - Diagramm



Bei den bestehenden Basisklassen handelt es sich um Pwm und AnalogInput. Pwm wird seinerseits an Servo und an Motor vererbt, AnalogInput an InfraredSensor und Odometry.

Im Beispiel Pwm bedeutet dies, dass Servo und Motor alle Eigenschaften von Pwm übernehmen, ohne dass ein Objekt in Servo oder Motor instanziiert wird.

Beispiel 1: Motor enthält ein Objekt Pwm; keine Vererbung

```
MOTOR MotorLeft;           //Hier wird ein Objekt Motor instanziiert

MotorLeft.motorPwm.dutyCycle; //das Objekt Pwm welches das Attribut dutyCycle
                             //beinhaltet, kann über Motor so aufgerufen werden

MotorLeft.motorPwm.getPeriod(); //Methodenaufruf von Pwm über Motor
```

Beispiel 2: Pwm wurde an die Klasse Motor vererbt

```
MOTOR MotorLeft;

MotorLeft.dutyCycle;           //Motor benötigt kein Objekt vom Typ Pwm, da diese Klasse
                              //vererbt wurde

MotorLeft.getPeriod();         //Methodenaufruf von Pwm über Motor
```

Zur Vererbung gibt es folgende Regel:

Wenn jede Instanz der Klasse B eine Art von A ist, sollte Vererbung zum Zuge kommen.

Ein Objekt Motor oder ein Objekt Servo besteht genau aus einer Pwm. Analog besteht ein Objekt Infrarotsensor oder ein Objekt Odometry genau aus einem AnalogInput. Außer einiger weniger Attribute und Methoden, weisen diese sonst keine Unterschiede auf.

6.2 Beschreibung der Klassen & ihrer Funktionalitäten

6.2.1 PWM

Pwm
-m_pin: PWMPin
-period: uint32_t
-dutyCycle: uint32_t
-periodPath: string
-dutyCyclePath: string
-enablePath: string
+setPin()
+getPin()
+exportPWM()
+setPeriod()
+getPeriod()
+setDutyCycle()
+getDutyCycle()
+setEnabled()
+getEnable()
+getPWMPath()

Um eine PWM betreiben zu können muss diese zunächst exportiert werden.

Ist diese erfolgreich exportiert müssen die jeweiligen Pfade für die Einstellung der Pwm generiert werden.

Dies geschieht mit dem Methodenaufruf `setPin()`.

Daraufhin können über die Ordnerpfade die jeweiligen Einstellungen (Period, DutyCycle, Enable) durchgeführt werden. Diese werden mit dem entsprechenden Methodenaufruf gesetzt.

6.2.2 AnalogInput

Die ADCs werden über Dateien im Pfad `/sys/bus/iio/devices/iio:device0/` ausgelesen.

```

debian@beaglebone:~$ ls /sys/bus/iio/devices/iio:device0
buffer  in_voltage0_raw  in_voltage2_raw  in_voltage4_raw  in_voltage6_raw  name      power      subsystem
dev      in_voltage1_raw  in_voltage3_raw  in_voltage5_raw  in_voltage7_raw  of_node   scan_elements  uevent
debian@beaglebone:~$ cat /sys/bus/iio/devices/iio:device0/in_voltage2_raw
486
  
```

Abb.6.4: Dateien zum Auslesen der ADCs

AIN0 entspricht `in_voltage0_raw`, AIN1 `in_voltage2_raw`, usw.

Über den Methodenaufruf `setInputChannel()` wird der gewünschte AnalogInput gesetzt und der zugehörige Dateipfad generiert.

Die Methode `readValue()` liest den ADC aus und gibt den Wert zurück.

AnalogInput
-channel: AnalogInputChannel
-channelPath: string
+setInputChannel()
+readValue()

6.2.3 GPIO

GPIO
-m_pin: GPIOPin
-directionPath: string
-valuePath: string
-edgePath: string
-activeLowPath: string
+gpioSetPin()
+gpioGetPin()
+gpioSetDirection()
+gpioGetDirection()
+gpioSetValue()
+gpioGetValue()
+gpioSetEdge()
+gpioGetEdge()
+gpioGetActiveLow()
+gpioSetActiveLow()

Ein Gpio-Pin wird über den Ordner `/sys/class/gpio/gpio[NUMBER]` eingestellt.

Die Ordnerpfade werden über die `gpioSetPin()` Methode generiert.

Daraufhin können die Einstellungen über die entsprechenden Methodenaufrufe durchgeführt werden.

6.2.4 Timer

Hierbei handelt es sich um einen Softwaretimer.

Der Timer wird mit `startTimer()` gestartet.

Über `getTime()` wird die Zeit in Abhängigkeit von `TimerOption` in Sekunden, Millisekunden oder Microsekunden in dem Attribute `time` gespeichert und zurückgegeben. Diese misst die Zeit zwischen dem letzten Methodenaufwurf von `startTimer()`.

Die Auflösung des Timers liegt in etwa bei 200 Microsekunden

Timer
-start_t: clock_t
-end_t: clock_t
-diff_t: diff_t
-time: double
-unit: TimerOption
+startTimer()
+getTime()

6.2.5 Motor

Motor
+MotorStart()
+MotorStop()

Die Klasse `Motor` ist eine abgeleitete Klasse von `Pwm`. Diese erbt somit alle Attribute und Methoden von `Pwm`.

Des Weiteren besteht diese aus zwei Objekten `GPIO`. Diese werden für die Freigabe und Drehrichtung benötigt. Die eingestellte Periode für die Motoren beträgt 2.000.000

6.2.6 Servo

Die Klasse Servo ist eine abgeleitete Klasse von Pwm. Diese erbt somit alle Attribute und Methoden von Pwm.

Die Attribute minDutyCycle und maxDutyCycle legen die Winkelstellung für 0° und 180°. Diese sind vom verwendeten Servomotor abhängig.

Die Attribute können über die entsprechenden Methoden verändert werden. SetWinkel() setzt die Pulsweite auf die gewünschte Winkelstellung. Über die Basisklasse Pwm, wird mit dem Methodenaufruf setEnable(ON) die entsprechende Winkelstellung gefahren.

Sofern der Servomotor die Winkelstellung halten soll, muss die Pwm eingeschaltet bleiben.

Bei den verwendeten Servomotor NES 100 STD von NineEagles beträgt die minDutyCycle 550.000 und die maxDutyCycle 2.450.000 bei einer Periode von 20.000.000.

Servo
-minDutyCycle: uint32_t
-maxDutyCycle: uint32_t
-winkel: uint8_t
+setMinDutyCycle()
+getMinDutyCycle()
+setMaxDutyCycle()
+getMaxDutyCycle()
+setWinkel()
+getWinkel()

6.2.7 InfraredSensor

InfraredSensor
-updateRate: uint8_t
+setUpdateRate()
+getUpdateRate()
+watchSensor()

Die Klasse InfraredSensor ist eine abgeleitete Klasse von AnalogInput. Diese erbt somit alle Attribute und Methoden von AnalogInput.

Über die Methode setUpdateRate() wird die Anzahl der Schleifendurchläufe zur Mittelwertbildung des ausgelesenen ADCs gesetzt.

WatchSensor() führt die Mittelwertbildung aus und setzt eine State Variable. Diese teilt den Motoren mit, sobald der Roboter auf ein Hindernis zufährt und entschleunigt die Fahrt. Dies soll den Fehler der gemessenen Strecke reduzieren. Die StateVariable wird über die Ecu Klasse als Referenz an den Thread der Methode watchSensor() übergeben.

InfraredSensor::watchSensor(volatile uint8_t* _state, volatile bool* _superviseInfrared)

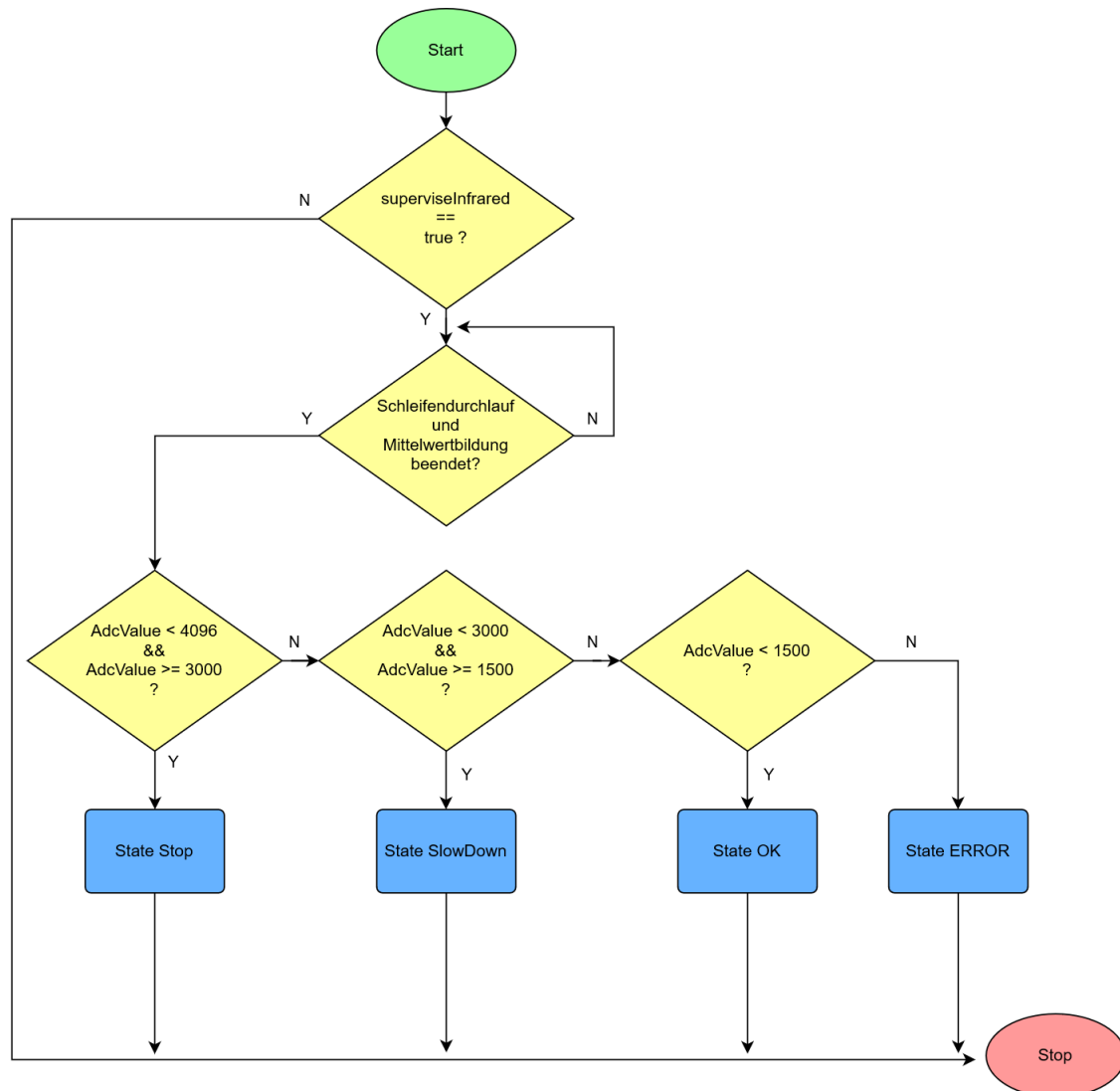


Abb.6.5: Flussdiagramm der Funktion InfraredSensor::watchSensor

Der gelesene Adc Wert kann maximal 4096 betragen. Sofern dieser über diesem Wert liegt, ist ein Fehler aufgetreten. Bei der AdcValue Variable handelt es sich um eine uint32_t Variable, sodass diese nie kleiner 0 sein kann.

Ausgeführte Threads sollten immer beendet werden. Aus diesem Grund wird eine volatile Bool Referenz an die Funktion übergeben. Sobald ein Fahrbefehl ausgeführt wurde, wird die Bool Variable auf False gesetzt und der Thread somit beendet.

6.2.8 Odometry

Odometry
-RpmAdcRatio: double
-PI: const double
-diameterWheel: float
-zeroPoint: double
-repetitionRate: uint8_t
+ZeroPointAdjustment()
+setRepetitionRate()
+getRepetitionRate()
+superviseDriving()

Die Klasse Odometry ist eine abgeleitete Klasse von AnalogInput. Diese erbt somit alle Attribute und Methoden von AnalogInput.

Der RpmAdcRatio wurde messtechnisch bestimmt. Der zeroPoint beschreibt den ADC-Wert der Analogen Eingänge für die Drehzahlmessung.

Die Drehzahlmessung liefert 0.15V bei -600 min^{-1} , 0.95V bei 0 min^{-1} und 1.75V bei $+600 \text{ min}^{-1}$. Nach einem Rebootvorgang des Beaglebones kann es vorkommen, dass die Nullpunktreferenz sich um einige Millivolt verändert. Dieses Verhalten konnte auch ohne das Beaglebone-Cape festgestellt werden. Die Funktion ZeroPointAdjustment() misst die anliegende Spannung bei nicht drehenden Motoren. Dies

geschieht auch hier über eine Mittelwertbildung.

superviseDriving() ermittelt die gefahrene Wegstrecke und wird mit einer double Referenz an Ecu zurückgegeben. Hierzu wird ein Timer benötigt. Dieser misst lediglich die Zeit zwischen dem letzten Mal messen und der Umrechnung der gefahrenen Strecke.

Diese Funktion wird ebenfalls als Thread ausgeführt und bekommt einen volatile bool Referenz, als Übergabeparameter.

6.2.9 Ecu

In der Ecu Klasse finden alle Funktionen zum Berechnen der Wegstrecke und zum Steuern der Motoren statt. Alle Variablen die als volatile deklariert wurden, werden von anderen Funktionen in anderen Objekten genutzt. Um diese im Original bearbeiten zu können, müssen Pointer auf die entsprechenden Variablen an den Thread übergeben werden.

Die config-Funktionen konfiguriert die jeweiligen Objekte. Hier werden durch Methodenaufrufe, der bestehenden Objekte, die String Pfade generiert, Pwm-Module konfiguriert, AnalogInputs zugewiesen und GPIOs gesetzt.

Die Funktionen driveForward, driveBackward, driveAngle und driveCircula setzen die entsprechende Drehrichtung für die Motoren. driveAngle und driveCircula müssen zusätzlich noch die zu fahrende Distanz für den jeweiligen Fahrbefehl berechnen. In diesen Funktionen werden ebenfalls die Threads initialisiert und ausgeführt. Für jeden dieser Fahrbefehle besteht eine control Funktion. Diese agieren in Abhängigkeit der übergebenden Referenzen von anderen Threads.

Ecu
+configMotorRight()
+configMotorLeft()
+configInfraredSensors()
+configServo()
+driveForward()
+driveBackward()
+driveAngle()
+driveCircula()
+controlMotorsStraight()
+controlMotorsAngle()
+controlMotorsCircula()

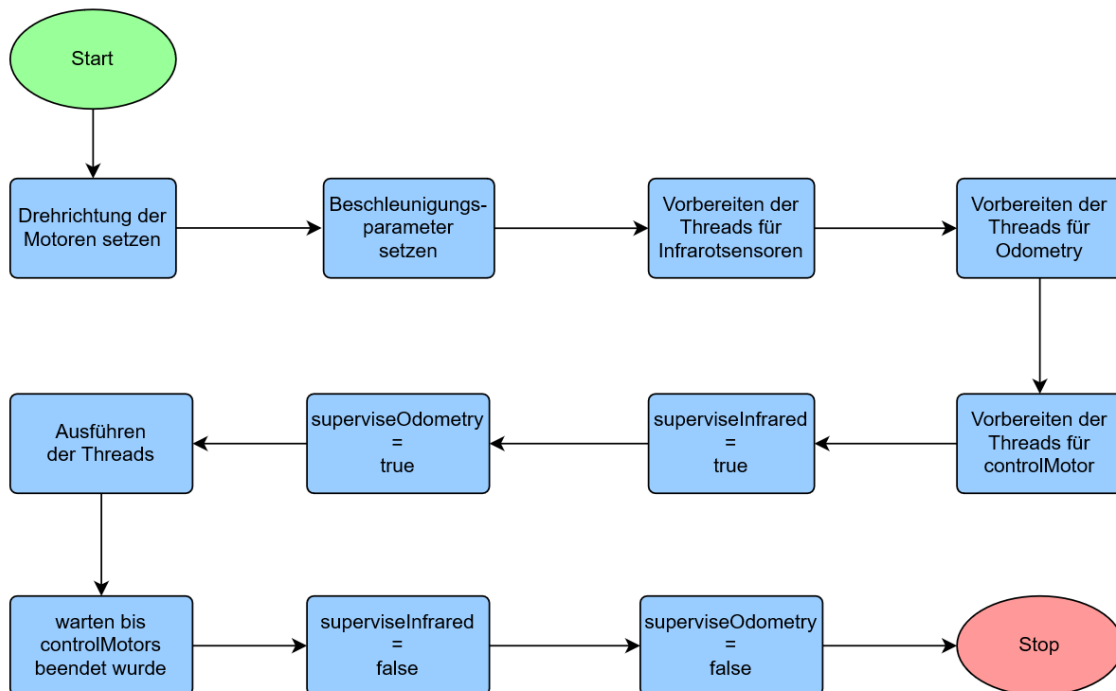


Abb. 6.6: Flussdiagramm Ecu::driveForward und Ecu::driveBackward

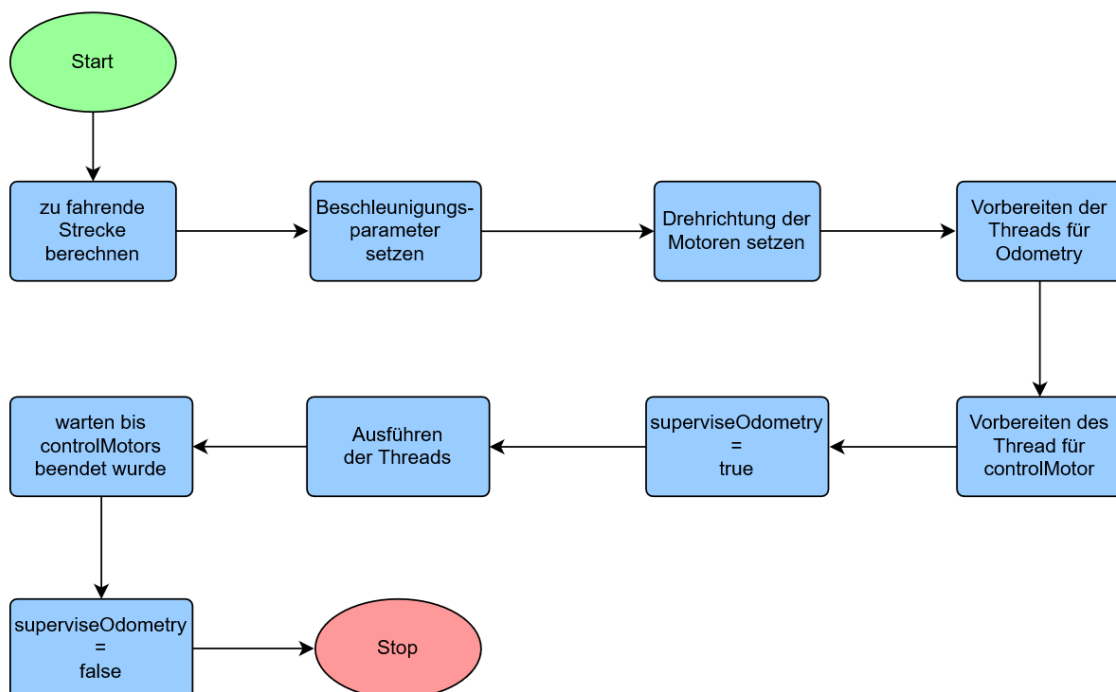


Abb. 6.7: Flussdiagramm Ecu::driveAngle

Der Roboter dreht bei einer Winkelstellungsänderung lediglich um sich selbst. Daher sind keine Infrarotsensoren nötig, die diese Aktion überwachen. Folglich ist das Flussdiagramm für `controlMotorsAngle()` und `controlMotorsStraight()` das Gleiche. Bei Ersterem können die Infrarotsensoren jedoch nicht den Programmablauf beeinflussen.

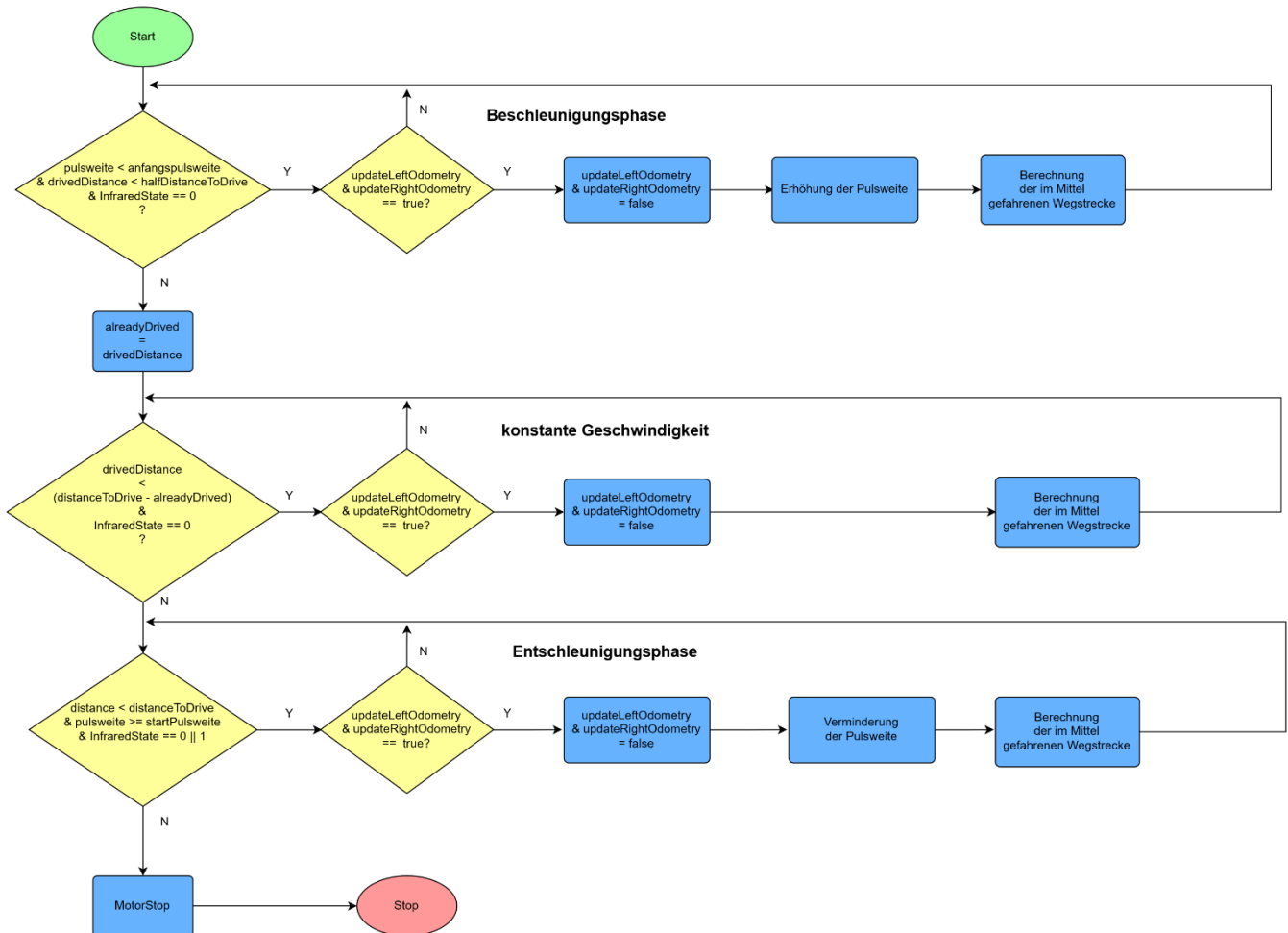


Abb. 6.8: Flussdiagramm `Ecu::controlMotorsStraight`

7. Fazit

Die Funktion `driveCircula` berechnet die Geschwindigkeiten zum Fahren eines Kreisbogens nicht richtig. Das Rad, welches langsamer dreht, kann die zu fahrende Strecke nicht erreichen.

Im Nachhinein ist klar geworden, dass es in Bezug der Systemeffizienz, vorteilhafter gewesen wäre, mit Variablen vom Typ `float` anstatt vom Typ `double` zu arbeiten. Der Prozessor ist auf einer 32-Bit Architektur aufgebaut und die Größe eines `float` entspricht genau 32-Bit.

Die Funktionen zum be-, entschleunigen und konstante Geschwindigkeit fahren, könnten in separate Funktionen ausgelagert werden. Diese könnten dann über einen Signal Handler entsprechend des States, der Infrarotsensoren, für die jeweiligen Fahrbefehle aufgerufen werden.

Hierzu müsste vorher allerdings noch eine Klasse `SignalHandler` entworfen werden.

Es wurden kaum Errors verarbeitet. Wenn ein bestimmter Fehler auftritt, wird die zugehörige Funktion nicht ausgeführt und eine Fehlermeldung über `std::cout` erscheint. Es gibt bessere Möglichkeiten unter C++ mit Errors umzugehen.

Die Elektronik für die Analogen Spannungen sollte überarbeitet werden. Hierzu sollte man prüfen, warum sich die Spannung um einige Millivolt hebt oder senkt. Ebenfalls könnte eine geeignete Filterschaltung¹ die Signale sauberer an die ADCs bringen. Der Roboter wurde im Rahmen dieser Projektarbeit in die GTEM-Zelle gestellt. Da sich diese Projektarbeit mehr um die Software drehte, wurde darauf nicht weiter eingegangen. Das Bild des Messergebnisses ist im beiliegenden Ordner `Bilder->GTEM_Zelle` zu finden.

Es wurde noch keine Klasse für die Schrittmotoren erstellt. Es ist noch nicht klar, welcher Motortyp beim Hissen der Flagge zum Einsatz kommt.

Es gibt noch keine Funktionen, um alternative Routen zu fahren. Der Regler hat bis zur geplanten Abgabe nicht richtig funktioniert. Hier musste somit mehr Zeit investiert werden als geplant. Die Idee ist es, entsprechend der Infrarotsensorik eine Ausweichroute zu finden.

¹<https://www.aktivfilter.de/>

8.1 Abbildungsverzeichnis

Abb.1: Übertragung in den eMMC Speicher

Abb.3.1: Einstellen des Verbindungsziels in WinSCP

Abb.3.2: Ausführen der Makefile und anschließendes testen

Abb.3.3: Ordnerpfad zu den jeweiligen pwmchips

Abb.3.4: Ordnerinhalt eines pwmchip

Abb.3.5: exportieren der PWM am Beispiel pwm-7:0

Abb.3.6: Ordnerinhalt der fertig exportierten PWM

Abb.3.7: Auskommentierte Option in /boot/uEnv.txt

Abb.4.1: Prinzip der ADC-Schutzschaltung

Abb.4.2: Spannungsteiler Eingang Operationsverstärker Infrarotsensoren

Abb.4.3: Anschluss der analogen Eingangsspannungen an das Cape

Abb.5.1: Auto-Regler-Tuning abgeschlossen

Abb.5.2: Einstellungen Datenaufzeichnung

Abb.5.3: Motormessung

Abb.5.4: gemessene Drehzahl; orange: hinzugefügte ADC-Werte

Abb.5.5: Regelstrecke

Abb.5.6: Messstreckenaufbau

Abb.5.7: Codeausschnitt zur zeitlichen Erfassung der for-Schleife in Odometry::superviseDriving

Abb.5.8: Output Zeitmessung

Abb.5.9: Verbesserung des zeitlichen Verhaltens

Abb.5.10: Funktion zur Bestimmung des RpmAdcRatio

Abb.5.11 Output für RpmAdcRatio von 1

Abb.5.12: Output nach Fehleranpassung

Abb.5.13: Flussdiagramm des Softwarereglers

Abb. 6.1: Komposition: A besteht aus B

Abb.6.2: Vererbung: Jede Instanz von A besteht aus B und C

Abb.6.3: Assoziation; A benutzt ein B

Abb.6.4: Dateien zum Auslesen der ADCs

Abb.6.5: Flussdiagramm der Funktion InfraredSensor::watchSensor

Abb. 6.6: Flussdiagramm Ecu::driveForward und Ecu::driveBackward

Abb. 6.7: Flussdiagramm Ecu::driveAngle

Abb. 6.8: Flussdiagramm Ecu::controlMotorsStraight

8.2 Tabellenverzeichnis

Tab. 3.1: Lookup-Tabelle der genutzten Chips und den zugeordneten Pins

Tab.5.1: Motor Messung Links

Tab.5.2: Motor Messung Rechts

Tab.5.3: Gegenüberstellung gemessen und errechnet

Tab.5.4: Streckenmessung

Tab.5.5 Abweichungen der Messergebnisse

9 Konzept

Aufgabenstellung für eine **Projektarbeit**

am Fachbereich Elektro- und Informationstechnik der THM Gießen

Titel: „Weiterentwicklung der Eurobot-Fahrbasis“

Bearbeiter: Christopher Sauer

Betreuer: Prof. Dr. Thomas Glotzbach

Datum: ab 01.09.2020

9.1 Individueller Teil

9.1.1 Beschreibung des Themas

Die studentische Arbeitsgruppe M.A.M.U.T. nimmt regelmäßig am EUROBOT-Wettbewerb teil. Dazu existiert ein mobiles Robotersystem, welches gegenwärtig auf der Grundlage der Projektarbeit von Nikolas Wenzlitschke weiterbearbeitet wird. Ziel seiner Projektarbeit war es, eine angenehme Arbeitsumgebung zur Weiterentwicklung des Systems aufzustellen und mit gewissen Ansätzen der Softwareprogrammierung einen ordentlichen Einstieg für Interessierte zu bieten.

Die Fahrbasis konnte zu dem Zeitpunkt Fahrbefehle entgegennehmen, diese jedoch nicht richtig umsetzen. Der Weg, der zurückgelegt wurde, hat nicht dem entsprochen, was der Roboter eigentlich fahren sollte. In weiterführenden Arbeiten bestand die Aufgabe darin, die Programmierung mit diesen Vorgaben und eigenen Ansätzen zu übernehmen. Dazu wurden bereits Codeabschnitte gefertigt, die in der Lage sein sollten, über einen ADC die Drehbewegung der Motoren zu detektieren, um Einfluss auf die Fahrtstrecke nehmen zu können (Felix Köppge). Hierzu wurden allerdings noch keine Tests durchgeführt.

Ebenfalls wurde bereits eine kleine OP-Schaltung von Christopher Sauer und Alexander Busch ausgearbeitet, welche es ermöglicht, die analogen Signale der Infrarotsensoren in digitale umzuwandeln. Ab wann die Sensoren ein High liefern, kann durch ein Potenziometer variiert werden. Dies war ein erfolversprechender Ansatz, führt aber zu folgendem Problem:

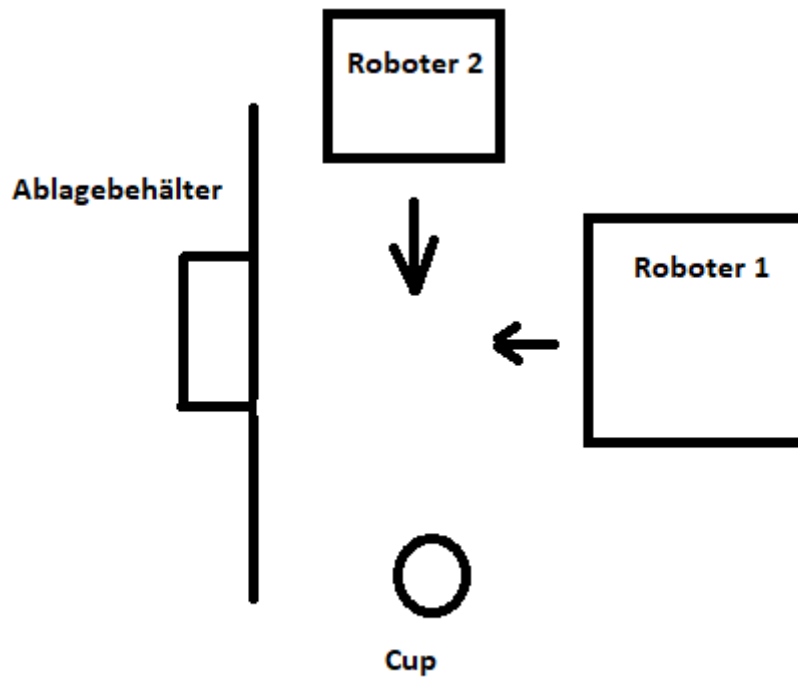


Abbildung 1: Beispielszenario

Da nur Bool-Werte von den Infrarotsensoren an das System zurückgemeldet werden, müssten im Beispielszenario nach Abbildung 1 die Infrarotsensoren softwaretechnisch „abgeschaltet“ werden, da sich Roboter 1 dem Cup bzw. der Ablagestation annähern soll. Dies führt dazu, dass Roboter 1 den anfahrenden Roboter 2 nicht mehr erkennt und es so zu einer Kollision kommen könnte. Werden allerdings die ADCs der IR-Sensoren ausgewertet, könnten durch geschickte Programmierung solche Worst-Case Fälle besser bearbeitet werden. Hierzu müssten noch geeignete Methoden gefunden werden. (Software)

Das genutzte Board hat hierzu 6 ADC-Eingänge, welche mit maximal 1,8V versorgt werden dürfen. Um dies zu gewährleisten wird eine ADC-Schutzschaltung mit Operationsverstärkern entworfen.

Da es noch keinen funktionierenden Code für die Odometrie gibt, wird sich die Projektarbeit vornehmend über die Ausprogrammierung dessen drehen.

Ebenfalls sollen einfache Methoden und Funktionen für die Infrarotsensoren gefunden werden, um oben beschriebene und folgende Situationen meistern zu können:

Was passiert, wenn sich beide Roboter gegenüberstehen und zum Halten kommen?

Der Roboter sollte einen alternativen Weg fahren, um die Aufgaben weiter zu erledigen. Dabei sollte dieser wieder auf seine überlegte Strecke zurückfinden und die verlorene Zeit im späteren Programmablauf berücksichtigt werden.

Des Weiteren werden allgemeine Klassen zur Ansteuerung diverser Motortypen entwickelt (Servo-, Schritt- und Gleichstrommotoren), um weitere Module, wie zum Beispiel zum Aufheben der Cups, leicht in den Code implementieren zu können.

Der entwickelte Code sollte bestimmte Sicherheitsaspekte erfüllen (Ports dürfen nicht mehrmals belegt werden, Fehlerhandling, usw.)

9.2 Allgemeiner Teil

9.2.1 Dokumentation

Die Dokumentation soll aus den folgenden Unterlagen bestehen:

- Der eigentlichen Projektarbeit, welche im Stil einer wissenschaftlichen Arbeit die Aufgabenstellung als Problembeschreibung enthält sowie durchgeführte Recherchen, aus der Literatur übernommene Ansätze sowie eigenen Ansätze, deren Adaptionen und Umsetzungen (inkl. der einzelnen erfolgreichen und auch nicht erfolgreichen Entwicklungsschritte), Validierungsergebnis sowie eine Zusammenfassung und einen Ausblick. Beachten Sie, dass das Schreiben dieser Arbeit Sie auf Ihre bevorstehende Bachelorarbeit vorbereiten soll.
- Ein Servicehandbuch, welches alle relevanten Informationen (je nach Hard-/Softwareprojekt: den Quellcode, den Schaltplan, die Gerberdaten für die Platine, die mechanischen Zeichnung, eine vollständige Stückliste, Programmablaufplan als Flußdiagramm, Blockschaltbild, und alle zum Verständnis nötigen und hilfreichen Unterlagen) enthält. Anhand dieses Servicehandbuches soll ein Nachbau bzw. spätere Adaption an ähnliche Problemstellungen problemlos möglich sein.
- ein Bedienerhandbuch für eine nicht-Fachkraft, die das Projekt verwenden will.
- ein Datenträger (CD, DVD, Karten-USB-Stick, welche/r "verliersicher" der Projektarbeit beigelegt ist), welcher alle o.g. Dokumente im .pdf-Format und alle zusätzlichen zum Projekt gehörenden Dateien (Gerberdaten, Firmware, Platinenlayout, Schaltplan im Format des jeweiligen Layoutprogramms, Simulationsdateien, etc.) enthält.

Hinweis: Die Punkte zwei und drei können als Anhang in Punkt eins enthalten sein, sofern dies nicht den Umfang völlig ausufern lässt.

9.2.2 Durchführung und Abgabe

Zu Beginn der Projektphase ist ein Zeitplan mit Meilensteinen anzufertigen, um den Zeitaufwand grob abzuschätzen. Die Bearbeitungszeit beträgt maximal ein Jahr, wobei in Abständen von etwa zwei Wochen ein kurzer formloser Fortschrittsbericht (mündlich oder per Email) erwartet wird, um einem etwaigen Stagnieren des Projekts frühzeitig entgegen zu wirken und eventuelle Probleme zu klären. Sollte in einem Zeitrahmen von zwei Monaten keine Rückmeldung erfolgen, so gilt die Projektarbeit als nicht bestanden und der Betreuer behält sich vor, das Thema neu zu vergeben.

Zur Abgabe des Projekts sind bei Hardware-Projekten von jedem Aufbau mindestens ein Exemplar anzufertigen, welches an der Hochschule verbleibt. Alle im Abschnitt 9.2.1 beschriebenen Unterlagen müssen selbst erstellt sein. Alle Zitate sind mit Quellenangabe als solche kenntlich zu machen. Der Arbeitsaufwand muss mindestens 150 Stunden betragen. Sollte sich herausstellen, dass der Arbeitsaufwand zur Realisierung des Projekts signifikant weniger als 150 h beträgt, so behält sich der Betreuer vor, zusätzliche Leistungsmerkmale in die Projektanforderungen aufzunehmen. Die Abgabe aller Dokumente erfolgt sowohl in Papierform als auch in Form eines Datenträgers gemäß den Angaben in Abschnitt 9.2.1. Bei Hardware-Aufbauten soll sich im Gehäuse des Aufbaus ein Datenträger mit der installierten Firmware sowie das Servicehandbuchs in Papierform befinden.

9.2.3 Bewertung

Die Bewertung der Projektarbeit erfolgt nach dem folgenden Schema:

Praktische Tätigkeit

- Arbeitsqualität (Kreativität, analytische & konzeptionelle Fähigkeiten, Lernbereitschaft)
 - 30%
- Arbeitsstil (Selbstständigkeit, Eigeninitiative, Zuverlässigkeit, Termintreue, Effizienz)
 - 20%
- Kommunikationsfähigkeit (persönliches Auftreten, Visualisierung, Dialog- und Konfliktfähigkeit, Zeitmanagement)
 - 10%

Dokumentation

- Sachlicher Inhalt (Korrektheit, Verständlichkeit, Aussagekraft)
 - 20%
- Ausdruck, Stil, Grammatik (Rechtschreibung, Zeichensetzung, Formulierungen)
 - 10%
- Technik des wissenschaftlichen Arbeitens (Zitieren, Verzeichnisse)
 - 10%

9.3 Meilensteine

Index	Name	Erfolgt
MS1.0	Diverse Fehlerbehebung ➔ Übertragung in den EMMC-Speicher ➔ Reproduzierbare Fehler der Motoransteuerung beheben	Ja
MS1.1	Basisklassen (Motortypen) für die Nutzung weiterer Module erstellen	Ja
MS2.0	Löten der ADC-Schutzschaltung	Ja
MS3.0	Klassen für Analoginputs erstellen	Ja
MS3.1	Inbetriebnahme der Infrarotsensoren	Ja
MS4.0	Prüfung der Störempfindlichkeit	Ja
MS5.0	Überarbeitung der Odometrie	Ja
MS5.1	Entwicklung geeigneter Methoden und Funktionen der Infrarotsensoren	Ja

Zeit	Ziele	Notiz
26.10.20-30.10.20	MS1.0 MS1.1	erfolgt
02.11.20-06.11.20	Start MS2.0	erfolgt
09.11.20-13.11.20	Placeholder	-
16.11.20-20.11.20	MS3.0 MS3.1	Noch nicht abgeschlossen
23.11.20-27.11.20	Placeholder	-
30.11.20-04.12.20	Placeholder	-
07.12.20-11.12.20	MS4.0	Keine Gegenmaßnahmen erforderlich
14.12.20-18.12.20	Ende MS2.0	Warten auf Bestellung
21.12.20-24.12.20	Placeholder	-
04.01.21-08.01.21	Placeholder	MS 3.0 erfolgt, MS 3.1 nur mit Ende MS 2.0 möglich
11.01.21-19.02.21	Prüfungsvorbereitung + Prüfungen	ADC-Schutzschaltung gelötet, MS 2.0 erfolgt
22.02.21 – 26.02.21	Start MS 5.0	erfolgt
01.03.21 – 05.03.21	Placeholder	-
08.03.21 – 26.03.21	Prüfungsvorbereitung + Prüfungen	-
29.03.21 - 02.04.21	Ende MS 5.0	Regler läuft nicht
05.04.21 – 09.04.21	MS 5.1	Nicht erfolgt
12.04.21 - 30.04.21	Placeholder	Überarbeiteten des Reglers
03.05.21	Abgabe	Nicht erfolgt
31.05.21	Ende MS 5.0	Erfolgt
01.06.21	Start MS 5.1	Erfolgt
08.06.21	Ende MS 5.1	Erfolgt
10.06.21	Abgabe	Erfolgt

Note: Das Rot markierte wurde nachträglich hinzugefügt