Hochschule Esslingen
University of Applied Sciences
Fakultät Informationstechnik

[HElikopter

Studienarbeit

Quadrocopter - Hardware Migration auf das Raspberry Pi Board

im Studiengang Technische Informatik
der Fakultï¿½t Informationstechnik
Wintersemester 2015/2016

Chris Mönch

Prï¿½fer: Prof. Dr. Jörg Friedrich
Zweitprï¿½fer: M.Sc. Vikas Agrawal

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 9. Februar 2016 _____

Unterschrift

Kurzzusammenfassung

In diesem Projekt wird die Migration des MicroKopter/HElikopter von der FlightCtrl Hauptplatine zu dem Raspberry Pi ausgearbeitet und dokumentiert. Es werden die grundsätzliche Vorgehensweise und die verwendeten Programm- und Linuxfunktionen festgehalten.

Zielaufgabe ist es, den HElicopter einen Schritt weiter zum autonomen Fliegen zu bekommen und einen *Quickstart* über den HElicopter und Umgebung bereitzustellen.

Grundaufbau des HElikoptern basiert auf die Anleitung, die auf dieser Seite beschrieben ist. Ebenso findet man dort weitere Informationen uns Wikis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das Projekt HElicopter ist für Bachelor- und Masterstudenten aus fakultätsübergreifenden Bereichen, um einen Quadrocopter flugfähig zu machen.

Die einzelnen verschieden Quadrocopter Typen wurden bereits mit studentischer Hilfe zum Fliegen gebracht und können mittels einer Funk-Fernbedienung gesteuert werden.

In den nächsten Schritten soll der Quadrocopter autonom flugfähig gemacht werden. Hierfür wurde die Standardplatine ausgetauscht und durch ein Raspberry Pi mit benötigter Peripherie ersetzt.

Das Projekt deckt die folgende Themengebiete ab:

- Steuer- und Regelungstechnik
- Echtzeit-Betriebssysteme
- Elektronik und Elektrotechnik
- Embedded Systems
- Sichere Systeme
- Echtzeit Programmierung
- Autonome Systeme

2 Grundlagen

2.1 Virtuelle Maschine/Destop PC

Für das Projekt *Quadrocopter* wird eine virtuelle Maschine verwendet. Hierfür wird eine bereits fertig funktionierende Vorlage bereitgestellt.

Auf dieser VM läuft das Betriebssystem Ubuntu v14.0.

Für die Verwendung der VM wird ein Programm wie z.b. VirtualBox von Oracle oder VMware Workstation Player (abgespeckte Version von VMware Workstation, kostenlos für private Nutzung) benötigt.

Änderungen an den VMware-Einstellungen müssen nicht vorgenommen werden.

Als Alternative wird bei Problemen oder ähnlichem ein Desktop-PC im EZS Labor bereitgestellt.

LogIn Daten

Desktop Rechner:
Computer-Name: EZS-Labor
Username: ezs
Passwort: labor

VMware:
VMware Name: Ubuntu 64-bit
Username: user
Passwort: user

Raspberry Pi:
Username: pi
Passwort: raspberry

2.1.1 Ubuntu

VPN Zugang einrichten *Derzeit kann maximal eine VPN-Konfiguration zur selben Zeit aktiv sein, da die Zertifikate global abgespeichert sind.*

Verbinden Sie nun den PC mit einem schwarzen Ethernet-Port (z.B. Box F1.301.16). Die Seite *hs-esslingen.de* sollte nun erreichbar sein. Um von diesem PC mit seinen Hochschuldaten Internet zu haben müssen die folgenden Schritte gegebenenfalls durchgeführt werden.

Eventuell muss die Quelle der Pakete/packages geändert werden. In der */etc/apt/sources.list* muss auf eine verfügbare Quelle verwiesen werden, da ohne VPN die Standard-Pfade nicht erreichbar sind. Unter dieser Seite sind alle nötigen Daten verfügbar: ftp-stud.hs-esslingen.de/ Mit der Angabe des folgenden Befehls macht man diesen Pfad bekannt:

```
1 deb http://ftp-stud.hs-esslingen.de/ubuntu/ trusty main universe
    restricted multiverse
```

Für VPN müssen die folgenden Pakete *openvpn* und *easy-rsa* per

```
1 sudo apt-get install PACKETNAME
```

installiert werden.

Als nächsten Schritt erfolgt die Einrichtung des VPN-Zuganges wie hier angegeben.

Gestartet wird die OpenVPN-Verbindung mit folgendem Befehl:

```
1 sudo /etc/init.d/openvpn start
```

oder alternativ:

```
1 sudo service openvpn start
```

Der LogIn erfolgt mit dem jeweiligen Benutzerkonto.

Die VPN-Verbindung wird mit demselben Kommando und dem Parameter *stop* abgebaut.

2.1.1.1 Autostart bei Bootvorgang

Achtung: Das Linux Betriebssystem startet nicht mehr, falls es sich bei dem Script oder Programm um ein Endlos-Script/Programm handelt oder eine User-Interkation erwartet wird!

Es ist für einmalige Einstellungen und Konfigurationen vorgesehen.

Um ein Script zu schreiben müssen Sie zunächst ein File in dem Verzeichnis */init.d* erzeugen: sudo nano /etc/init.d/myAutostartScript

```
sudo nano /etc/init.d/myAutostartScript
```

Als nächstes müssen sie den folgenden Grundaufbau des Inhalt übernehmen:

```
#! /bin/sh
### BEGIN INIT INFO

# Short-Description: Starts myProgramm
# Description:
### END INIT INFO

case "$1" in
start)
echo "noip wird gestartet"
# Starte Programm
/usr/local/bin/myProgramm
;;
stop)
echo "myProgramm wird beendet"
# Beende Programm
killall myProgramm
;;
*)
echo "Benutzt: /etc/init.d/myAutostartScript {start|stop}"
exit 1
;;
esac

exit 0
```

Später wird beim Start des PC oder Raspberry das Script automatisch mit dem Parameter *start* ausgeführt. Beim Herunterfahren wird natürlich der Parameter *stop* übergeben. Für jeden anderen Fall wird hier *Benutzt: /etc/init.d/myAutostartScript start|stop* ausgegeben, um auf die Syntax-Anwendung des Scripts zu verweisen.

Anschließend muss das Programm noch durch den folgenden Befehl ausführbar gemacht werden:

```
sudo chmod 755 /etc/init.d/myAutostartScript
```

Um einen Test durchzuführen kann das Script mit den folgenden Befehlen gestartet bzw. gestoppt werden:

```
sudo /etc/init.d/myAutostartScript start
sudo /etc/init.d/myAutostartScript stop
```

Nach erfolgreichem manuellen Test erfolgt die Einstellung, dass das Script beim booten und beim Herunterfahren automatisch aufgerufen wird.

```
sudo update-rc.d NameDesSkripts defaults
```

Falls das Script nicht mehr automatisch aufgerufen werden soll wird dies mit folgendem Befehl verhindert:

```
sudo update-rc.d -f  NameDesSkripts remove
```

2.1.1.2 Autostart bei LogIn für User

Für das Starten von Programmen für bestimmte User können die *.desktop* Dateien verwendet werden. Diese Dateien müssen in den jeweiligen Home-Verzeichnissen des Users unter folgendem Pfad abgelegt werden: *.config/autostart/*

Der Name darf bis auf die Endung *.desktop* frei gewählt werden
(z.B. *myAutostart.desktop*).

Ein neues File kann mit

```
sudo nano ~/.config/autostart/myAutostart.desktop
```

erzeugt werden. Der Aufbau eines *.desktop*-Files ist wie folgt dargestellt:

```
[Desktop Entry]
Type=Application
Terminal=true
Exec=/home/user/workspace/RaspberryDemoUdpSendHost/Debug/
   RaspberryDemoUdpSendHo$
Hidden=false
NoDisplay=false
X-GNOME-Autostart-enabled=true
Name[en_US]=RaspberryDemoUdpSendHost
Name=RaspberryDemoUdpSendHost
Comment[en_US]=
Comment=
```

Benötigte Parameter:
Zur Programmerstellung eines Autostarts wird der *Typ* Äpplication"gewählt.
Der Parameter *Exec* beinhaltet den Pfad zu der ausführbaren Datei, welche im Terminal zu starten ist.
Mit dem Parameter *Name* wird der Prozessname festgelegt.

Optionale Parameter:
Um ein Terminal sichtbar zu machen muss die Einstellung *Terminal* auf "true"gesetzt werden.
Über den Parameter *X-GNOME-Autostart-enabled* kann man das Autostarten des Programmes aussetzen, wenn man diesen Parameter auf "falseßtellt. Standardmäßig ist er auf "true! gesetzt.

Falls das Programm bei Beendigung erneut gestartet werden soll ist dies über X-GNOME-AutoRestart möglich.

2.1.2 VMWare Workstation Player

Der VMware Player steht hier zum Download zur Verfügung.

Eine bereits existierende VM muss geöffnet werden, um die Kopie der VW verwenden zu können. Wählen Sie die .vmx aus und starten Sie di VM. Bei der Frage, ob es sich hierbei um eine *moved* oder *copied* VM handelt, wählen Sie *I copied it* aus. Die VM kann verwendet werden.

Die VM kann jetzt verwendet werden.

2.1.3 VirtualBox

Für die hier verfügbare VirtualBox können durch Änderungen in den Einstellungen ebenfalls .vmdk-Dateien verwendet werden. Das File sollte aber keinesfalls konvertiert werden und im .vdmk-Format bestehen bleiben.

Zunächst muss eine neue virtuelle Maschine des Typs Linus mit Betriebssystem Ubuntu 64bit-Version erstellt werden. Weisen Sie der VM 2 GB Hauptspeicher zu. Wählen Sie eine bereits vorhandene Festplatte aus. Bestimmen Sie dann den Ordner mit grünem Pfeil sowie die *.vmdk* der kopierten VM aus und erstellen die VM.

Wählen Sie nun die gerade erstellte VM aus und gehen auf ändern. Im Menü *System* muss das IO-APIC aktivieren aktiviert sein. Unter dem Menüpunkt *Massenspeicher* entfernen Sie, falls vorhanden, die *.vdmk* unter SATA-Controller und fügen diese unter Controller:DIE neu hinzu indem Sie auf *Festplatte hinzufügen* klicken. Bestätigen Sie die Einstellungen.

Nun kann die *.vmdk* auch unter VirtualBox verwendet werden.

2.2 Quadrocopter

Falls es nötig sein sollte, den HELicopter zu demontieren, muss zuvor die Spannungsversorgung abgeschaltet werden. Beim Wiederaufbau sollte der Raspberry so montiert werden, dass das Gyroskop genau mittig sitzt, um keine verfälschten Sensorwerte zu bekommen

2.2.1 Raspberry Pi B+

Das Raspberry Pi B+ ist ein von der *Raspberry PI Foundation* entwickelter Einplatinen Computer. Auf diesen Mini-PC läuft die Hauptanwendung der Software für das Projekt. Auf dem Pi läuft das Betriebssystem Ubuntu.

2.2.1.1 Technische Daten

- Broadcom BCM2835 Mainboard
- ARM1176JZF-S Prozessor 700 MHz
- Braodcom VideoCore 4 Grafikkarte mit OpenGL ES 2.0, 1080p30 h.264/MPEG-4 AVC high-profile decoder
- 512 MB Arbeitsspeicher
- Audio Ausgänge: 3.5 mm Jack
- Onboard microSD Speicherkartenslot
- 10/100 Ethernet RJ45 onBoard
- Speichermöglichkeit über microSD Speicherkartenslot
- 4x USB 2.0.

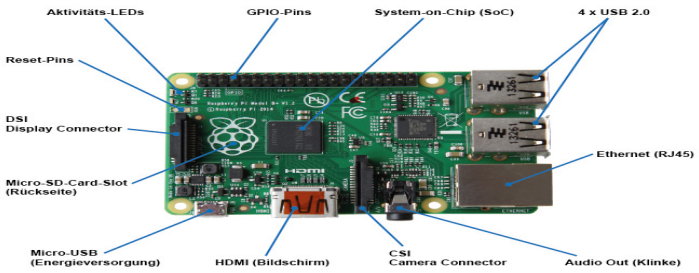


Abb. 2.1: Raspberry PI B+[2]

2 https://www.elektronik-kompendium.de/sites/raspberry-pi/bilder/19052513.jpg

2.2.1.2 Verbinden mit einem PC

Benötigt wird eine Ethernet-Verbindung zwischen Host und Raspberry Pi, die SD-Karte mit Image des verwendeten Betriebssystems Ubuntu und die Stromversorgung.

Auf das Raspberry Pi kann über mehrere Möglichkeiten zugegriffen werden. Hierfür sind die IP Konfigurationen des Pi´s und Hosts anzupassen (die IP vom Pi sollte unverändert sein).

Host-Ip:
IP 192.168.22.160
Maske: 255.255.255.0
GW: 192.168.22.09

PI-IP:
IP 192.168.22.161
Maske: 255.255.255.0

Die Login-Daten auf dem Raspberry lauten:
User: pi
Passwort: raspberry

Testen Sie die Einstellungen indem sie das Raspberry Pi von dem jeweiligen verwendeten Host anpingen.

Nach erfolgreich ausgeführtem Ping-Befehl ist es nun möglich, Programme oder Dateien mit dem Kommando scp an das Raspberry zu senden.

Folgende Schritte sind bei einem auftretenden Ping-Befehl-Fehler vorzunehmen:

Das Default Gateway muss zunächst ermittelt werden, um die Ip-Einstellungen zu ändern.

```
1 sudo nano /etc/network/interfaces
```

Nun müssen noch die Einstellungen in der Datei */etc/network/interfaces* anpasst werden.

```
1 sudo nano /etc/network/interfaces
```

Das geöffnete File sollte folgendermaßen aussehen:

```
1 auto lo
2
3 iface lo inet loopback
4 iface eth0 inet static
5 address 192.168.22.161
6 netmask 255.255.255.0
```

```
7  gateway 192.168.22.9
8  #iface eth0 inet dhcp
9
10 allow-hotplug wlan0
11 iface wlan0 inet manual
12 wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
13 iface default inet dhcp
```

Das Interface eth0 ist eine statische Adresse mit der dargestellten IP, Maske und Gateway. Ein Eintrag für das Routing ist vorzunehmen.

Der Netzwerkdienst muss neu gestartet werden, um die Änderungen übernehmen zu können.

```
1 sudo /etc/init.d/networking restart
```

```
1 scp Quelle Ziel[USER@IP:PFAD]
2 scp RaspberryDemoUdpSendHost pi@192.168.22.161:/home/pi/
```

Es können auch Dateien vom PI auf den Host kopiert werden. Folgender Befehl kopiert die Datei *file* vom Pi in das aktuelle Verzeichnis.

```
1 scp pi@192.168.22.161:/home/pi/file .
```

Benutzung eigener Monitor und Tastatur Es besteht die Möglichkeit über HDMI und USB einen Monitor und Tastatur anzuschließen. Hierfür muss die Netzwerkkonfiguration angepasst werden.

Über das Terminal Sie können sich über eine *ssh* Verbindung auf das Raspberry Pi einloggen. Sollte die Ip-Adresse des Raspberry verändert worden sein, muss diese angepasst werden.

```
1 ssh pi@192.168.22.161
2 ssh USERNAME@IP
```

Die Eingabe des Passwortes des Benutzers *pi* erfolgt über eine Blindeingabe.

2.2.2 Peripherie

2.2.2.1 Motoren

Bei den Motoren handelt es sich um über Treiber "BL-Ctrl V1.2"gesteuerte Brushless DC Motoren "Robbe ROXXY BL-Outrunner 2824-34". Die Ansteuerung des Treibers erfolgt über den I2C Bus. An den Treiber werden 2 Byte Daten gesendet:

- die I2C Adresse des gewünschten Motors
- den PWM-Wert (kleiner Wert langsam drehen, hoher Wert schnelles drehen)

Der Treiber behält die Werte 10 msec lang und verliert diese anschließend. Die Werte müssen also zyklisch gesendet werden.

Die Anzahl und Ordnung der Motoren ist je nach Typ des Helikopters unterschiedlich. Es gibt vier- oder acht Motorvarianten welche verschiedene Grundaufbauten haben können (+ oder x).

Der Code soll so allgemein gehalten werden, dass er für jeden dieser Typen verwendet werden kann. In der main.h wird mittels Definition angegeben für welchen Helikopter das Programm definiert werden soll. In der */hal/MOTOR/MOTOR.h* sind helikopterspezifische Daten abgelegt, darunter die I2C-Adressen jedes Motors, die Anzahl der Motoren, die min-Werte und max-Werte des PWM-Signales.

2.2.2.2 Adafriut Ultimate GPS PI HAT[3]

Dieses Modul wird direkt auf das PI aufgesteckt und leitet alle Pins bis auf die UART Pins weiter. Folgende Pins werden für das Bord benötigt und können nicht mehr verwendet werden:

Pin	Verwendung	Fest
UART TXD UART RXD	Die einzige serielle Schnittstelle muss verwendet werden um mit dem GPS-Modul zu kommunizieren	✓
GPIO #4	Kann bei Bedarf verwendet werden, falls die Zeitsynchronisation mit dem Raspberry nicht benötigt wird	✗
EEDATA EECLK	Werden für die Verbindung mit dem EEPROM benötigt, derzeit noch nicht vom Raspberry verwendet	✓

Tab. 2.1: Ultimate GPS Pins

Dieses Modul liefert das GPS-Modul *FGPMMOPA6H*(Datasheet). Das Modul bietet Platz für eine Batteriezelle, für eine Real-Time-Clock und Prototypen Platz, um weitere Bauteile mittels Löten hinzuzufügen.

2.2.2.3 ADS1015 12-Bit ADC[4]

Hierbei handelt es sich um einen hochauflösenden 12bit Analog-Digital-Konverter. Der Konverter kann bis zu 3300 Umwandlungen in der Sekunde berechnen und besitzt eine I2C-Schnittstelle. Die Standardadresse des Bausteines ist die 7bit-Adresse 0x47. Diese Adresse kann durch die Verbindung folgender Pins mit dem *ADDR-Pin* bei Bedarf angepasst werden:

Adresse	Pin 1	ADDR
0x48	GND	ADDR
0x49	VDD	ADDR
0x4A	SDA	ADDR
0x4B	SCL	ADDR

Tab. 2.2: ADC 12 Bit I2C Adress Manipulation

Dieser ADC liefert folgende zwei verschiedene Betriebsmodi, welche über die Anschlüsse *A0* bis *A3* verwendet werden:

- *Single Ended*: berechnet den ADC der Spannung zwischen jedem der *Ax-Pins* und Ground. Hiermit ist nur das Messen von positiven Spannungen möglich. Effektiv verliert man dadurch ein Bit der Auflösung. Der Modus bietet doppelt so viele Inputs.
- *Differential*: berechnet den ADC der Spannung zwischen *A0* & *A1* und *A2* & *A3*. Der Modus liefert weniger rauschanfällige Signale.

Eine Spannung über 5V darf nicht angelegt werden, da diese das Modul zerstören würde.

2.2.2.4 Pololu AltIMU-10 v4[5]

Dieses Modul besteht aus drei IC´s. Sie beinhalten die folgenden Funktionen:

Funktion	IC	Default I2C Adresse	Manipulate I2C
Beschleunigungsmesser Magnetometer	LSM303D	0x1D	0x1E
Gyrometer	L3GD20H	0x6B	0x6A
Barometer	LPS25H	0x5D	0x5C

Tab. 2.3: IMU ICs

Auch dieser Baustein verwendet zur Konfiguration und Kommunikation den I2C-Bus (7bit Adresse). Jeder dieser ICs besitzt eine eigene Adresse, die sich ebenfalls manipulieren lässt, indem man den Pin *SA0* auf *GND* zieht.

2.2.2.5 LIDAR-Lite v2[6]

Dieser Lasersensor besitzt eine Reichweite bis zu $40m$ mit einer Genauigkeit von $\pm 0{,}025$, eine Verzögerung von 0,02 Sekunden und kann bis zu 500 Messwerte pro Sekunde liefern. Der I2C-Bus kann mit $100kbits/s$ oder $400kbits/s$ betrieben werden. Eine eigene Adressierung ist möglich. Als Standard-I2C Adresse ist die 0x62 festgelegt. Falls mehrere LIDAR Light v2 Sensoren an einem I2C-Bus hängen ist dies die Broadcast-Adresse.

2.3 IDE

In der VM wird die IDE *Eclipse* mit einem bereits installierten *GCC* verwendet. Dieser *GCC* kann für verschiedene Plattformen compilieren:

- zum *Native Compiler* (übersetzt in den Maschinencode, der für die aktuelle Plattform benötigt wird)
- zum *Cross Compiler* (für die Verwendung um den Maschinencode für eine andere Plattform zu generieren)

Das Storeage Passwort von Eclipse der VMware lautet:
user
Das Storeage Passwort von Eclipse des Desktop-PC im EZS-Labor lautet:
labor

2.3.1 Compiler, Linker und Loader

Beim Compilieren werden sämtliche definierte Konstanten-, Makros- und Präprozessoranweisungen durch die angegebenen Werte ersetzt.

Der Compiler hat die Hauptaufgabe aus dem Quellcode (.c, .cpp und .h) in nativen Maschinencode zu übersetzen. Der Maschinencode kann nur von einem bestimmten Prozessor gelesen werden. Ausnahmen hierbei sind z.B. Compiler in der .Net (C#) Umgebung. Bei diesem werden die Source Dateien in sogenannte *Intermedia Languages* übersetzt, welche unabhängig vom Zielsystem sind. Bei der Ausführung des Codes wird der *Intermedia Languages* durch den Compiler bei jedem Start erneut in den prozessorspezifischen Maschinencode übersetzt.

Weitere Aufgaben eines Compilers:

- Anzeige von Syntax Fehler und Warnungen
- Codeoptimierungen durchzuführen (verschiedene Stufen möglich)
- Entfernen von Codesegmenten, die nie ausgeführt werden können
- Einfügen von eigenen Funktionscode bei einem Funktionsaufruf

Als Ausgabe liefert der Compiler Code denn sogenannten Objekt Code.

Die einzelnen Objekt-Files müssen noch durch einen sogenannten Linker miteinander verbunden werden. Nach erfolgreicher Verbindung generiert der Linker die ausführbare Datei. Vom Linker werden unter anderem Funktionen aus der Standard Libary hinzugefügt und alle Objektdateien, die noch nicht aufgelöste Symbole enthalten, überprüft und gelöst. Man unterscheidet zwischen statischen und dynamischen linken. Das statische linken wird einmalig durchgeführt, dynamisches linken hingegen jedes mal zur Laufzeit.

Zuletzt gibt es noch den Loader. Dieser Loader lädt Teile des ausführbaren Codes, die in nächster Zeit benötigt werden, in den Hauptspeicher.

2.3.2 GNU Debugger GDB

Der *Gnu DeBugger* ist ein Standard Debugger von Linuxsystemen der die folgenden Sprachen unterstützt: C, C++, Objective C, FORTRAN, Java, Pascal...

Neben den Standardaufgaben wie Stacktrace und Breakpoints setzen ermöglicht der GDB auch Manipulationen von Variablen während der Laufzeit des Programmes und Reverse Debugging.

Für Eclipse ist ein PlugIn installiert, das den DBG zur Verfügung stellt.

2.3.3 Makefile

Makefiles sind eine Möglichkeit (mittels des Programmes *make*) um aus mehreren Quell-Files und Bibliotheken über Objekt-Files einen ausführbaren Code zu generieren. Das Besondere an dieser Methode ist, dass beim Ausführen des Makefiles nur geänderte Quell-Dateien neu kompiliert werden müssen. Die daraufhin generierten Objekt-Files und die unveränderten Objekt-Dateien werden mit dem Linker neu verbunden. Beim herkömmlichen Compilieren werden sämtliche Quell-Dateien generiert und verbunden. Kurz: es lassen sich Abhängigkeiten definieren. Dies ist zeitsparend, da nicht bei Änderung eines Files das komplette Projekt compiliert werden muss. Dies ist bei großen Projekten zu berücksichtigen.

Die Makefiles werden in einer Baumstruktur angelegt.

Bearbeitung von Makefiles Für die Kompilierung neuer Source-Files müssen diese in den Makefiles bekanntgemacht werden und die Übersetzungs-Einstellungen bestimmt werden.

Zu bearbeiten sind im Trunk Ordner die Files *Makefile* und *makeopts* und das jeweilige *makefile* im Ordner, in dem eine neue Quelldatei hinzugefügt wurde. Für Host-Programme sind die files mit der Endung *-host* zu bearbeiten. Änderungen sind am Beispiel einer neuen c Source Datei *MOTOR.c* im neu erstellen Ordner *hal/MOTOR*.

Im *trunk/Makefile* sind die Verweise auf alle anderen Makefiles aufgelistet. Dies ist das Root-Makefile. Folgende Änderungen sind vorzunehmen:

```
1 product: ... MOTOR ...
```

Für den Make all Befehl:

```
MOTOR:FORCE
cd hal/MOTOR; make all
```

Für den Clean Befehl muss noch ein weiterer Eintrag erstellt werden.

```
cd hal/MOTOR; make clean
```

In dem *trunk/makeopts* sind Einstellungen für das Makefile oder Generierung mit dem Makefile festgelegt. Fügen Sie die nachfolgende Zeile ein:

```
../hal/MOTOR/MOTOR.lib\
```

Zuletzt muss noch das Makefile im Ordner MOTOR erstellt und bearbeitet werden. Dieses legt nun fest, welche Files für die Generierung verwendet werden sollen. Hierfür kopiert man am besten ein vorhandenes makefile um und ändert anschließend die Zeilen, die aussagen welches *.obj* und *.lib* file zum kompilieren und builden verwendet werden soll.

```
OBJS= MOTOR.obj MOTOR1.obj
LIBRARY=MOTOR.lib
```

Sollten sich mehrere *c* files im Ordner *Motor* befinden, können diese durch einen Space getrennt angeben werden. Mit diesem Makefile wird aus *MOTOR.C* wird das object file *MOTOR.obj* generiert.

2.3.4 Host Programme

Um ein Project für eine Host-Anwendung zu erstellen, muss unter Eclipse ein neues *C/C++ Projekt* mit dem Projekttyp *Executable Empty Projec t* und den Toolchains *Linux GCC* angelegt werden.

2.3.5 Raspberry Pi Programme

Um ein Projekt für eine Raspberry Pi Remote-Anwendung zu erstellen, muss unter Eclipse ein neues *C/C++ Projekt* mit dem Projekttyp *Executable Empty Project* und den Toolchains *Cross GCC* angelegt werden.

Bei den Projekteigenschaften muss bei *C/C++ Build/Settings* beim Tool Settings der Cross Settings ausgewählt werden und der Prefix und Pfad zum verwendeten GCC angepasst werden.

Prefix: *arm-linux-gnueabihf-*

Pfad: */home/user/rpi/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin*

Debug Remote Einstellungen Bei *Run as/Run Configurations* muss unter *C/C++ Remmote Apllication* eine neue Konfiguration erstellt werden. Im Feld *Projekt* die Projektbezeichnung eingetragen.

Unter *Remote Absolute File Path for C/C++ Application* wird der Pfad angegeben, in welchem die ausführbare Datei gespeichert ist
(z.B. */home/ezs/git/helicopter-raspberry/NameDerAnwendung.elf*)

Falls das Projekt neu gebaut werden muss sind die Target Einstellungen zu wählen.

Connection: Raspberry Target auswählen.

Remote Absolute File Path for C/C++ Application: Hier den Pfad und Dateinamen angeben, in welchen die ausführbare Datei gespeichert werden soll z.B. /home/pi/HELIKOPTER.elf

In das Feld *Commands to execute before application* ist folgender Eintrag vorzunehmen:
sudo -i
chmod +x RaspberryDemoUdpReceiveTarget
Dies macht das File zu einer ausführbaren Datei und wird bei jedem Laden des Programmes auf das Raspberry Pi ausgeführt.

Unter dem Reiter *Debugger* ist das Häckchen bei *Stop on Starup at* zu setzen und in das daneben liegende Feld *main* einzutragen. Dies sorgt dafür das bei einem Programmstart automatisch ein Breakpoint bei Aufruf der Funktion Main erzeugt wird.

Bei Debugger Optionen ist der Pfad des GDB auf den *gcc-linaro-arm-linux-gnueabihf-raspian-x64/bin/arm-linux-gnueabihf-gdb* und *GDB command file* auf *./gdbinit* zu ändern. Der Debugger kann auf GitHub runtergeladen werden.

Alternativ kann auch der scp Befehl verwendet werden um die Datei auf das Raspberry Pi zu kopieren.
scp ProgramName UserNameVonPi@IPAdressePi:SpeicherpfadAufPi
Dann per ssh Zugriff, die Datei im angegeben Pfad zu einer ausführbaren Datei machen und mit ./PfadZumFile.elf starten.

2.3.6 Beispiel Code

Anbei befinden sich zwei Codesegmente, die eine Nachricht über UDP/IP vom Raspberry Pi an den Host senden. Mit diesen Programmen können die zuvor beschriebenen Schritte zum Testen durchgeführt werden.

RaspberryDemoUdpSendHost.cpp:

```
1  /*
2  * RaspberryDemoUdpSendHost.cpp
3  *
4  *  Created on: Oct 23, 2015
5  *      Author: Chris Mönch
6  */
7
8  #include <iostream>
9  #include <stdio.h>
10 #include <sys/socket.h>
11 #include <netinet/in.h>
12 #include <string.h>
13 #include <unistd.h>
14
15 using namespace std;
16
17 int main(int argc, char * argv[]) {
18
19 cout << "Start\n";
20
21 int clientSocket, nBytesMessage, nBytesMessage2;
22 char message[12] = "Hello_World";
23 char message2[16] = "AnotherMessage";
24
25 nBytesMessage= sizeof(message)/ sizeof(message[0]);
26 nBytesMessage2 =sizeof(message2)/ sizeof(message2[0]);
27
28 struct sockaddr_in serverAddress;
29 socklen_t addressSize;
30
31 /*Create UDP socket*/
32 clientSocket = socket(PF_INET, SOCK_DGRAM, 0);
33
34 /*Configure settings in address struct*/
35 serverAddress.sin_family = AF_INET;
36 serverAddress.sin_port = htons(9999);
37 serverAddress.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
38 memset(serverAddress.sin_zero, '\0', sizeof serverAddress.sin_zero
      );
39
40 /*Initialize size variable to be used later on*/
41 addressSize = sizeof(serverAddress);
42
43 printf("Start␣Sending␣Messages\n");
44
45 while(1){
```

```
46 sleep(1);
47 sendto(clientSocket,message,nBytesMessage,0,(struct sockaddr *)&
      serverAddress,addressSize);
48 sleep(1);
49 sendto(clientSocket,message2,nBytesMessage2,0,(struct sockaddr *)&
      serverAddress,addressSize);
50 printf("And␣send␣again....\n");
51 }
52
53 return 0;
54 }
```

RaspberryDemoUdpReceiveHost.cpp:

```
1 /*
2 * RaspberryDemoUdpReceiveHost.cpp
3 *
4 *  Created on: Oct 23, 2015
5 *      Author: Chris Mönch
6 */
7
8 #include <sys/types.h>
9 #include <sys/socket.h>
10 #include <netinet/in.h>
11 #include <arpa/inet.h>
12 #include <netdb.h>
13 #include <stdio.h>
14 #include <unistd.h>
15 #include <string.h>
16 #include <stdlib.h>
17 #include <errno.h>
18 #include <time.h>
19
20 #define LOCAL_SERVER_PORT 9999
21 #define BUF 255
22
23 using namespace std;
24
25 int main(int argc, char * argv[]) {
26 int s, rc, n;
27 socklen_t len;
28 struct sockaddr_in cliAddr, servAddr;
29 char puffer[BUF];
30 const int y = 1;
31 s = socket (AF_INET, SOCK_DGRAM, 0);
32 if (s < 0) {
33 printf ("%s:␣Kann␣Socket␣nicht␣öffnen␣...(%s)\n");
```

```
34 return 1;
35 }
36
37 /* Lokalen Server Port bind(en) */
38 servAddr.sin_family = AF_INET;
39 servAddr.sin_addr.s_addr = htonl (INADDR_ANY);
40 servAddr.sin_port = htons (LOCAL_SERVER_PORT);
41 setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &y, sizeof(int));
42 rc = bind ( s, (struct sockaddr *) &servAddr,
43 sizeof (servAddr));
44 if (rc < 0) {
45 printf ("%s: Kann Portnummern %d nicht binden (%s)\n");
46 return 1;
47 }
48 printf ("%s: Wartet auf Daten am Port (UDP) %u\n",
49 argv[0], LOCAL_SERVER_PORT);
50 /* Serverschleife */
51 while (1) {
52 /* Puffer initialisieren */
53 memset (puffer, 0, BUF);
54 /* Nachrichten empfangen */
55 len = sizeof (cliAddr);
56 n = recvfrom ( s, puffer, BUF, 0,(struct sockaddr *) &cliAddr, &
    len );
57 if (n < 0) {
58 printf ("%s: Kann keine Daten empfangen ...\n",
59 argv[0] );
60 continue;
61 }
62
63 /* Erhaltene Nachricht ausgeben */
64 printf ("%s \n", puffer);
65
66 }
67 return 0;
68
69 }
```

2.4 Git

Git ist ein dezentrales Versionsverwaltungsprogramm. Es wird kein zentraler Server und es wird kein Internetzugang benötigt. Git ist komplett über die Kommandozeile Steuerbar, mittlerweile existieren aber auch viele Graphische Oberflächen für Git.

Das Paket wird mittels des Befehls

```
sudo apt-get install git
```

installiert werden.

Für das HElicopter Project existiert bereits ein Git-Repository. Dieses Repository wird von Herrn Agrawal verwaltet. Für das Clonen und Pushen des verwalteten Quellcodes muss mit ihm Abgesprochen werden (Quelle/Ziel und Zugang).

Um Quellcode eines Repositorys zu klonen wird der folgende Befehl verwendet:

```
git clone PfadDesRepositories SpeicherOrtDesRepositories
```

Als nächstes sollte man Git bekannt machen wer man ist um zu erkennen wer welche Änderungen gemacht hat. Hierfür geht man in das lokal heruntergeladene Repository Verzeichnis und setzt sich mit den folgenden Befahl seinen Namen und seine EMail Adresse:

```
git config --global user.name MeinName
git config --global user.name MeineEMail@Mail.de
```

Diese Parameter sind bis zur einer Änderung in diesem lokalen Verzeichnis fest.

Anschließend sollte man sich einen Neuen Branch/Zweig erstellen auf welchem anschließend gearbeitet wird. Dies gelingt über das Kommando:

```
git checkout -b MyOwnBranch
```

Möchte man auf einen anderen Branch wechseln muss folgender Befehl verwendet werden:

```
git checkout AnotherBranch
```

Natürlich kann auch so auf ein anderes commit geänder werden.

Um Dateien oder ganze Verzeichnisse in das Quellverwaltung auszunehmen ist das Kommando

```
git add filename.c
```

zu verwenden.

Um Änderungen des Repositorys lokal zu speichern wird das Kommando

```
git commit -m "Message"
```

verwendet. Der Parameter -m sollte immer einen passenden Schlüssigen Namen besitzen. Coder der commitet wird solle zuvor getestet werden oder zumindest keinen kompilerfehler liefern Auf andere commits kann jederzeit zugriffen werden. Mit dem Befehl

```
git log
```

werden alle commits und pushes angezeigt die auf die aktuelle lokale Version zutreffen. Mit den Information des Authoren und der Messsage des commits.

Bearbeitete oder unversionierte Dateien lassen sich mit

```
1 git status -u
```

sichtbar machen.

Sollen Zwei Branches zusammengeführt werden müssen zunächst sämtliche Konflikte aufgelöst werden. Die Änderungen können mit

```
1 git diff MyBranch TargetBranch
```

Angezeigt werden.

Mit dem Befehl

```
1 git merge Targetbranch
```

wird der Branch in dem der Befahl aufgeführt, nach erfolgreicher Konflikt Lösung, wird mit dem angegeben TargetBranch Zusammengeführt.

Nach Pull Files alle als geändert gekennzeichnet Direkt nach dem pull erfolgt eine Überprüfung, ob der pull erfolgreich durchgeführt wurde. Falls der Fehler erst später erkannt wird, kann es zum Verlust von Daten kommen. Es empfiehlt sich eine temporäre lokale Version abzuspeichern. Der Befehl

```
1 git status -s
```

darf nichts zurückgeben, d.h. es gibt keine Änderungen im Vergleich zur Git Version. Wenn fehlerhafteiweise angezeigt wird, dass Änderungen vorgenommen wurden, wird mit

```
1 git -diff
```

angezeigt, welche Änderungen vorgenommen wurden. Nun gibt es zwei verschiedene Möglichkeiten: Wenn ...

... Steuerzeichen Controll M hinzugefügt worden ist, liegt es daran, dass die Files von einem anderen Betriebssystem aus bearbeitet wurden, z.B. von Windows zugegriffen wurde.

Bei Windows basierenden Systemen werden für ein Line Terminator (neuer Absatz und erste Position) zwei Steuerzeichen benötigt (*Carriage Return* (ctrlM) und *New Line Feed*(ctrlJ)). Bei Unix existiert nur das Steuerzeichen *New Line Feed*. Das *Carriage Return* besitzt keine spezielle Bedeutung in Linux basierenden Systemen. Daher wird das *Carriage Return* nicht als Steuerzeichen erkannt und das Zeichen wird an jedes Zeilenende ins File hinzugefügt.

Mit dem Befehl *dos2unix*, welcher eine File-Konvertierung von dos nach unix durchführt, kann das Problem gelöst werden.

Zu beachten hierbei ist, dass die Konvertierung nur an benötigten Daten durchgeführt wird. Sonst könnte sie Verbindung mit Git zerstört werden.

Einzelne Files können wie folgt konvertiert werden:

dos2unix MAIN.c

Wenn mehrere Files konvertiert werden müssen, empfiehlt sich der Befehl find und dessen Ausgabe umzuleiten z.B:

find . -name *.c | xargs dos2unix

Der Befehl sucht alle Files im aktuellen Verzeichnis, die eine .c Endung im Namen tragen und konvertiert diese anschließend.

... wenn keine Unterschiede in den Files angezeigt werden, müssen die folgenden Befehle ausgeführt werden (falls Änderungen vorgenommen wurden ist eine lokale Kopie zu erstellen, sonst sind die Änderungen gelöscht!):

```
1 git rm --cached -r .
2
3 git reset ?-hard
```

Dies löscht rekursiv alle Dateiinhalte von dem Index. Anschließend wird der Index und der Working tree resetet. Jede Änderung seit dem letzen Commit werden rückgängig gemacht.

2.5 Bekannte Probleme

2.5.1 Udp Socket Programm mit Windows 10 und VMware

Wenn sich auf dem Host-System das Betriebssystem Windows 10 befindet, gibt es beim Empfangen der Udp-Packete Probleme. Der Programmcode ist voll funktionsfähig. Dies kann man nachprüfen indem man beide Programme in der VMware über die localhost Adresse laufen lässt.

Die gesendeten Daten kommen nicht bei der VMware an. Außerdem ist die VMware vom Pi nicht anzupingen, deshalb wurde die Firewall temporär deaktiviert. Daraufhin wurde der Ping Befehl erfolgreich ausgeführt. Die Daten kamen weiterhin nicht an.

Mit dem Tool Wireshark wurde mit aktiver und deaktivierter Firewall die Kommunikation mitgeschnitten. Bei deaktivierter Firewall kam die ICMP-Meldung, dass der Port nicht erreichbar ist. Bei aktiver Firewall kamen diese Nachrichten nicht vor.

Daraufhin wurden für die Firewall Ausnahmen generiert, das jegliche Kommunikation von der IP des Raspberry Pi auf den Port 9999 ankommt und nicht blockiert wird.

Da dies ebenfalls keine Auswirkung hat, wird wohl noch in einem von Windows bereitgestellten Dienst/Service die Kommunikation verhindert. Auf Funktion mit einem anderen Windows Betriebssystem können keine Annahmen getroffen werden.

Mit einem Linux/Ubuntu System funktionieren die Programme fehlerfrei.

Als Ausweichmöglichkeit wurde ein PC für folgende Arbeit in der Hochschule bereitgestellt.

3 Realisierung

Die Hardware kann bei Bedarf in einem abschlieï¿½barem Container gelagert werden und nach Absprache mit Projektteilnehmer und Projektleiter die Hardware auch mitgenommen werden.

3.1 Inbetriebnahme der Hardware

Eine MicroSD-Karte mit aufgespieltem Ubuntu-Image muss in den Raspberry Pi Kartenslot gesteckt werden.

Als Versorgungsspannung benï¿½tigt der Quadrocopter 11.7 Volt mit Akku oder 10V-11V mit Netzgerï¿½t. Beim Umlegen des Hauptschalters sollen die LEDï¿½s am Quadrocopter und PI blinken sowie die Motoren einen Impuls erhalten.

Beim Motorentests sollte der Quadrocopter befestigt werden, damit dieser nicht beschï¿½digt wird geht. Im EZS-Labor wurde hierfiï¿½r an einem Arbeitsplatz ein Schraubstock montiert.

Der I2C-Bus (braunes rot/blaues Kabel) wird mit den I2C Pins des Raspberry PI B+ verbunden. Pin2 ist Serial Data und Pin5 ist Serial Clock. Braun-blaues Kabel ist fiï¿½r das Clocksignal , braun-rotes Kabel ist fiï¿½r das Datensignal. Testen kann man die I2C Configuration des PI mit dem Befehl:

sudo i2cdetect -y 1

In der nachfolgenden Tabelle finden sich alle beinhalteten Komponenten des I2C1-Buses wieder.

Typ	I2C Adresse 7bit	Information
ADC	0x49	ADDR <-> VDD
BeschleunigungssensorMagnetometer	0x1E	SA0 <-> GND
Barometer	0x5C	SA0 <-> GND
Gyrometer	0x6A	SA0 <-> GND
Brushless Motoren Treiber 1-4	0x29,0x2A, 0x2B,0x2C	Je nach HElicopter Typ unterschiedlich, I2C Adressen sind per HW festgelegt , per Software nicht ï¿½nderbar
Brushless Motoren Treiber 5-8	0x5A,0x5C, 0x5E,0x60	Nicht getestet, zu ï¿½berprï¿½fen und ggf. zu ï¿½berarbeiten in der *MOTOR.h*
Laser Sensor	0x62	Standardadresse des Herstellers

Tab. 3.1: I2C Adsressvergabe

Der Remoteempfï¿½nger GR-16 hat drei Verbindungen zum Raspberry Pi: Eine 3,6V bis 8,4V Spannungsversorgung, Ground und eine Datenverbindung.

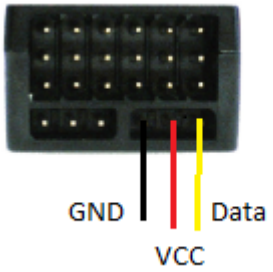


Abb. 3.1: GR-16 Verkabelung[2]

Die Spannung und Ground kann direkt ï¿½ber das Raspberry Pi bezogen werden. Fï¿½r die Daten muss einer der GPIO Pins als Input deklariert werden.

Um die Fernbedienung mx-20 mit dem Empfï¿½nger GR-16 zu verbinden mï¿½ssen beide Gerï¿½te eingeschaltet sein (Empfï¿½nger LED blinkt Rot). Beim Anschalten der Fernbedienung wird gefragt ob *HF* EIN oder AUS ist. Stellen Sie auf AUS und bestï¿½tigen Sie mit Drï¿½cken der Set-Taste. Drï¿½cken sie erneut Set um in die Einstellungen zu gelangen. Scrollen Sie sich mit den Pfeiltasten durch das Menï¿½ bis Sie das Menï¿½

2 http://www.graupner.de/mediaroot/files/33508_Kurzanleitung_de.pdf

Grundeinstellungen Mod. sehen. ï¿½ffnen Sie mit Set das Menï¿½ und scrollen Sie durch bis zu Punkt *Modul*. Bestï¿½tigen Sie noch NICHT mit der Set-Taste. Halten Sie nun am Empfiï¿½nger solange die Set Taste gedrï¿½ckt, bis sich zur der roten LED noch eine grï¿½ne LED einschaltet. Betiï¿½tigen Sie jetzt den Set-Taster auf der Fernbedienung. Es sollte der Info-Text *Binden ...* angezeigt werden. Wenn die Verbindung erfolgreich durchgefiï¿½hrt wurde leuchtet die LED dauerhaft grï¿½n. [1]

3.2 Debuggen der Programme

Target laden Rechtsklick auf Projekt Ordner,
Build Configurations -> Set Active -> Target.
Rechtsklick auf Projekt Ordner, Build Project.

Debug Pfeil anklicken und helicopter-raspberry target auswï¿½hlen.

Oder ï¿½ber das Terminal: In das Verzeichnis *helikopter-raspberry/impl/trunk/* wechseln und mit dem Befehl
make MAKECMDGOALS=target
das Programm kompilieren und automatisch auf das Raspberry Pi laden.

Host laden Rechtsklick auf Projekt Ordner, Build Configurations -> Set Active -> Host. Rechtsklick auf Projekt Ordner, Build Project.

Debug Pfeil anklicken und READ-UDP auswï¿½hlen.

Programme laufen lassen In die Debug-Ansicht von Eclipse wechseln (oben rechts). In dem Debug-Fenster sollten nun eine Remote-Anwendung (Anwendung auf Pi) und eine Host-Anwendung gestartet sein. Um diese laufen zu lassen muss der jeweilige Prozess ausgewï¿½hlt werden und auf Resume geklickt werden sowie ggf. Breakpoints entfernt werden.

Nachdem die beiden Programme laufen muss noch der zu startende Testfall ausgewï¿½hlt werden. Dies geschieht, in dem man ï¿½ber die Console die jeweiligen Testfall auswï¿½hlt, z.B. "testmatlabimu". Das im Grundlagenteil Udp-Programm ist hier als Testfall eingefï¿½gt und kann mit der Eingabe "testudp"gestartet werden.

Nun laufen die Programme. Falls im ausgewï¿½hlten Testfall Ausgaben vorhanden sind, sind diese in der Console zu sehen.

Beenden der Applicationen Um Probleme zu vermeiden ist es wichtig, die laufenden Programme immer manuell anzuhalten und zu *killen*. Dies gelingt, in dem man in der Debug-Ansicht auf das zu beendene Programm rechtsklickt und auf Terminate klickt.

Testfiï¿½lle In der main.c wird in dem definiertem *enum* enumTestCases ein neuer Testfall angelegt. In der int main()... wird bei Start auf eine Eingabe des zu startenden Testfalles gewartet. Hier muss noch ein *else if*-Anweisung ergï¿½nzt werden. In der Variable runCommand wird der Name des abzuarbeitenden Testfalles abgespeichert. In der nachfolgenden *Switch*-Anweisung steht der eigentliche Code, welcher bei Auswahl dieses Testfalles ausgefiï¿½hrt wird.

3.3 Umstellung auf eine automatische dynamische Testumgebung

Die Main-Funktion zum Testen soll umstrukturiert werden, da die vorherige Struktur recht umstï¿½ndlich zu testen und zu bedienen war. Es muss vor jedem einzelnen Testfall das Programm komplett auf das Raspberry ï¿½bertragen werden und ï¿½ber die Remote-Verbindung der zu startende Testfall hï¿½ndisch ausgewï¿½hlt werden. Dieser wird ausgefï¿½hrt und das Programm ist beendet, sobald der Testfall beendet ist. Um einen weiteren Test zu starten musste wieder eine Remote-Verbindung aufgebaut werden. Dies funktioniert, ist aber nicht die optimale Lï¿½sung.

Stattdessen soll die neue Main-Funktion ï¿½ber eine Dauerschleife verfiï¿½gen, in dieser ein Textfile ausgelesen wird. Dieses File beinhaltet die zu startenden Testcases des Programmes.

In diesem Textfile werden alle Testfiï¿½lle festgehalten, mit einem Status ob dieser laufen soll oder nicht. Der Status soll wï¿½hrend der Laufzeit des Programmes ï¿½nderbar sein und dieses auch nicht unterbrechen.

Dieses Textfile sieht folgendermaï¿½en aus:

```
...
testmotorpwm=1
testmotorisr=1
testmotortxt=0
....
```

Dies bedeutet: die Testfiï¿½lle testmotorpwm und testmotorisr sollen gestartet werden, testmotortxt soll nicht gestartet werden. Der Testfall testmotorpwm wird zuerst ausgelesen und von der Main-Funktion auf Null zurï¿½ckgesetzt. Anschlieï¿½end lï¿½uft der Testfall ab.

Wï¿½hrenddessen wird der Testfall testmotortxt aktiviert.

```
...
testmotorpwm=0
testmotorisr=1
testmotortxt=1
....
```

Ist der Testfall beendet wird das Textfile erneut eingelesen und der nï¿½chste gesetzte Testfall wird gestartet.

Umstrukturierung der main.c: Es wurde eine Endlosschleife eingefi¿½gt. Statt auf eine Tastatureingabe i¿½ber eine Konsole zu warten, wird in dieser alle zwei Sekunden das *_Testfile* ausgelesen und i¿½berpri¿½ft, ob ein Testfall ausgefi¿½hrt werden soll.

In allen Testfi¿½llen darf keine Endlosschleife vorhanden sein, bzw. muss in weiteren Endlosschleifen ein Abbruchkriterium festgelegt sein. In den meisten Fi¿½llen kann der Testfall mit Dri¿½cken der Taste *p* abgebrochen werden. In Testfi¿½llen, in denen ein Testfile ausgelesen wird, wird bei Erreichen des Dateiendes der Testfall beendet und gelangt so wieder in die Hauptschleife zuri¿½ck.

Script zum Setzen der Testfi¿½lle: Zum Setzen oder Li¿½schen eines Testfiles wurde ein Script geschrieben, welches auf das textfile *_Testfile* zugreift und die Eintri¿½ge je nach Eingabe i¿½berarbeitet.

Verwendet wird das Script folgendermai¿½en:

```
1 .\testcase NameTestfall [Modus]
```

Als ersten Parameter ist der zu i¿½berarbeitende Testfall anzugeben.
Der Modus ist ein optionaler Parameter. Ist dieser Parameter nicht vorhanden, wird automatisch *set* angenommen. *set* markiert den Testfall als abzuarbeiten, *clear* li¿½scht diese Markierung wiederrum. Das Script zeigt keinen Fehler oder Warnung auf, wenn der testcase nicht gelistet ist.

Der Quellcode ist im Kapitel A.1 Script *'testcase'* auf S.57 einzusehen.

3.4 Valedierung der empfangenen Werte

Im testcase testmatlabimu stimmen die empfangenen und gesendeten Werte nicht i¿½berein. Hier ist der Fehler zu finden und zu beheben.

Es werden insgesamt 11 double Values von den diversen Sensoren an den Host gesendet. Die Kalkulation der Werte ist richtig, kommen jedoch beim Empfi¿½nger falsch an. Dieses Fehlverhalten gilt es zu untersuchen.

Hierfi¿½r wurde ein neuer Testfall angelegt, alle 11 Double Werte per UDP versendet und diese auf beiden Seiten ausgegeben.

Nach dem Setzen der Breakpoints, so das nur einmal die Werte gesendet werden, erhalten wir folgende Ausgaben auf der Console:

```
1  Starting read over UDP
2  /home/ezs/git/helikopter-
     raspberry/impl/trunk/host
     /READ-UDP.elf: Wartet auf
     Daten am Port (UDP) 5000
3  Acc X 0.000000
4  Acc Y 0.910135
5  Acc Z -0.488599
6  Mag X -10.533617
7  Mag Y 0.000024
8  Mag Z 0.000021
9  Gyro yaw 0.000014
10 Gyro pitch -0.473037 nGyro
     roll 0.473037
11 Temp -0.396741
12 Press 31.825000
```

```
1  Received string is
     testallsensordata
2  Starting IMU Matlab Test
3
4
5  Acc X 0.910135
6  Acc Y -0.488599
7  Acc Z -10.533617
8  Mag X 0.000024
9  Mag Y 0.000021
10 Mag Z 0.000014
11 Gyro yaw -0.473037
12 Gyro pitch 0.473037
13 Gyro roll -0.396741
14 Temp 31.825000
15 Press 987.931396
```

Nach einem Vergleich fi¿½llt auf, dass beim Empfang der Daten sich bei X-Wert vom Acc eine Null eingefi¿½gt hat. Ansonsten scheinen die Daten zu stimmen. Die Daten sind nur um eins versetzt.

In der read-udp-host.c werden die Werte in der Variable *l_recvImuState_st* gespeichert. Beim Betrachten der Inhalte dieser Variablen ist zu erkennen, dass diese in *acc.f64* einen Wert hat der Circa null entspricht, die restlichen Werte sehen richtig aus, nur weiterhin um einen Wert verschoben.

Verdacht: Beim Senden der Daten wird auch noch ein Zeitstempel dieses Telegramms mitgesendet:

```
//time.h
struct timespec{
  __time_t tv_sec; /* Seconds */
  __syscall_slong_t tv_ns /* Seconds */
  };

//udpImuLib.c
struct timespec l_timespec_st;
```

Diese Daten haben die Lï¿½nge von 8 Byte (beide vom Typ long int). Diese Daten werden mit den Sensordaten gesendet. Die Paketlï¿½nge nimmt zu. Da auf der Empfiï¿½ngerseite die Empfangsstruktur aber diesen timestamp nicht erwartet, geht er davon aus, dass der erste Wert, den er bekommt, fiï¿½r den Acc X Sensor-Wert steht. Aus diesem Grund verrutschen die restlichen Daten um eins ab und der erste Sensor-Wert ist falsch da dieser den Timestamp widerspiegelt.

Problembehebung Die fehlenden Daten mïï¿½ssen in dem Struct bekannt gemacht werden, sodass die Lï¿½nge der beiden Felder nun gleich Groïï¿½ sind (96 Byte). Hierfiï¿½r musste das Struct *halImu_orientationValues* im imu.h um einen Datentyp struct timespec erweitert werden sowie die notwendige Standardlibary *time.h* hinzugefiï¿½gt werden. Dann wurde die Testausgabe um den Timestamp erweitert. Die Werte stimmen nun ïï¿½berein:

```
  Starting read over UDP
  /home/ezs/git/helikopter -
     raspberry/impl/trunk/host /
     READ-UDP.elf: Wartet auf
     Daten am Port (UDP) 5000
  Time  1434105267.000000000
  Acc X 0.792776
  Acc Y -0.555661
  Acc Z -10.581519
  Mag X 0.000024
  Mag Y 0.000021
  Mag Z 0.000014
  Gyro yaw -0.228889
  Gyro pitch -0.122074
  Gyro roll -0.488296
  Temp 32.056250
  Press 987.923828
```

```
   Received string is
      testallsensordata
    Starting IMU Matlab Test


  Acc X 0.792776
  Acc Y -0.555661
  Acc Z -10.581519
  Mag X 0.000024
  Mag Y 0.000021
  Mag Z 0.000014
  Gyro yaw -0.228889
  Gyro pitch -0.122074
  Gyro roll -0.488296
  Temp 32.056250
  Press 987.923828
```

```
#include <time.h>
typedef struct{
  struct timespec l_timestamp_st;
  halAccmag_3dDoubleVector acc;
```

```
5   halAccmag_3dDoubleVector mag;
6   strGyro gyro;
7   double temperature_f64;
8   double pressure_f64;
9 } halImu_orientationValues;
```

3.5 Software Motortreiber

Es soll Software fi¿½r den Hardware-Treiber, der die Motoren ansteuert, geschrieben werden. Zunï¿½chst muss der Raspberry Pi mit dem I2C-Bus verbunden werden. Die I2C Adressen der Motoren sind in der Motor.h hinterlegt. Fï¿½r die Quadrocopter wurden die Werte definiert. Die Werte des Octocopter mï¿½ssen zunï¿½chst ï¿½berprï¿½ft werden.

Mittels des Befehl:

i2cdetect -y 1

kï¿½nnen die vergebenen Adressen im Bussystem angezeigt und validiert werden.

Die Daten, die von dem Hardware Brushless-Controller erwartet werden, besitzen folgendes Format:

I2C Adresse [1 Byte]	PWM Value [1 Byte]

Tab. 3.2: I2C Frame Brushless Motoren Treiber

Mit Hilfe des Befehls

i2cset -y 1 0x29 0x55

wird an den Controller mit der I2C Adresse 0x29 (Motor Nr.1) der Wert 0x55 gesendet.

3.5.1 Verwendung des Software-Treibers

In der main.h muss zunï¿½chst definiert werden, auf welchem Typ von den HElicoptern das Programm geladen werden soll.

```
1 #include <time.h>
2 #define Quadro_Plus 1
3 //#define Quadro_X 1
4 //#define Okto_Plus 1
```

Anschlieï¿½end sollte baldmï¿½glichst die InitMotor() aufgerufen werden, die unter anderem einen Timer initialisiert und startet. Bei Ablauf des Timers wird ein Flag gesetzt. Dieses Flag muss im Quellcode mit der Funktion *GetFlagRunSendPwmToMotor()* abgefragt werden. Wenn dieses Flag gesetzt ist, muss die Funktion *sendPwmToMotor()* aufgerufen werden.

```
...
InitMotor();
...
while(1){
...
if(GetFlagRunSendPwmToMotor() == 1){
sendPwmToMotor();
}
...

}
```

Der aktuelle PWM-Wert eines Motors kann mittels der Funktion *GetPwmMotor(...)* zurï¿½ckgegeben werden.

```
value = GetPwmMotor(6);
value > 0? value--: (value=DEFMotorSetpointMIN);
SetPwmMotor(DEFMotorNo7_PWM, value ,0);
```

Wichtig: Alle ISR sollen knapp gehalten werden, da ansonsten die Motoren nicht mehr angesprochen werden kï¿½nnen.

Mit der Funktion *SetPwmMotor(...)* kï¿½nnen die PWM-Werte, die per I2C gesendet werden, ï¿½berschrieben werden. Optional kann ein Flag gesetzt werden. Ist dies der Fall wird anschlieï¿½end die Funktion *sendPwmToMotor()* aufgerufen.

3.5.2 Headerfile

Hier sind die verschiedenen HElicopter Varianten sowie deren definierte Eigenschaften (Anzahl Motoren, Drehrichtung der Motoren, Motorenreihenfogen) festgehalten.

Fï¿½r weitere Informationen (wie z.B. Namen der defines) siehe in Kapitel A in Code auf S.53 .

3.5.3 Funktionen

Beschreibung der Funktionen befinden sich in den jeweiligen darï¿½ber liegenden Kommentaren mit Parametern und Return Values.

Auf alle globalen Variablen/Flags werden mit Funktionen zugegriffen. Ein direkter Zugriff ist zu vermeiden.

```
14 /* Global Variables */
15 char BLCtrlADRExecuteOrder[DEFMotorsCount];
16 char PWMValue[DEFMotorsCount];
17
18 //Flags
19 char flagRunSendPwmToMotor;

21 /*!*****************************************************************
22 * \author Chris Mï¿½nch( chmoit00 )
23 *
24 * \brief calls init functions which needed for the motor driver:
25 * SetFlagRunSendPwmToMotor(0);
26 * SetMotorExecutionOrder();
27 * SetPwmMotor(DEFMotorALL_PWM, DEFMotorSetpointMIN, 0);
28 * Last one always initMotorTimer()
29 * InitMotorTimer(microSeconds);
30 * SetFlagRunSendPwmToMotor(1);
31 *
32 * \param[ in ] microSeconds - Time in uS when Timer expired.
33 *
34 * \internal
35 * CHANGELOG:
36 *
37 * \endinternal
38 *****************************************************************/
39 void InitMotor(int microSeconds){
40   SetFlagRunSendPwmToMotor(0);
41   SetMotorExecutionOrder();
42   SetPwmMotor(DEFMotorALL_PWM, DEFMotorSetpointMIN, 0);
43   //Last one always initMotorTimer()
44   InitMotorTimer(microSeconds);
45   SetFlagRunSendPwmToMotor(1);
46 }

48 /*!*****************************************************************
49 * \author Chris Mï¿½nch( chmoit00 )
50 * \date 2016/01/08
51 *
52 * \brief set Motor Exectution Order
53 *
54 * \internal
55 * CHANGELOG:
56 *
57 * \endinternal
58 *****************************************************************/
```

```
59 void SetMotorExecutionOrder(){
60   GetBLCtrlADRExecuteOrder(&BLCtrlADRExecuteOrder[0]);
61 }

63 /*!*****************************************************************
64 * \author Chris Mï¿½nch( chmoit00 )
65 * \date 2016/01/08
66 *
67 * \brief sets PWM Signal of selected Motor to pwmValue
68 * \details toSet = 00001111 sets the first 4 Motors in Execution
       Order to pwmValue
69 *
70 * \param[ in ] toSet - Which Motor to Set
71 * \param[ in ] pwmValue - Which Value so Set
72 * \param[ in ] forceSend - optional Parameter if !0
       flagRunSendPwmToMotor will be set
73 *
74 * \internal
75 * CHANGELOG:
76 *
77 * \endinternal
78 *******************************************************************/
79 void SetPwmMotor(char toSet , int pwmValue, int forceSend){
80   int i=0;
81   pwmValue = pwmValue >= DEFMotorSetpointMIN ? pwmValue :
         DEFMotorSetpointMIN;
82   pwmValue = pwmValue <= DEFMotorSetpointMAX ?  pwmValue :
         DEFMotorSetpointMAX;
83   while(toSet != 0 && i < DEFMotorsCount){
84
85     if(toSet%2){
86       PWMValue[i]= pwmValue;
87     }
88     toSet= toSet >>1;
89     i++;
90   }
91   if(forceSend != 0){
92     SetFlagRunSendPwmToMotor(1);
93   }
94 }

96 /*!*****************************************************************
97 * \author Chris Mï¿½nch( chmoit00 )
98 * \date 2016/01/08
99 *
100 * \brief adds to the current PWM Signal of selected Motor the
       pwmValue
```

```
101 * \details toSet = 00001111 add to the first 4 motors pwmValue
102 *
103 * \param[ in ] toSet - Which Motor to Set
104 * \param[ in ] pwmValue - adding pwm value to current PWMValue
105 * \param[ in ] forceSend - optional Parameter if !0
       flagRunSendPwmToMotor will be set
106 *
107 * \internal
108 * CHANGELOG:
109 *
110 * \endinternal
111 ****************************************************************/
112 void AddPwmMotor(char toSet , int pwmValue, int forceSend){
113   int i=0;
114
115   while(toSet != 0 && i < DEFMotorsCount){
116
117     if(toSet%2){
118       pwmValue = pwmValue+GetPwmMotor(i);
119       pwmValue = pwmValue >= DEFMotorSetpointMIN ? pwmValue :
             DEFMotorSetpointMIN;
120       pwmValue = pwmValue <= DEFMotorSetpointMAX ?  pwmValue :
             DEFMotorSetpointMAX;
121       PWMValue[i]= pwmValue;
122     }
123     toSet= toSet >>1;
124     i++;
125   }
126   if(forceSend != 0){
127     SetFlagRunSendPwmToMotor(1);
128   }
129 }

131 /*!****************************************************************
132 * \author Chris Mï¿½nch( chmoit00 )
133 * \date 2016/01/08
134 *
135 * \brief Subtract to the current PWM Signal of selected Motor the
       pwmValue
136 * \details toSet = 00001111 subtract to the first 4 motors
       pwmValue
137 *
138 * \param[ in ] toSet - Which Motor to Set
139 * \param[ in ] pwmValue - pwm value to subtract from Current
       PWMValue
140 * \param[ in ] forceSend - optional Parameter if !0
       flagRunSendPwmToMotor will be set
```

```
141 *
142 * \internal
143 * CHANGELOG:
144 *
145 * \endinternal
146 ****************************************************************/
147 void SubbPwmMotor(char toSet , int pwmValue, int forceSend){
148   int i=0;
149
150   while(toSet != 0 && i < DEFMotorsCount){
151
152     if(toSet%2){
153       pwmValue = GetPwmMotor(i)- pwmValue;
154       pwmValue = pwmValue >= DEFMotorSetpointMIN ? pwmValue :
              DEFMotorSetpointMIN;
155       pwmValue = pwmValue <= DEFMotorSetpointMAX ?  pwmValue :
              DEFMotorSetpointMAX;
156       PWMValue[i]= pwmValue;
157     }
158     toSet= toSet >>1;
159     i++;
160     }
161   if(forceSend != 0){
162     SetFlagRunSendPwmToMotor(1);
163   }
164 }

167 /*!****************************************************************
168 * \author Chris Mï¿½nch( chmoit00 )
169 * \date 2016/01/08
170 *
171 * \brief Gets pwmValue from a specific motor
172 * \details
173 *
174 * \param[ in ] motorNumber - which motor
175 *
176 * \param[ out ] pwmValue of the chosen Motor, returns 0 if chosen
     Motor not exist in these HElicoptertype
177 *
178 * \internal
179 * CHANGELOG:
180 *
181 * \endinternal
182 ****************************************************************/
183 int GetPwmMotor(int motorNumber){
184   return motorNumber < DEFMotorsCount ? PWMValue[motorNumber]: 0;
185 }
```

```
188 /*!*************************************************************
189 * \author Chris Mï¿½nch( chmoit00 )
190 * \date 2016/01/08
191 *
192 * \brief init Timer for the IsrMotor
193 * \details
194 *
195 * \param[ in ] microSeconds - Time in uS when Timer expired.
196 *
197 * \internal
198 * CHANGELOG:
199 *
200 * \endinternal
201 **************************************************************/
202 void InitMotorTimer(int microSeconds){
203
204   struct sigaction sa;
205   struct itimerval timer;
206
207   //Creates Signal, if signal Rising a_handler called
208   memset(&sa, 0 , sizeof(sa));
209   sa.sa_handler = &IsrSetFlag;
210   sigaction(SIGVTALRM, &sa, NULL);
211
212   //Expire the Timer after:
213   timer.it_value.tv_sec = 0;
214   timer.it_value.tv_usec = 0;
215   //And every ... after that:
216   timer.it_interval.tv_sec = 0;
217   timer.it_interval.tv_usec = microSeconds;
218   //upon expiration the signal SIGVTALRM raised
219   setitimer(ITIMER_VIRTUAL, &timer , NULL);
220 }

222 /*!*************************************************************
223 * \author Chris Mï¿½nch( chmoit00 )
224 * \date 2016/01/08
225 *
226 * \brief set flag flagRunSendPwmToMotor
227 *
228 * \param[ in ] 1 Set Flag, else clear Flag
229 *
230 * \internal
231 * CHANGELOG:
232 *
233 * \endinternal
234 **************************************************************/
```

```
235 void SetFlagRunSendPwmToMotor(char value){
236   if(value == 1){
237     flagRunSendPwmToMotor=value;
238   }else{
239     flagRunSendPwmToMotor=0;
240   }
241 }

243 /*!*****************************************************************
244 * \author Chris Mï¿½nch( chmoit00 )
245 * \date 2016/01/08
246 *
247 * \brief ISR for set flag as flagRunSendPwmToMotor
248 *
249 * \internal
250 * CHANGELOG:
251 *
252 * \endinternal
253 *****************************************************************/
254 void IsrSetFlag(){
255   flagRunSendPwmToMotor=1;
256 }

258 /*!*****************************************************************
259 * \author Chris Mï¿½nch( chmoit00 )
260 * \date 2016/01/08
261 *
262 * \brief get flag flagRunSendPwmToMotor
263 *
264 * \param[out] flag flagRunSendPwmToMotor
265 *
266 * \internal
267 * CHANGELOG:
268 *
269 * \endinternal
270 *****************************************************************/
271 char GetFlagRunSendPwmToMotor(){
272   return flagRunSendPwmToMotor;
273 }

275 /*!*****************************************************************
276 * \author Chris Mï¿½nch( chmoit00 )
277 * \date 2016/01/08
278 *
279 * \brief sends every timer interrupt to the motors the specific
       pwm values
280 *
```

```
281 * \internal
282 * CHANGELOG:
283 *
284 * \endinternal
285 ***********************************************************************/
286 void sendPwmToMotor(){
287   int i;
288   for(i = 0; i < DEFMotorsCount ;i++)
289   {
290     g_lldI2c_WriteI2c_bl(BLCtrlADRExecuteOrder[i],&PWMValue[i],1);
291   }
292 }

294 /*!***********************************************************************
295 * \author Chris Mï¿½nch( chmoit00 )
296 * \date 2016/01/08
297 *
298 * \brief Get the I2C addresses orderd by execution (defined in
      MOTOR.h)
299 * \details
300 *
301 * \param[ in ] Array where the I2C Addresses will be stored
302 *
303 * \internal
304 * CHANGELOG:
305 *
306 * \endinternal
307 ***********************************************************************/
308 void GetBLCtrlADRExecuteOrder(char BLCtrlADRExecuteOrder[]){
309 #if defined(Quadro_X) || defined(Quadro_Plus)
310 int BLCTRLADR[4] = {DEFMotorNo1_BLCtrlADR, DEFMotorNo2_BLCtrlADR,
      DEFMotorNo3_BLCtrlADR, DEFMotorNo4_BLCtrlADR};
311
312 BLCtrlADRExecuteOrder[DEFMotorNo1_OrderIDX ]=BLCTRLADR[0];
313 BLCtrlADRExecuteOrder[DEFMotorNo2_OrderIDX]=BLCTRLADR[1];
314 BLCtrlADRExecuteOrder[DEFMotorNo3_OrderIDX]=BLCTRLADR[2];
315 BLCtrlADRExecuteOrder[DEFMotorNo4_OrderIDX]=BLCTRLADR[3];
316
317 #endif
318
319 #ifdef Okto_Plus
320 int BLCTRLADR[8] = {DEFMotorNo1_BLCtrlADR, DEFMotorNo2_BLCtrlADR,
      DEFMotorNo3_BLCtrlADR
321 DEFMotorNo4_BLCtrlADR, DEFMotorNo5_BLCtrlADR,
      DEFMotorNo6_BLCtrlADR,
322 DEFMotorNo7_BLCtrlADR, DEFMotorNo8_BLCtrlADR};
323
```

```
324 BLCtrlADRExecuteOrder[DEFMotorNo1_OrderIDX]=BLCTRLADR[0];
325 BLCtrlADRExecuteOrder[DEFMotorNo2_OrderIDX]=BLCTRLADR[1];
326 BLCtrlADRExecuteOrder[DEFMotorNo3_OrderIDX]=BLCTRLADR[2];
327 BLCtrlADRExecuteOrder[DEFMotorNo4_OrderIDX]=BLCTRLADR[3];
328 BLCtrlADRExecuteOrder[DEFMotorNo5_OrderIDX]=BLCTRLADR[4];
329 BLCtrlADRExecuteOrder[DEFMotorNo6_OrderIDX]=BLCTRLADR[5];
330 BLCtrlADRExecuteOrder[DEFMotorNo7_OrderIDX]=BLCTRLADR[6];
331 BLCtrlADRExecuteOrder[DEFMotorNo8_OrderIDX]=BLCTRLADR[7];
332
333 #endif
334 }
```

3.5.4 Testfi¿½lle

Um die Funktionaliti¿½t des Treiber zu testen und kontrollieren sind die folgenden Testfi¿½lle geschrieben worden.

TESTMOTORPWM Dieses Testprogramm sendet alle 10ms an die Motoren einen stets steigenden PWM-Wert bis zum Wert 0x50. Bei erreichen des Wertes wird der PWM-Wert auf den definiertes Minimum gesetzt. Dieser Testfall li¿½uft ohne ISR ab. Die Daten werden direkt i¿½ber I2C gesendet. Die Schrittweite und das Maximum des Testfalles sind per Konstanten definiert und ki¿½nnen veri¿½ndert werden.

TESTMOTORISR Mit diesem Testprogramm wird die ISR-Funktion getestet. Durch Eingabe in der Konsole wie z.B.

+0+0+0+0+7+7+7-8

wird der PWM-Wert des Motors 0 um vier erhi¿½ht, der Motor 7 um drei erhi¿½ht und der Motor 8 um eins verringert. Die Eingabe ist nicht blockierend. Zum Starten des Testfalls muss ein '+' eingetippt werden.

TESTMOTORTXT Der letzte Testfall liest aus einem Textfile, das sich auf dem Raspberry unter dem Pfad */home/pi/MotorTest.txt* befinden muss, Zeile fi¿½r Zeile aus und setzt die PWM-Werte so, wie sie in der Zeile angegeben sind.

Die Befehlszeile hat folgendes Format:

#MOTORNUMER[+][-][=][PWMWERT] DELAY

z.B. #0+100;10 ⌢D̂er PWM des Motors 0 erhi¿½ht sich um 100. Die ni¿½chste Zeile wird in 10s eingelesen.

3.6 Help Functions

Nicht blockierende Eingabe Funktion kbhit Diverse Testfi¿½lle hï¿½ngen von einer Tastatur Eingabe ab. Die Standardfunktionen hierfi¿½r sind blockierende Funktionen. Diese dï¿½rfen jedoch nicht verwendet werden, da die Motoren nicht angesteuert werden kï¿½nnen. Um Tastatureingaben mï¿½glich zu machen, wurde eine Funktion geschrieben, die den Programmablauf nicht blockiert. Als Return-Wert gibt es immer die zuletzt gedrï¿½ckte Taste zurï¿½ck.

Eine Eingabe, die aus mehreren Chars, besteht muss mit mehreren *kbhit()* aufrufen und durch Schleifen oder if-Verzweigung ï¿½berprï¿½ft werden. Die Eingabe wird aus dem Buffer gelesen.

```
0  #include <termios.h>
1
2  int kbhit(void)
3  {
4    struct termios term, oterm;
5    int fd = 0;
6    int c = 0;
7    tcgetattr(fd, &oterm);
8    memcpy(&term, &oterm, sizeof(term));
9    term.c_lflag = term.c_lflag & (!ICANON);
10   term.c_cc[VMIN] = 0;
11   term.c_cc[VTIME] = 1;
12   tcsetattr(fd, TCSANOW, &term);
13   c = getchar();
14   tcsetattr(fd, TCSANOW, &oterm);
15   return c;
16 }
```

3.7 Remotecontroller-Treiber

Um den autonomen Start bzw. Landevorgang des Quadrocopter einzuleiten muss eine Funkverbindung ï¿½ber eine Fernbedienung und einen Transmitter eingerichtet werden. Die Daten sind anschlieï¿½end auszuwerten und zu verarbeiten.

Als Fernbedienung von der Firma Graupner dient der *MX-20* und als Receiver der *GR-16*. Der Anschluss des Receivers erfolgt ï¿½ber drei Leitungen:

- Versorgungsspannung: 5Volt (vom Raspberry)
- Ground (Raspberry)
- PPM (Puls-Pause Modulation) (auf einen der GPIO Pins)

Verbinden mit Receiver Schalten Sie die Fernbedienung ein und schlieï¿½en den Receiver an den Raspberry Pi an [siehe Abschnitt 3.1 auf Seite 24]

PPM-Signal genannt Puls-Pausen-Modulation (oder auch Puls-Position-Modulation) ist ein fiï¿½r analoge Werte verwendetes Kodierungsverfahren und wird vor allem in Funkfernsteuerungen verwendet. Der zu kodierende Wert wird in der Liï¿½nge des Pausen/Low Signals zwischen zwei Peaks/High Signalen gesendet. Diese Peaks haben stets die gleiche Liï¿½nge als auch gleiche Amplitude.[2]

Meist besteht ein PPM-Signal aus mehreren Kanïï¿½len zu einem Frame zusammengefiï¿½gt und anschlieï¿½end versendet.

Fiï¿½r die Decodierung des PPM-Signals ist es empfehlen, ein weiteres Board zu integrieren, das die Decodierung iï¿½bernimmt und die Ergebnisse an das Raspberry Pi weiterleitet, da das Signal sehr genau aufgeliï¿½st werden muss und die Decodierung iï¿½ber einen Interrupt gesteuert werden soll. Bei zu hiï¿½ufigen Auftreten des Interrupts wiï¿½rde der Helicopter destabilisiert werden. Zur Auswertung miï¿½ssen nur die Raising Edges oder Falling Edges beachtet und der Offset des High-Pegels abgezogen werden.

Um auf ein weiteres Bauteil zu verzichten wurde stattdessen eine Real-Time-Linux-Version auf dem Raspberry Pi installiert, die die Signale im Nanosekundenbereich dekodieren kann.

PPM Signal Mitschnitte Der Receiver sendet die empfangenen Signale im Format der Pulse-Pausen-Modulation. Mitgeschnittene ï¿½bertragungen[3]mit Beschreibung sind beigefi¿½gt.

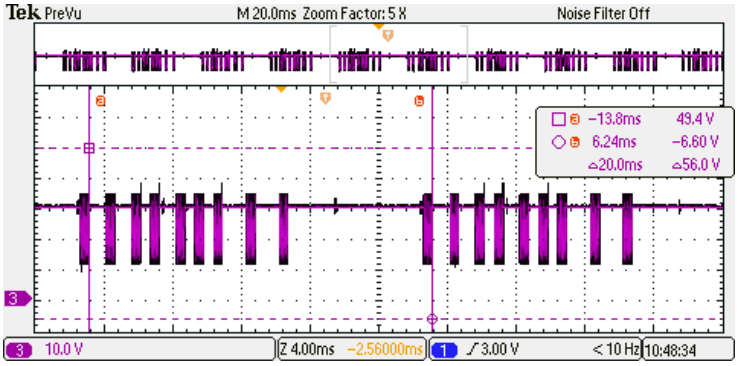


Abb. 3.2: 8 Channel Frame

Es gibt zwei Mï¿½glichkeiten: ein Frame mit acht Kanï¿½len, bestehend aus neun Peaks und acht Pausensignalen...

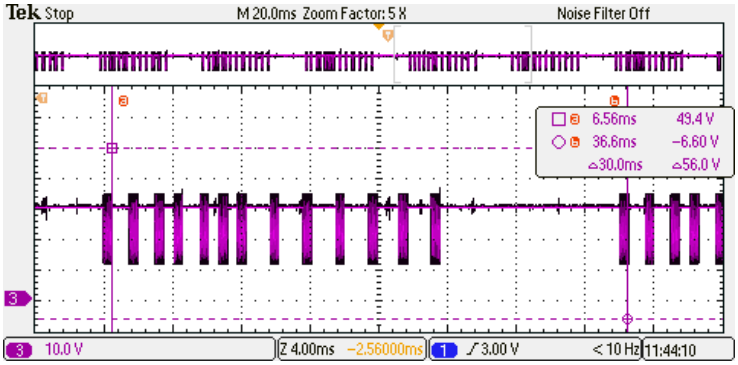


Abb. 3.3: 12 Channel Frame

... oder einen 12 Kanï¿½len Frame, bestehend aus 13 Peaks und 12 Pausensignalen.

3 Von Herrn Trybek bereitgestellt

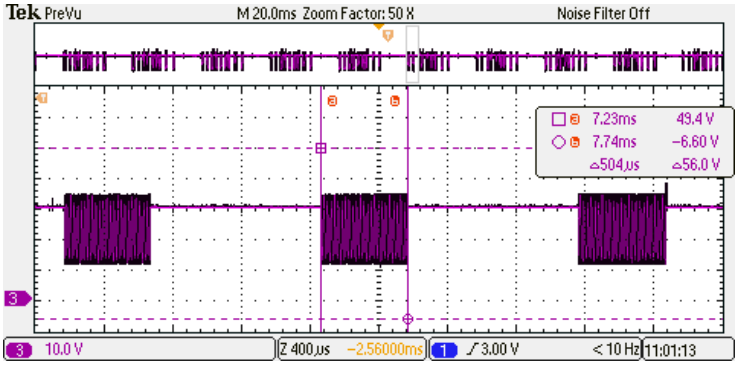


Abb. 3.4: Dauer 8 Chanel Frame

Folgend die Anschaung eines Frames mit acht Kanï¿½len mit der Dauer von 504us.

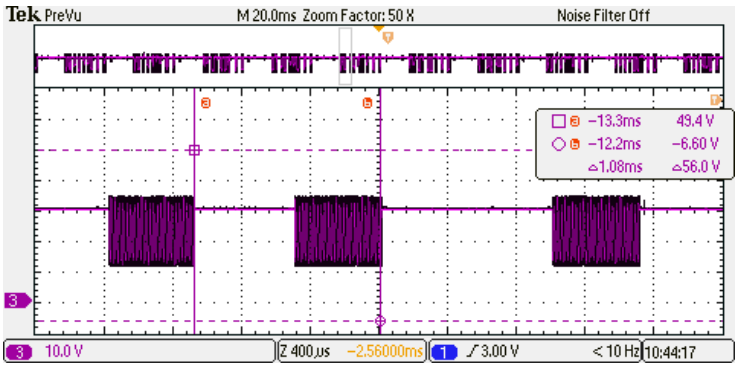


Abb. 3.5: Minimale Pause zwischen Frames

Zu sehen ist hier der kleinstmï¿½gliche zeitliche Abstand zwischen zwei Frames mit der Dauer von nur 504us.

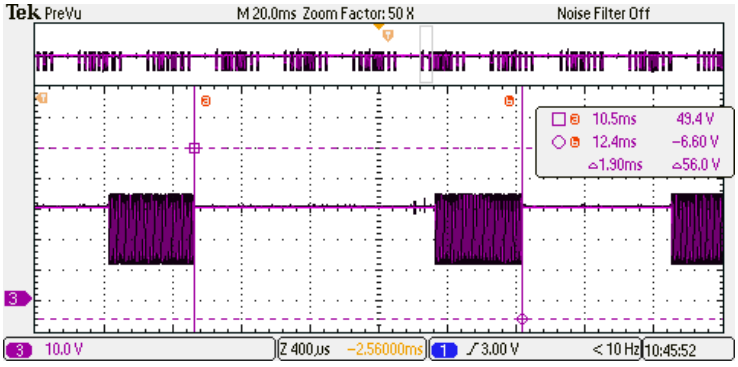


Abb. 3.6: Maximale Pause zwischen Frames

Hier ein Ausschnitt mit dem grï¿½ï¿½tmï¿½glichen Abstand zwischen zwei Frames mit der Dauer 1,396ms.

3.8 Hardware Layout

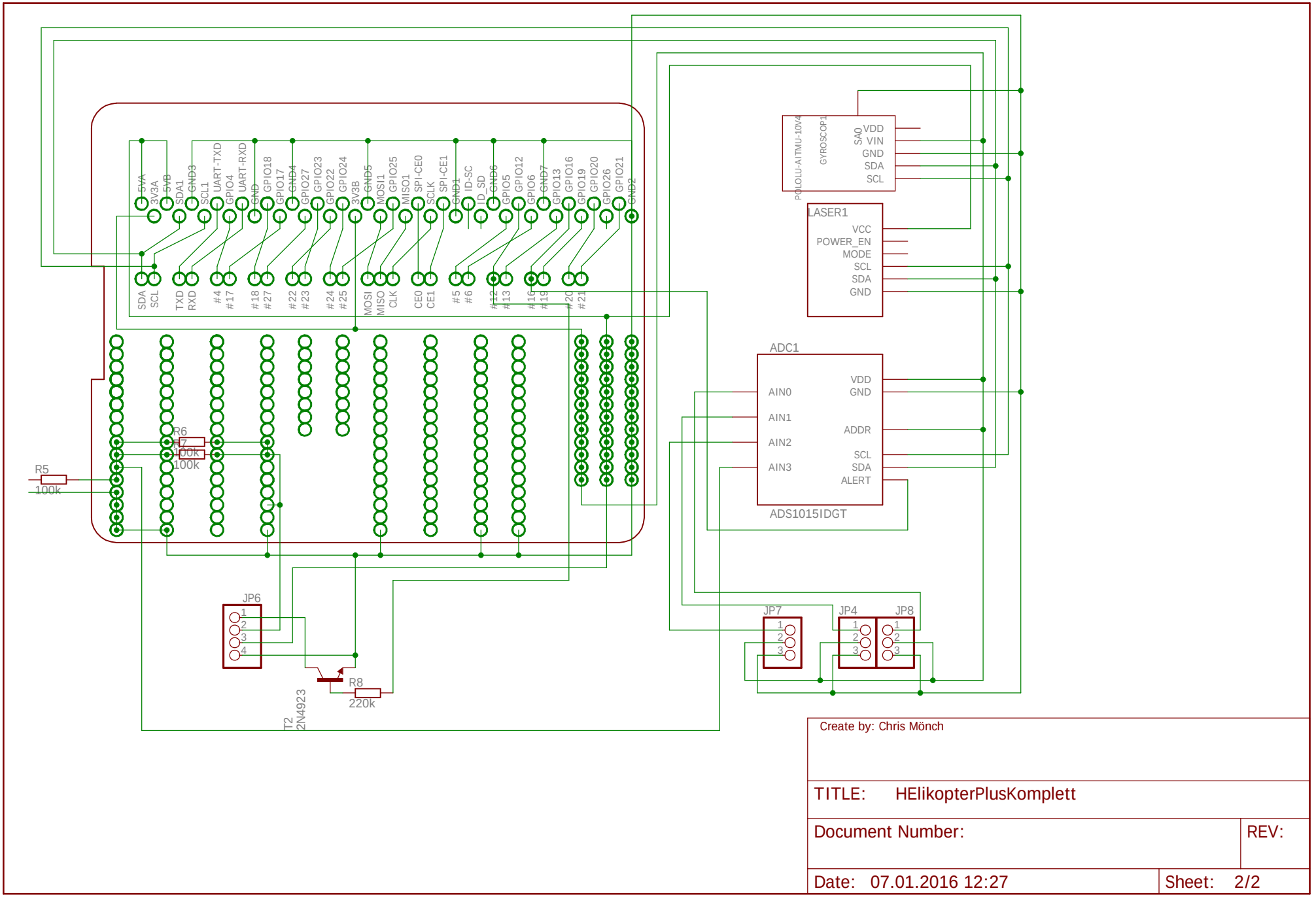
Fï¿½r die schnelle Durchfi¿½hrung von i¿½nderungen, Erweiterungen oder Anpassungen an der Hardware werden die Leiterplatinen und Schaltplï¿½ne mittels Tool elektronisch festgehalten. Die dazu mï¿½gliche Software sind: EAGLE oder Fritzing, die kostenlos oder als eingeschrï¿½nkte Freeversion angeboten werden, aber diversen Einschrï¿½nkungen unterliegen. Beide Tools haben eine Community, die viele Bauteile in Liberias bereitstellt.

Im direktem Vergleich macht EAGLE einen professionelleren Eindruck und bietet eine Sammlung von Tutorials und Libraries. Aus diesen Grï¿½nden fiel die Entscheidung auf die EAGLE SOFTWARE. Es sollte aber mit dem Platz mï¿½glichst sparend umgegangen werden, da bei dieser Freeeware die Grï¿½ï¿½e und Anzahl der Layouts begrenzt ist.

3.8.1 HElicopter Schematic Layout

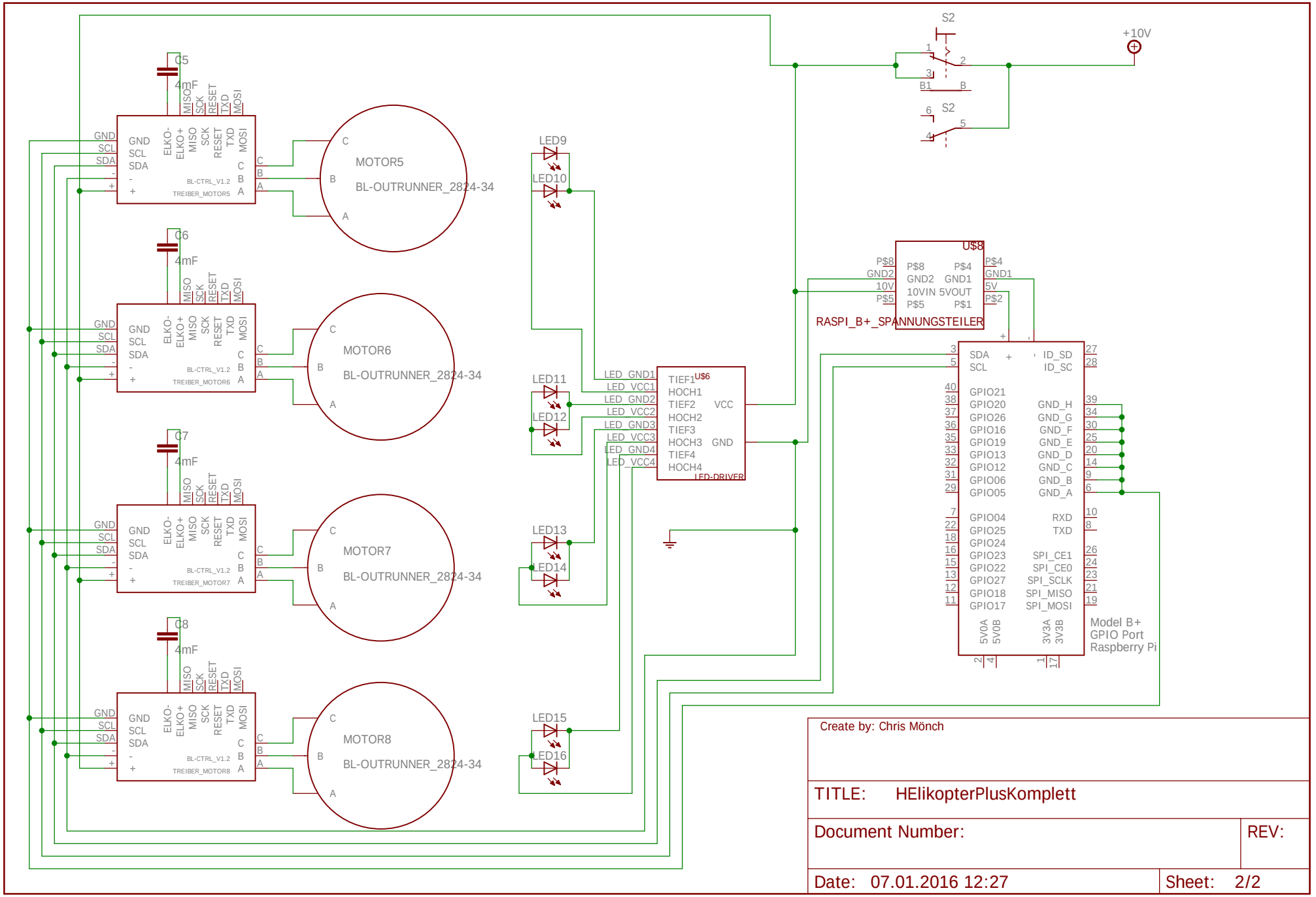
Es wurde fï¿½r die schematischen Zeichnungen ein EAGLE Projekt angelegt, dass die schematischen Zeichnungen aller selbst gebauten bzw. zusammengefï¿½gten Bauteile enthï¿½lt. Das Projekt File ist im Verzeichnis *trunk/hardware/* unter *HElicopterPlus* abgespeichert.

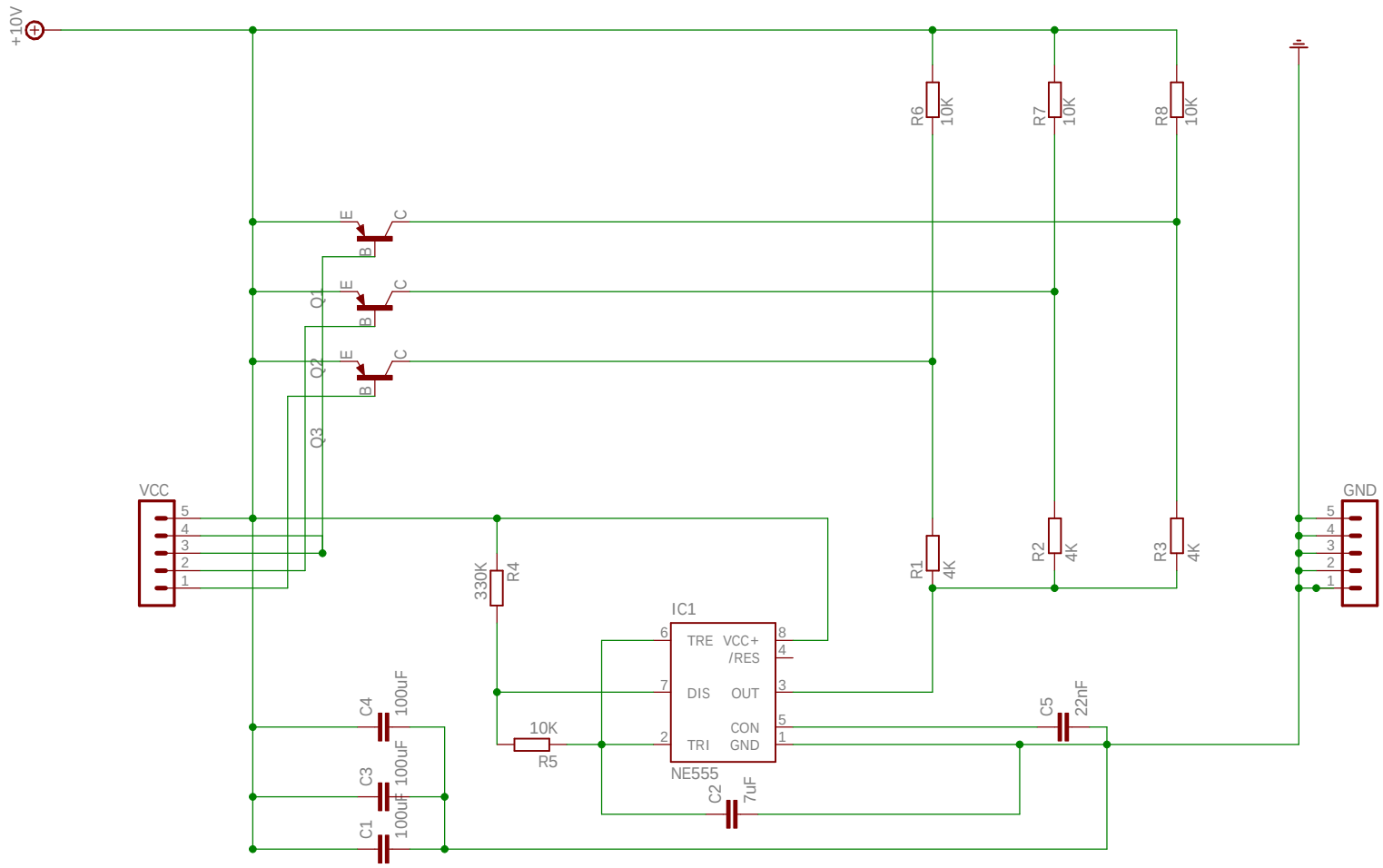
Derzeit befinden sich zwei verschiedene schematischen Zeichnungen im Projektverzeichnis, die maximal aus zwei sogenannten Sheets besteht (Begrenzung durch die Freeware-Version von EAGLE). Der erster Sheet beinhaltet immer, denn Komplettaufbau des jeweiligen Systems. Der zweite Sheet wird verwendet um druckbare Versionen das Schaltplans zu erstellen (DINA4 Frame). Hierbei ist es wichtig, eine ï¿½bersichtliche Abspaltung vorzunehmen.



Create by: Chris Mönch

TITLE: HElikopterPlusKomplett	
Document Number:	REV:
Date: 07.01.2016 12:27	Sheet: 2/2


S2
+10V
C5
4mF
C6
4mF
C7
4mF
C8
4mF
MISO SCK RESET TXD MOSI
GND SCL SDA - +
ELKO- ELKO+ MISO SCK RESET TXD MOSI
C B A
BL-CTRL_V1.2
TREIBER_MOTOR5
TREIBER_MOTOR6
TREIBER_MOTOR7
TREIBER_MOTOR8
MOTOR5
MOTOR6
MOTOR7
MOTOR8
BL-OUTRUNNER_2824-34
LED9
LED10
LED11
LED12
LED13
LED14
LED15
LED16
LED_GND1
LED_VCC1
LED_GND2
LED_VCC2
LED_GND3
LED_VCC3
LED_GND4
LED_VCC4
TIEF1
HOCH1
TIEF2
HOCH2
TIEF3
HOCH3
TIEF4
HOCH4
VCC
GND
U$6
LED-DRIVER
U$8
P$8 GND2 10V P$5
P$4 GND1 5V P$2
10VIN 5VOUT
RASPI_B+_SPANNUNGSTEILER
SDA SCL
ID_SD ID_SC
GPIO21 GPIO20 GPIO26 GPIO16 GPIO19 GPIO13 GPIO12 GPIO06 GPIO05
GND_H GND_G GND_F GND_E GND_D GND_C GND_B GND_A
GPIO04 GPIO25 GPIO24 GPIO23 GPIO22 GPIO27 GPIO18 GPIO17
RXD TXD
SPI_CE1 SPI_CE0 SPI_SCLK SPI_MISO SPI_MOSI
5V0A 5V0B 3V3A 3V3B
Model B+
GPIO Port
Raspberry Pi
Create by: Chris Mönch
TITLE: HElikopterPlusKomplett
Document Number:
REV:
Date: 07.01.2016 12:27
Sheet: 2/2



R1,R2,R3 = 4K = gelb schwarz schwarz braun braun
R4 = 330K = orange orange schwarz orange braun
R5 = 10K = braun schwarz schwarz rot braun
R6,R7,R8 = 10K (SMD)

Created by: Chris Mönch	
TITLE: LED-DRIVER	
Document Number:	REV:
Date: 07.01.2016 12:07	Sheet: 1/1

3.8.1.1 Richtlinien[7]

- Platzierung von mindestens einem Frame in jedem Sheet
- Verwendung von "CommonSSymbolen
- Jedem Bauteil einen Wert zuordnen (wenn mï¿½glich)
- Fï¿½r Verbindungen ausschlieï¿½lich *Net* verwenden (nicht *Wire*)
- Verbindungen oder ï¿½berbrï¿½ckungen wie in Abb. 3.7 dargestellt
- Abkï¿½rzungen direkt in dem jeweiligen Sheet festhalten
- Name und Bauteile immer in eine Richtung flieï¿½en lassen (wenn mï¿½glich)
- ï¿½berbrï¿½cken von Netzen mï¿½glichst gering halten
- Fï¿½r groï¿½e schematische Skizzen immer auch druckbare Versionen in DINA4 Format erstellen
- Wichtigen Netzen immer einen Namen zuordnen (GND, +5V, SDA, SLC, ...)
- Namen und Bezeichnungen kurz halten

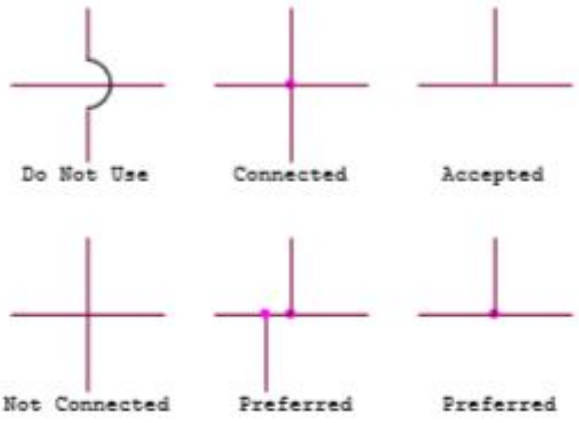


Abb. 3.7: Net verbinden/ï¿½berbrï¿½cken[4]

4 http://www.k-state.edu/ksuedl/publications/Technote%208%20-%20Guidelines%20for%20Drawing%20Schematics.pdf

3.8.2 HElicopter Libary

Auf der Bibliothek von Eagles kï¿½nnen unter der Rubrik *Downloads* Bauteile zu denn Schaltplï¿½nen gefunden werden. Die dort vorhandenen Bibliotheken sind von Usern bereitgestellt worden. Deshalb besteht die Mï¿½glichkeit, dass Bauteile fehlerhaft sind.

Wenn in der Bibliothek kein Element fï¿½r das Bauteil gefunden werden konnte, muss dieses erstellt und in der bereits vorhandene Libary *HElicopterLib.lbr* hinzugefï¿½gt werden. In dieser Libary sind alle Teile festgehalten, die fï¿½r die Quadrocopter benï¿½tigt werden.

Ein Bauteil oder auch ein Device besteht aus insgesamt drei eigenen Bestandteilen, die alle vorhanden sein mï¿½ssen, um das Bauteil zu verwenden:

Symbol	Darunter versteht man eine schematische Skizze eines Bauteils. Alle Schnittstellen des Bauteiles sind zu erkennen und ggf. zu kennzeichen. Der Orignalmaï¿½stab und Echtheit der Pinanordnung spielt keine Rolle. Diese soll mï¿½glichst so platziert werden, um im spï¿½teren Layout eine gewisse ï¿½bersichtlichkeit zu wahren. Die Namen der Pins sollen so benannt werden, wie diese vom Hersteller in den Datenblï¿½ttern festgelegt ist. Ansonsten kï¿½nnte es Spï¿½ter Probleme zur Pin-Zuordnung geben Die Schlï¿½sselwï¿½rter $>NAME$ und $>VALUE$ sollen sich (wenn mï¿½glich) im Symbol befinden.
Package	In dieser Skizze soll eine realitï¿½tsnahe Abbildung des Bauteiles erstellt werden, mit original Abmessungen wie z.B. Platinengrï¿½ï¿½e, Platinenform, Pins und Bohrlï¿½cher, die aus den jeweiligen Datenblï¿½ttern ersichtlich sind. Weitere Ergï¿½nzungen sind sonstige Bauteile oder SMD auf einem Device sowie deren Verbindungen auf Ober- bzw. Unterseite. Diese werden derzeit nicht benï¿½tigt und wurden deshalb weggelassen.
Device	In einem Device werden ein Symbol und ein Package miteinander zusammengefï¿½gt. Die Pins aus dem Symbol und Package Skizzen werden miteinander verknï¿½pft. Wenn jeder Pin zugeordnet wurde kann das Device verwendet werden.

Der Vorteil dieser Aufteilung ist der, dass einzelne Packages und Symbole fï¿½r mehrere Devices verwendet werden kï¿½nnen. Nur die Anzahl der Pins aus den Packages und Symbolen mï¿½ssen eindeutig ï¿½bereinstimmen und zugewiesen werden.

Um bereits existierende Libaries hinzuzufi¿½gen wird die gewï¿½nschte Libary in dem Ordner EAGLE-7.5.0/*lbrName* gespeichert. Anschlieï¿½end klickt man im Reiter *Bibliothek* auf *Benutzen* und wï¿½hlt die gewï¿½nschte Libary aus. Ein Schaltplan muss hierbei geï¿½ffnet sein. Im *Controll Panel* ist das Hinzufi¿½gen von libaries nicht mï¿½glich. Zuletzt mï¿½ssen die libaries noch aktualisiert werden(unter *Bibliothek* auf die Rubrik *Alle Aktualisieren*). Abschlieï¿½end auf *Add/Neues Bauteil hinzufi¿½gen*, es erscheint die Libary in der Liste.

Um ein vorhandenes Bauteil in die eigene Libary einzufi¿½gen muss die Ziel-Libary geï¿½ffnet sein. Im *Control Panel* von EAEGLE(in der linken Spalte das Menï¿½ *Bibliotheken* ï¿½ffnen). Hier sollten sich alle bereits hinzugefi¿½gten Libaries befinden. Ist dies nicht der Fall: rechtsklick auf die *Bibliotheken* und *alle Bibliotheken laden* auswï¿½hlen. Nun muss die Quell-Libary des Bauteils geï¿½ffnet werden.Das zu kopierende Bauteil rechts klicken und *In Bibliothek* auswï¿½hlen. Es ist auch mï¿½glich, einzelne Packages oder Symbole neben ganzen Bauteilen/Devices zu kopieren. Nun sollte das hinzugefi¿½gte Objekt in der Ziel-Libary geï¿½ffnet sein. Bestï¿½tigen sie das Hinzufi¿½gen mit Speichern der Libary.

Fï¿½r das Bearbeiten existierender Libaries ï¿½ffnen sie eine schematische Skizze. Klicken sie auf *Bibliothek*, dann *ï¿½ffnen..* und wï¿½hlen sie die zu ï¿½berarbeitende Libary aus. Nun kï¿½nnen alle existierende Bauteile, Packages und Symbole bearbeitet oder neue erstellt werden. Nach der ï¿½nderung speichern Sie die Libary und klicken sie auf wieder auf Bibliothek. Anschlieï¿½end aktualisieren Sie die Libaries. Die ï¿½nderungen sollten nun vorgenommen sein.

Achtung: ï¿½nderungen werden auf alle bestehenden und eingefi¿½gten Elementen vorgenommen. Dies kann sich z.B. unvorteilhaft auf die Lesbarkeit auswirken. Aus diesem Grund sollte zumindest eine Kopie des geï¿½nderten Devices oder eine neue Version erstellt werden.

4 Ausblick

Für die Flugfähigkeit des Quadrocopters sollten folgende Themen bearbeitet werden:

Behebung des Fehlers der VMware wurde im folgenden Abschnitt 2.5.1. Udp Socket Programm mit Windows 10 und VMware auf S.21 beschrieben. Um für weitere Projektarbeiten wieder das Arbeiten mit der VMware zu ermöglichen sollte dieses Problem gelöst werden.

Die verwendeten Scripts local und nicht mehr über ssh Zugriff starten lassen und z.B. per UDP die Daten an das Raspberry übertragen.

Entwicklung von Stabilitätsregelung für vertikales und horizontales Halten.

Anbindung der Fernsteuerung mittels GR-16 (siehe Abschnitt 3.7. Remotecontroller-Treiber auf S.42) mit einem externen PPM-Decoder oder über den Echtzeitfähigen installierten Linux Kernel.

A Code

```
/*
 * MOTOR.h
 *
 *  Created on: Nov 18, 2015
 *      Author: ezs
 */

#include "../../MAIN/main.h"

//=============================================================================================================rft=
// DEF DEFAULTS
//-------------------------------------------------------------------------------------------------------------
// ***** defaults: Motors - dependent on copter type
#ifdef Quadro_Plus
//-------------------------------------------------------------------------------------------------------------
// Quadro Plus - hardware configuration
//               motor rotating direction: (>) CW - clockwise / (<) CCW - counterclockwise
//
//                             Nick (+ x-axis)
//
//                                    o MotorNo1(>)
//                                    |
//  Roll (+ y-axis)    MotorNo4(<) o-F|C-o MotorNo3(<)  Roll (- y-axis)
//                                    |
//                         MotorNo2(>) o
//
//                             Nick (- x-axis)
//-------------------------------------------------------------------------------------------------------------
  #define DEFMotorsCount                4          // (CC) Quadrocopter
  #define DEFCopterType              0x50          // (CC) P (ASCII) Plus

  // order to put into execution [IDX]
  #define DEFMotorNo1_OrderIDX          0          // MotorNo1(>)     0 <=> 1st
  #define DEFMotorNo2_OrderIDX          2          // MotorNo2(>)     2 <=> 3rd
  #define DEFMotorNo3_OrderIDX          1          // MotorNo3(<)     1 <=> 2nd
  #define DEFMotorNo4_OrderIDX          3          // MotorNo4(<)     3 <=> 4th
#else

#ifdef Quadro_X
//-------------------------------------------------------------------------------------------------------------
// Quadro X    - hardware configuration
//               motor rotating direction: (>) CW - clockwise / (<) CCW - counterclockwise
//
//                             Nick (+ x-axis)
//
//                     MotorNo4(<) o     o MotorNo1(>)
//                                  \ /
//  Roll (+ y-axis)                 F|C                Roll (- y-axis)
//                                  / \
//                     MotorNo2(>) o     o MotorNo3(<)
//
//                             Nick (- x-axis)
//-------------------------------------------------------------------------------------------------------------
  #define DEFMotorsCount                4          // (CC) Quadrocopter
  #define DEFCopterType              0x58          // (CC) X (ASCII) X

  // order to put into execution [IDX]
  #define DEFMotorNo1_OrderIDX          0          // MotorNo1(>)     0 <=> 1st
  #define DEFMotorNo2_OrderIDX          2          // MotorNo2(>)     2 <=> 3rd
  #define DEFMotorNo3_OrderIDX          1          // MotorNo3(<)     1 <=> 2nd
  #define DEFMotorNo4_OrderIDX          3          // MotorNo4(<)     3 <=> 4th
#else

#ifdef Okto_Plus
//-------------------------------------------------------------------------------------------------------------
// Okto Plus(A)- hardware configuration
```

```
//              motor rotating direction: (>) CW - clockwise / (<) CCW - counterclockwise
//
//                        Nick (+ x-axis)
//
//                                    o MotorNo1(>)
//                      MotorNo8(<) o  |  o MotorNo2(<)
//                                   \|/
//  Roll (+ y-axis)  MotorNo7(>)  o--F|C--o MotorNo3(>)  Roll (- y-axis)
//                                   /|\
//                      MotorNo6(<) o  |  o MotorNo4(<)
//                         MotorNo5(>) o
//
//                        Nick (- x-axis)
//-------------------------------------------------------------------------------------------------------
  #define DEFMotorsCount                8          // (CC) Oktocopter
  #define DEFCopterType              0x50          // (CC) P (ASCII) Plus

  // order to put into execution [IDX]
  #define DEFMotorNo1_OrderIDX          0          // MotorNo1(>)     0 <=> 1st
  #define DEFMotorNo2_OrderIDX          1          // MotorNo2(<)     1 <=> 2nd
  #define DEFMotorNo3_OrderIDX          2          // MotorNo3(>)     2 <=> 3rd
  #define DEFMotorNo4_OrderIDX          3          // MotorNo4(<)     3 <=> 4th
  #define DEFMotorNo5_OrderIDX          4          // MotorNo5(>)     4 <=> 5th
  #define DEFMotorNo6_OrderIDX          5          // MotorNo6(<)     5 <=> 6th
  #define DEFMotorNo7_OrderIDX          6          // MotorNo7(>)     6 <=> 7th
  #define DEFMotorNo8_OrderIDX          7          // MotorNo8(<)     7 <=> 8th

#else
#error "*** copter type ***  not defined!!!! (MOTOR.h) ***"
#endif
#endif
#endif

//-------------------------------------------------------------------------------------------------------
// ***** defaults: Motors - common

#define DEFMotorSetpointMIN            10          // [MotorStep] Min Motors RPM Setpoint
#define DEFMotorSetpointMAX           255          // [MotorStep] Max Motors RPM Setpoint

// Brushless Controller adress (i2c-bus) for MotorNo1,..,MotorNo8
// bit7-1[0x52+2*(MotorNo99-1)] bit0[0:write; 1:read]
#define DEFMotorNo1_BLCtrlADR        0x29//0x52          // (CC) BL Ctrl adresse for MotorNo1
#define DEFMotorNo2_BLCtrlADR        0x2a//0x54          // (CC) BL Ctrl adresse for MotorNo2
#define DEFMotorNo3_BLCtrlADR        0x2b//0x56          // (CC) BL Ctrl adresse for MotorNo3
#define DEFMotorNo4_BLCtrlADR        0x2c//0x58          // (CC) BL Ctrl adresse for MotorNo4
#define DEFMotorNo5_BLCtrlADR        0x5a          // (CC) BL Ctrl adresse for MotorNo5
#define DEFMotorNo6_BLCtrlADR        0x5c          // (CC) BL Ctrl adresse for MotorNo6
#define DEFMotorNo7_BLCtrlADR        0x5e          // (CC) BL Ctrl adresse for MotorNo7
#define DEFMotorNo8_BLCtrlADR        0x60          // (CC) BL Ctrl adresse for MotorNo8

//defines for SetPwmMotor toSet parameter:
#define DEFMotorNo1_PWM         0b1
#define DEFMotorNo2_PWM         0b10
#define DEFMotorNo3_PWM         0b100
#define DEFMotorNo4_PWM         0b1000
#define DEFMotorNo5_PWM         0b10000
#define DEFMotorNo6_PWM         0b100000
#define DEFMotorNo7_PWM         0b1000000
#define DEFMotorNo8_PWM         0b10000000
#define DEFMotorALL_PWM         0xFF
#define DEFMotorCW_PWM          0x55
#define DEFMotorCCW_PWM         0XAA

void InitMotor(int microSeconds);
void SetMotorExecutionOrder();
```

```
void SetPwmMotor(char toSet , int pwmValue, int forceSend);
void AddPwmMotor(char toSet , int pwmValue, int forceSend);
void SubbPwmMotor(char toSet , int pwmValue, int forceSend);

int GetPwmMotor(int motorNumber);
void InitMotorTimer(int microSeconds);
void SetFlagRunSendPwmToMotor(char value);
char GetFlagRunSendPwmToMotor();
void IsrSetFlag();
void sendPwmToMotor();
void GetBLCtrlADRExecuteOrder(char BLCtrlADRExecuteOrder[]);
```

A.1 Script *'testcase'*

```
#!/bin/bash
#Sets or Clear Flag to Run a testcase
#Used Parameter:
# $1 - Testcase name which to set or clear
# $2 - Optional Parameter set or clear the Testcase,if Unknown
   testcase will be set

#Path of file where all testacses are stored:
path="/home/pi/testfiles/_Testcases"

if [ "$1" == '' ]
then
echo "First␣Paremeter␣does␣not␣exist,␣expected␣name␣of␣a␣testcase"
exit 1
else
testcase=$1
fi

if [ "$2" == "clear" -o "$2" == "set" ]
then
if [ "$2" == "clear" ]
then
mode=0
else
mode=1
fi
echo "mode␣$2␣is␣used"
else
mode=1
echo "used␣default␣mode␣set"
fi

if [ -e $path ]
then
echo "$path␣found"
else
echo "$path␣not␣found"
exit 2
fi

sed -i "s/^$testcase=[01]/$testcase=$mode/" $path
if [ $? == 0 ]
then echo "Command␣executed"
fi
```

Literaturverzeichnis

[1] Graupner/SJ Gmbh, Bedienungsanleitung Graupner HoTT 2.4, http://www.graupner.de/mediaroot/files/33508_Kurzanleitung_de.pdf; *Januar 2011 DE V1.3*

[2] Herbert Bernstein, Informations- und Kommunikationselektronik, Walter de Gruyter GmbH, 2015, *1. Auflage*

[3] Lady ada/Adafruit Industries, Adafruit Ultmate GPS HAT for Raspberry Pi, https://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-hat-for-raspberry-pi.pdf, *2016-01-11*

[4] Bill Earl/Adafruit Industries, Adafruit 4-Channel ADC Breakouts, https://learn.adafruit.com/downloads/pdf/adafruit-4-channel-adc-breakouts.pdf, *2014-11-30*

[5] Pololu Robotics & Electronics, AAltIMU-10 v4 Gyro, Accelerometer, Compass, and Altimeter (L3GD20H, LSM303D, and LPS25H Carrier), https://www.pololu.com/product/2470, *2016-01-13*

[6] EXP Tech, LIDAR-Lite v2, http://www.exp-tech.de/lidar-lite-v2, *2016-01-13*

[7] Olin Lathrop, Rules and guidelines for drawing good schematics, http://electronics.stackexchange.com/posts/28255/revisions, *2016-01-14*

[Gun04] Karsten Günther, LaTeX2 — Das umfassende Handbuch, Galileo Computing, 2004, http://www.galileocomputing.de/katalog/buecher/titel/gp/titelID-768; *1. Auflage*