Dokumentation
RoboMirror

Team 1: Eggs & Bacon

Tobias Gilgenreiner, Felix Dollinger, Marco Stoiber,
Maximilian Kritzenthaler, Eduard Schröder

Abgabefrist: 20.08.2020
Betreuer/Prüfer: Prof. Roth, Altmann
Fakultät: Informatik und Mathematik
Studiengang: Technische Informatik

Eggs & Bacon

>> Software_Solutions;

Inhalt

1 Sensorik

Im Folgenden wird der Aufbau der Sensorik, die Einrichtung der Entwicklungsumgebung und die Entwicklung der Firmware für die Arduino Mini Pros dokumentiert.

1.1 Systemhardware

Zur Bestimmung der menschlichen Bewegungen werden MPU6050 Einheiten verwendet. Hierbei handelt es sich um 3-Achsen Lage bzw. Beschleunigungssensoren. Dabei misst der Lagesensor die Winkelgeschwindigkeit bei Rotation um die x-/y-/z-Achse unter Verwendung des Corioliseffekts. Der Beschleunigungssensor macht sich die Gravitationskraft zu nutzen und bestimmt so die auf die Achsen wirkende Kraft bei Neigung des Sensors. Beide Sensoren besitzen unterschiedliche Sensitivitätsstufen. Für das Projekt wurden die Standard Einstellungen verwendet, also +/- 250°/s (Lagesensor) bzw. +/- 2g (Beschleunigungssensor). Dabei ist zu beachten, dass die Sensorwerte zunächst Rohdaten ausgeben, die zur tatsächlichen Verwendung erst noch bearbeitet werden müssen. Das geschieht im Backend.

Zum Auslesen der Sensorwerte und Weitergabe der Daten an das Backend werden Arduino Mini Pros verwendet. Diese kommunizieren über einen I2C Bus mit jeweils zwei zugeordneten Sensoren und fragen abwechselnd die Sensordaten ab und senden die Werte über eine UART Verbindung an das Backend.

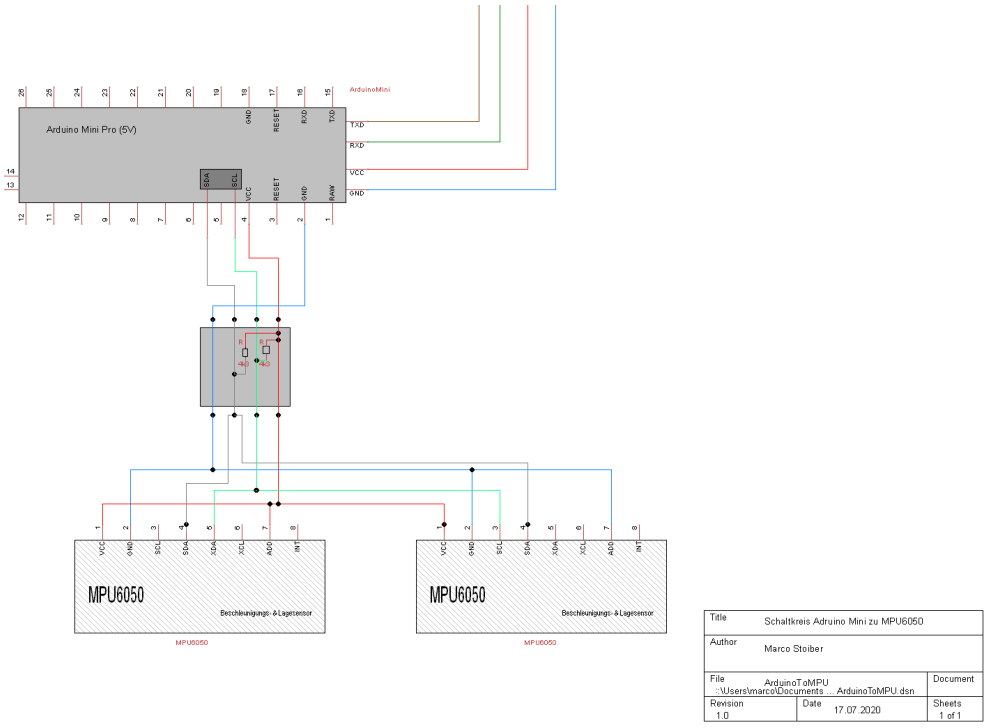


Abbildung 1: Schaltkreis MPUs und Arduino Mini

Für den I2C Bus sind externe Pullup Resistoren notwendig, die sowohl die Dataline, als auch die Clockline bei Nichtnutzung des Busses auf High ziehen.

Zur Berechnung des korrekten Widerstands können folgende Formeln verwendet werden:

$$\text{Freq} < 100\text{kHz} \implies R_{\min} = \frac{V_{cc} - 0.4\text{V}}{3\text{mA}}, R_{\max} = \frac{1000\text{ns}}{C_{\text{bus}}}$$

$$\text{Freq} > 100\text{kHz} \implies R_{\min} = \frac{V_{cc} - 0.4\text{V}}{3\text{mA}}, R_{\max} = \frac{300\text{ns}}{C_{\text{bus}}}$$

Abbildung 2:Formel Widerstandberechnung I2C

Wir haben für den I2C die Standard Übertragungsrate von 100kHz verwendet und liegen damit bei einem Wert zwischen 1,5 und 10kΩ. Letztlich sind 4,3kΩ Pullup Widerstände verbaut worden.

Über den ADD Pin der MPUs kann die Hardwareadresse manipuliert werden. Setzt man den Pin auf High, wird das letzte Bit der Adresse zu 1 bzw. beim Setzen auf Low zu 0. Das ist notwendig, um die beiden Sensoren bei der I2C Kommunikation voneinander unterscheiden zu können.

Unserer Erfahrung nach, halten die LEDs der MPUs nicht lange. Daher der Hinweis: Nur weil kein Lämpchen auf dem Sensor leuchtet, heißt es nicht, dass er nicht mehr funktioniert.

1.2 Einrichtung der Entwicklungsumgebung

Die Firmware ist mit Atmel Studio 7 auf einem Win10 Host entwickelt worden. Zum Flashen der Arduino Minis wurde ein USBasp verwendet. Zur Einrichtung wurde folgende Anleitung verwendet[1]. Bei der Anleitung ist zu berücksichtigen, dass es sich bei den Arduino Mini Pros um ein ATmega328P Board handelt.

Weiterhin ein Hinweis zur Installation des USBasp Treibers. Es muss aus Kompatibilitätsgründen der libusb-win32(v1.2.6.0) Treiber installiert werden.

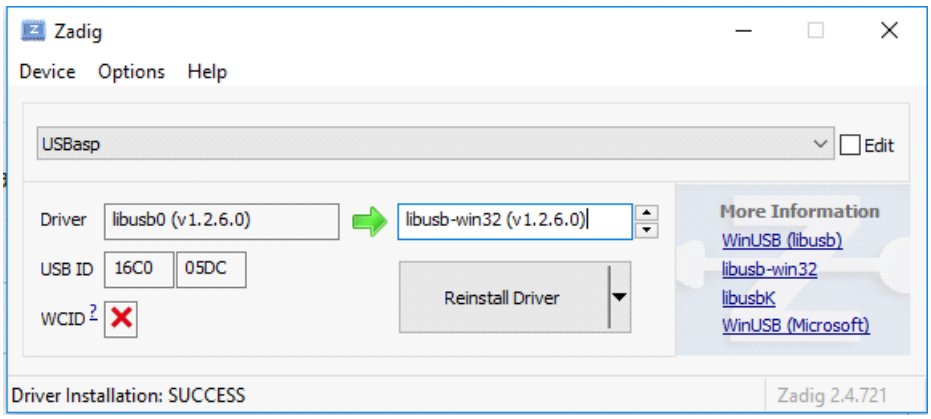


Abbildung 3:USBasp Treiber Installation

Zum Flashen sind male to female Stecker notwendig, die wie folgt verbunden werden müssen:

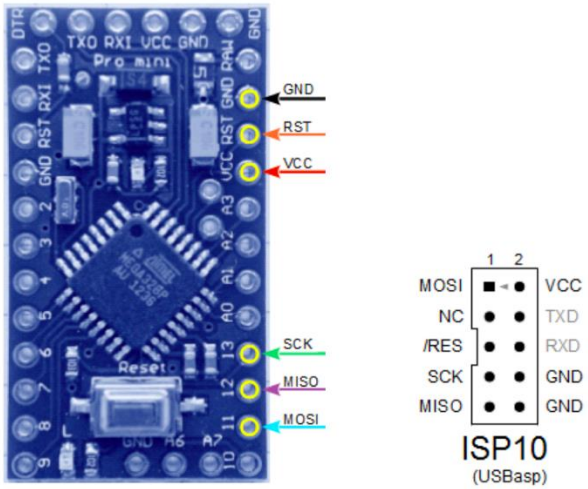


Abbildung 4: Arduino Mini Pro - USBasp Verbindung

[1] https://medium.com/manash-en-blog/setting-up-atmel-studio-for-usbasp-and-avr-programming-802bb4dcf1e9

1.3 Entwicklung der Firmware

Die Firmware beinhaltet zwei Standard Bibliotheken für die I2C bzw. die UART Kommunikation. Weiterhin wurde eine Bibliothek für die MPUs entwickelt. Diese ermöglicht die Initialisierung der Sensoren und das Abfragen von Lage- und Beschleunigungswerten. Die Initialisierung besteht im Grunde darin, die Sensoren aus dem Ruhezustand aufzuwecken. Zusätzlich zu den Sensordaten wird der jeweilige Zeitpunkt der Abfrage übergeben. Dafür wird der Timer1, des ATmega328P Boards verwendet, der die verstrichene Zeit in Millisekunden genau angibt. Um die Datenpakete voneinander zu unterscheiden wird eine ID mitversendet. Diese muss manuell in der Firmware angepasst werden, da sich die Sensoren zwar innerhalb eines Messpaars unterscheiden, aber auf das ganze System betrachtet mehrfach die gleiche Hardwareadresse auftritt.

Ein exemplarisches Datenpaket sieht so aus:

id 0 t 150 ax -8932 ay -204 az 12912 gx 3035 gy- 4083 gz -179

Die Abkürzungen a bzw. g stehen für accelrometer (= Beschleunigungssensor) und gyrometer (= Lagesensor).

Essentiell für die Entwicklung des Codes war das Debuggen mit Putty. Dafür ist aber eine funktionierende UART Kommunikation notwendig. Dafür wurde eine vorgefertigte Bibliothek verwendet.

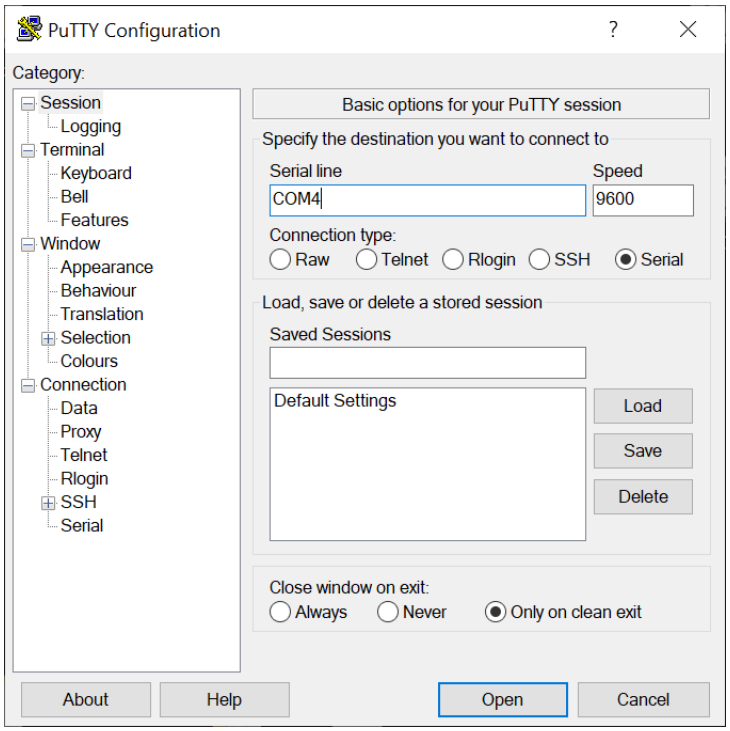


Abbildung 5: Putty

In Putty muss die Serielle Schnittstelle angepasst werden. Dafür muss im Geräte-Manager überprüft werden welchen Port der UART zu USB Wandler vom Betriebssystem erhalten hat. Zusätzlich muss die verwendete Baudrate angegeben werden. In diesem Fall liegt diese bei 9600. Zur verbesserten Übersichtlichkeit sollten Verzögerungen in Form von waits im Code eingebaut werden.

Ein weiterer Hilfreicher Punkt ist es, vorab die Sensordaten mit Hilfe von Plots zu visualisieren, um ein Gefühl für die Werte zu bekommen.

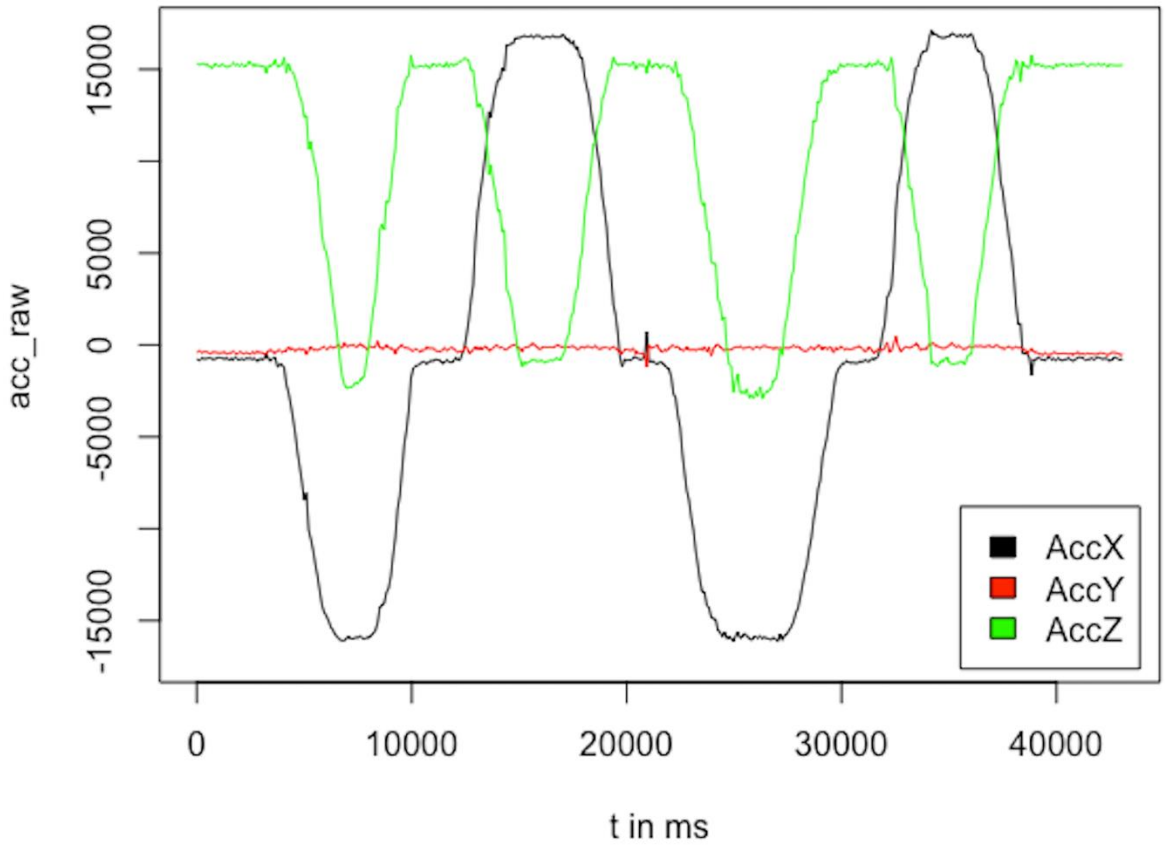


Abbildung 6: Accelerometer Links-Rechts-Rotation

Hier sieht man das Verhalten des Beschleunigungssensors bei abwechselnder rechts bzw. links Rotation. Man erkennt das grundsätzliche Verhalten der Achsen und stellt weiterhin fest, dass die Werte mit einem Rauschen behaftet sind.

2 Backend

Im Folgenden wird das Einrichten der Systeme als Access Point/Client, der Aufbau der Verbindungen vom/zum Backend und die laufenden Konsolen-Anwendungen genauer beschrieben.

2.1 Systemhardware

Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+, Banana Pi M2+

Die Systeme sollen auf Linux basieren. Raspbian als Betriebssystem ist am geeignetsten, da sämtliche inkludierten Bibliotheken direkt unterstützt werden sollten (besonders wiringPi.h/wiringSerial.h für UARTs am GPIO).

Es können mehrere Systeme verwendet werden, da dies erlaubt ggf. weitere UARTs Ports auszunutzen. So unterstützt theoretisch der Raspberry Pi 4 bis zu 6 UARTs (durch doppelte Belegung), aber es kann auch ein Setup erfolgen wie: Raspberry Pi 3b+ (1 GPIO UART + 1 USB UART) & ein Banana Pi M2+ (2 GPIO UARTs). Dies geschieht durch die Aufteilung auf die zwei Programme, welche in *2.4* für das Access Point System und in *2.5* für weitere verbundene Systeme erklärt werden.

2.2 Einrichtung des Access Point

Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+

Die Einrichtung findet nach dem offiziell von Raspberry vorgestellten Ablauf statt (Raspberry Pi, 2020)[2]. Zu beachten ist, dass abhängig von dem Modell unterschiedliche Konnektivität-Stabilitäten zu erwarten sind (Raspberry Pi 4 >> Raspberry Pi 3b+). Über LAN wird ein weiteres Netzwerk überbrückt, d.h. das über den Router somit noch Zugang zum Internet für das System und seine verbundenen Geräte hergestellt werden kann.

Die Einrichtung findet über die Konsole statt. Code nach dem Befehl „nano“ wird entsprechend in die Datei eingefügt und abgespeichert.

1. ***hostapd* installieren und beim Startvorgang aktivieren:**

```
sudo apt install hostapd
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
```

2. ***dnsmasq* installieren für Netzwerk Management Dienste (DNS, DHCP)**

```
sudo apt install dnsmasq
```

[2] https://github.com/raspberrypi/documentation/blob/master/configuration/wireless/access-point-routed.md

3. **Tools für Firewall Regeln (Speicherung/Wiederherstellung) installieren**

```
sudo DEBIAN_FRONTEND=noninteractive apt install -y
netfilter-persistent iptables-persistent
```

4. **Netzwerk konfigurieren: Feste IP mit 192.168.4.1 (darf nicht bei Nutzen eines Routers über LAN vergeben sein)**

```
sudo nano /etc/dhcpcd.conf
```

```
interface wlan0
static ip_address=192.168.4.1/24
nohook wpa_supplicant
```

5. **Routing und IP masquerading aktivieren (falls Internet erwünscht ist)**

```
sudo nano /etc/sysctl.d/routed-ap.conf
```

```
net.ipv4.ip_forward=1
```

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo netfilter-persistent save
```

6. **Konfigurieren der DHCP- und DNS-Dienste: Dynamische Zuweisung der Geräte. Zusätzlich sicherstellen, dass WiFi nicht blockiert wird.**

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig
sudo nano /etc/dnsmasq.conf
```

```
interface=wlan0 # Listening interface
dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
                # Pool of IP addresses served via DHCP
domain=wlan     # Local wireless DNS domain
address=/gw.wlan/192.168.4.1
                # Alias for this router
```

```
sudo rfkill unblock wlan
```

7. **Konfigurieren des Access Points: Nun die erwünschten Parameter eingeben. Netzwerk Name bei *ssid* angeben.**

   Netzwerk Passwort bei *wpa_passphrase* angeben. (Zwischen 8-64 Chars)

```
sudo nano /etc/hostapd/hostapd.conf
```

```
country_code=DE
interface=wlan0
ssid=RPI4
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=INSERTPASSWORDHERE
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Anm.: Der Channel wird nicht automatisch festgelegt, d.h. falls es mit einem umgebenen Router eine Kollision gibt wird der Raspberry mit aller Wahrscheinlichkeit verlieren und kein Netzwerk anbieten! Deshalb ggf. an die Situation anpassen.

8. **Neustarten: Access Point wird fortan mit dem Starten aktiviert.**

```
sudo systemctl reboot
```

2.3 Einrichten weiterer Systeme

Getestet auf: Banana Pi M2+

Damit sich weitere Systeme mit den Hauptsystem, also dem Access Point verbinden, muss nur eine WLAN-Verbindung zwischen diesen Systemen aufgebaut werden.

Dafür sucht man sich das Netz, das in der *hostapd.conf* von dem eingerichteten Access Point angegeben ist. Also die *ssid* als Name des Netzes und *wpa_passphrase* als Passwort.

In der *main.cpp* des zentralen Backend muss noch ein weiterer Thread angelegt werden, welcher diese Daten ausließt. Dies erfolgt ähnlich der Funktion *thread_get_rpi3bplus_data()* in der *main.cpp*. Der angegebene Port muss mit der in der *DistributeData::send_data()* des weiteren Systems übereinstimmen und nicht anderweitig belegt sein.

Anm.: Allgemein ist zu sagen, dass das WLAN-Netzwerk von derartigen Systemen nicht stark genug ist um datenintensive Aufgaben, wie Remote-Desktop oder Remote-Debug einwandfrei und stabil auszuführen. Deshalb hat sich in der Praxis gezeigt, dass solche Aufgaben am besten in einem weiteren Netzwerk über LAN stattfinden sollten. Beispielsweise wenn der Banana Pi und das Notebook jeweils über LAN an einen Router verbunden wird, sodass nun über dieses Netzwerk Remote-Desktop ausgeführt wird.

2.4 Programmausführung des zentralen Backend

Getestet auf: Raspberry Pi 4, Raspberry Pi 3b+

Das Linux-Projekt „rm_backend_rpi4“ soll am besten über Visual Studio am Remotecomputer ausgeführt werden, um mehr Debug Informationen/Möglichkeiten zu erhalten. Eine Vorgehensweise wird von Microsoft beschrieben (Microsoft, 2020)[3]

Alternativ kann die erstellte .out Datei (rm_backend_rpi4.out) in einer Konsole ausgeführt werden.

Es wird zunächst nach einer Benutzereingabe gefragt. Mögliche Eingaben sind „live“ für die Ausführung mit den Echtzeitdaten der momentan angeschlossenen Sensoren, oder eine *.txt Datei, welche eine Abfolge von voraufgezeichneten Sensordaten enthält. Diese können etwa während einer live Ausführung aufgenommen werden. Dazu muss nur währenddessen „R“ eingegeben werden, und in einer weiteren Eingabe einen Namen für die zu erstellende Textdatei. Mit „S“ wird die Aufnahme gestoppt und bei dem Verzeichnis „/home/pi/Documents/*.txt“ gespeichert.

Wird eine live Demo erwünscht versucht das Programm alle lokalen UART Verbindungen auszulesen und ggf. TCP/IP Verbindungen mit weiteren Systemen aufzubauen. Diese sind in der *main.cpp* aufgeführt. Sollte nur eine Verbindung fehlerhaft sein, wird das Programm unterbrechen.

Anschließend werden die TCP/IP Verbindungen mit den Unity Clients und dem RoboNova WLAN-Modul aufgebaut. Ebenso gilt hier: Sollte nur eine Verbindung fehlerhaft sein wird das Programm nicht fortsetzten. Daten werden nun an die Clients gesendet.

2.5 Programmausführung weiterer Systeme

Getestet auf: Banana Pi M2+

Das Linux-Projekt „rm_backend_rpi3bplus2“ soll am besten über Visual Studio am Remotecomputer ausgeführt werden, um mehr Debug Informationen/Möglichkeiten zu erhalten. Eine Vorgehensweise wird von Microsoft beschrieben (Microsoft, 2020)[2]

Alternativ kann die erstellte .out Datei (rm_backend_rpi3blus2.out) in einer Konsole ausgeführt werden.

[3] https://docs.microsoft.com/de-de/cpp/linux/connect-to-your-remote-linux-computer?view=vs-2019

Bei Ausführung versucht der Client sich mit dem Access Point zu verbinden und leitet diesem die unverarbeiteten Daten, der an den UARTs angeschlossenen Sensoren weiter.

2.6 Zeitlicher Ablauf der Ausführungen

Die Startreihenfolge der Dienste kann variieren. Hierbei handelt es sich um die Standardreihenfolge:

1. Programmstart des zentralen Backend mit „live“ oder *.txt Datei
2. Ausführung der Unity Simulation
3. Verbinden des RoboNova WLAN-Modul
4. Starten weiterer Systeme

2.7 Software des WLAN-Modul

Die Software zum Flashen des ESP8266 WLAN-Modul beinhaltet die festen *ssid* und *wpa_passphrase* des Access Point. Außerdem sind dort die IP-Adresse und Port des zentralen Backend fest vergeben. Das Modul versucht sich nach der Stromversorgung mit dem Access Point zu verbinden und empfängt daraufhin die Daten und schickt diese weiter über UART. Bei einem Timeout versucht sich das Modul alle 5 Minuten neu zu verbinden. Das Resetten des Moduls durch das Unterbrechen der Stromversorgung ist ebenso möglich.

2.8 Mögliche nachträgliche Änderungen

Die aktuelle Vorgehensweise kann dem Code entnommen werden. Hier werden noch mögliche Änderungen erwähnt:

1. **Filteränderung**

   Aktuell wird ein einfacher Komplementärfilter verwendet. Falls ein genauerer Filter erwünscht ist, muss die Funktion *filter_values()* in der Klasse *DistributeData* verändert bzw. ausgetauscht werden.
2. **Shutdown der TCP/IP Verbindungen**

   Die Umsetzung der TCP/IP Verbindungen in C++ führt teils zu Komplikationen bei den gebundenen Ports, sodass ein abruptes Abbrechen des Programmes noch zu existierenden Binds bei den Ports führt, welche erst von dem Betriebssystem nach einer gewissen Zeit beendet werden. Dafür wurden Benutzereingaben eingerichtet, welche eine Verbindung sicher beendet. Diese wurden aber nicht getestet und weiterhin entsteht das Problem bei sonstigen Beendigungen.

3 Kommunikation

3.1 Einrichten der Verbindung zwischen Backend und Arduino Mini

Getestet auf: Raspberry Pi 4

Zur Kommunikation zwischen dem Backend und den Arduino Minis wird das RS232 UART-Protokoll verwendet. Da es sich bei UART um eine serielle Schnittstelle zum Senden und Empfangen von Daten handelt, muss auch auf dem Raspberry Pi die serielle Schnittstelle in den Systemeinstellungen aktiviert werden.

Hier wird zur Einrichtung die Konsole verwendet.

1. **Aktivieren der seriellen Schnittstelle unter Raspbian.**

   Aufrufen der Systemeinstellungen:

```
sudo raspi.config
```

   Anschließend werden der Reihe nach folgende Optionen ausgewählt:

   1. Interfacing Options
   2. Serial
   3. No
   4. Yes
   5. Reboot

2. **Bluetooth deaktivieren, da dieser GPIO-UART-Ports verwendet.**

```
sudo nano /boot/config.txt
```

   Folgende Zeile muss ergänzt werden:

```
dtoverlay = disable-bt
```

3. **GPIO-UART-Ports für als serielle Schnittstelle konfigurieren.**

```
sudo nano /boot/config.txt
```

   Folgende Zeile muss ergänzt werden:

```
enable_uart = 1
```

Da der Raspberry Pi 4 über vier weitere GPIO-UART-Ports verfügt, müssen diese durch die folgenden Zeilen separat aktiviert werden:

```
dtoverlay = uart2
dtoverlay = uart3
dtoverlay = uart4
dtoverlay = uart5
```

4. **Hardware-Verbindung zwischen Raspberry Pi und Arduino Mini herstellen.**
   Da der Arduino Mini Pro die Daten mit einer Spannung von 5V sendet, der Raspberry Pi am GPIO-Eingang jedoch nur für eine Maximalspannung von 3,3V ausgelegt ist, muss die Spannung des Signals auf 3,3V geregelt werden.
   Hierzu wird ein Spannungsteiler wie folgt verwendet:

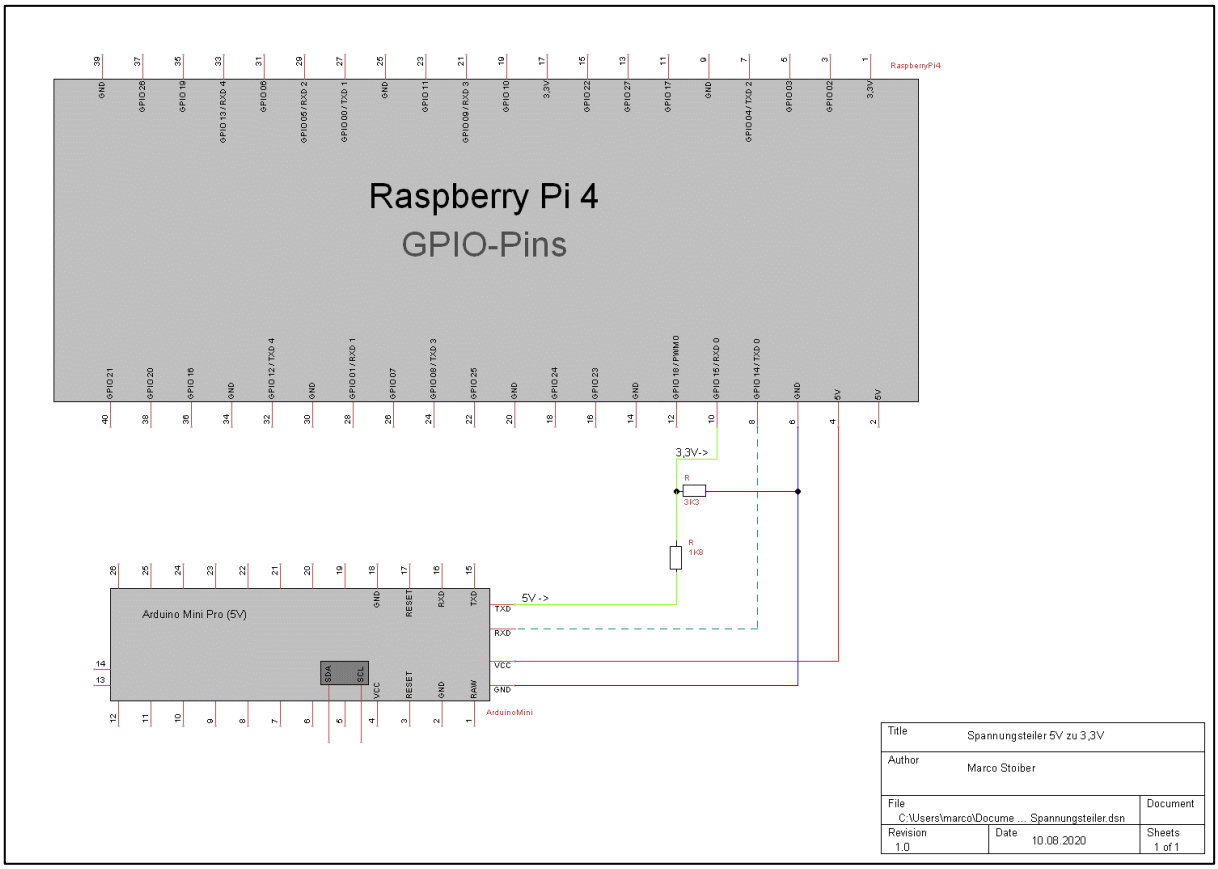


Abbildung 7: Schaltplan für Spannungsteiler (5V → 3,3V)

3.2 Klasse zur seriellen Kommunikation

Zur einfacheren Verwendung der seriellen Schnittstellen aus dem Programm heraus, wird eine eigene Klasse zur seriellen Kommunikation verwendet. Diese wurde in der Programmiersprache C implementiert.

Nachfolgend werden die Kernelemente der Klasse kurz dargestellt.

1. **Einbinden der Bibliotheken:**

```
#include <wiringPi.h>
#include <wiringSerial.h>
```

2. **Öffnen eines seriellen Ports:**

```
wiringPiSetup();

int fileDescriptor = serialOpen(char* port, int baudrate);
```

3. **Auslesen eines Zeichens des seriellen Portes:**

```
If(serialDataAvail(fileDescriptor))
{
      char input = serialGetChar(fileDescriptor);
}
```

Prüft, ob Daten am Eingang vorhanden sind und liest anschließend ein Zeichen aus.

Folgende Tabelle zeigt die verwendeten UART-Ports und deren Adressen:

Port	**Adresse**	**TX-Pin**	**RX-Pin**
USB UART	"/dev/ttyUSB0"	-	-
GPIO UART0	"/dev/ttyAMA0"	GPIO14	GPIO15
GPIO UART1	"/dev/ttyAMA1"	GPIO0	GPIO1
GPIO UART2	"/dev/ttyAMA2"	GPIO4	GPIO5
GPIO UART3	"/dev/ttyAMA3"	GPIO8	GPIO9
GPIO UART4	"/dev/ttyAMA4"	GPIO12	GPIO13

3.3 Anmerkungen zum Löten der Verbindungen

Da Hardwarefehler im Nachhinein oft sehr schwer zu finden sind und man gleichzeitig Softwarefehler nicht ausschließen kann, lohnt es sich die Hardware-Entwicklung mit besonderer Präzision durchzuführen.

Beim Löten der Verbindungen zwischen den Arduino Minis und den Sensoren als auch zwischen dem Raspberry Pi und den Arduinos sind ein paar Dinge zu beachten, um mögliche Fehler bzw. Fehlerursachen vorzubeugen:

1. **Den Platz auf den Lochrasterplatinen großzügig auslegen.**
   Also besser eine Reihe frei lassen, als einen Kurzschluss zu riskieren.

2. **Regelmäßiges durchmessen aller Lötverbindungen.**
   Auf diese Weise kann man bereits früh Kurzschlüsse oder falsche gelötete Verbindungen erkennen und ernsthaften Schaden an Bauteilen vermeiden.

3. **Befestigen der gelöteten Bauteile**
   Um Wackelkontakte oder Kontaktabbrüche zu vermeiden ist es wichtig alle Bauteile (z.B. auf einer dünnen Holzleiste) mit einander zu befestigen. Die Kabel sind zur Stromversorgung bzw. Datenübertragung da, und nicht zum Zusammenhalten der Bauteile. Deswegen sollte man bei allen Kabeln auch für die nötige Zugentlastung sorgen.

3.4 Einrichten des Wifi-Moduls - ESP8266

Zur Verwendung des ESP8266 wird die offizielle Anleitung des Herstellers *Simac Electronics* verwendet (Simac Electronics GmbH, 2020)[4].

Als Entwicklungsumgebung wird, wie vom Hersteller empfohlen, die Arduino IDE verwendet.

Im Rahmen dieses Projekts wurde stets ein Arduino Uno zum Hochladen von Programmcode auf den ESP8266 genutzt.

[4] https://joy-it.net/files/files/Produkte/SBC-ESP8266/SBC-ESP8266-Anleitung-20200626.pdf

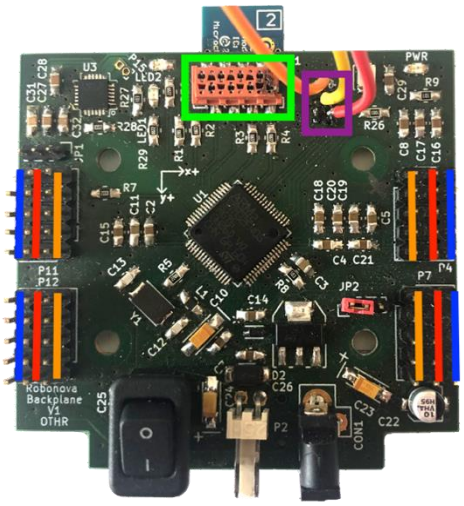
4 RoboNova

In diesem Kapitel geht es um den RoboNova Roboter von HiTec. Es werden alle Hardwarebestandteile des RoboNova sowie benötigte Hardwaretools vorgestellt. Anschließend wird erklärt, wie man eine Entwicklungsumgebung für den RoboNova auf dem Computer aufsetzt und diese nutzt. Danach wird die selbst entwickelte Treibersoftware vorgestellt. Zuletzt wird beschrieben wie man die Firmware auf den Servomotoren neu programmieren kann.

4.1 Hardwareübersicht

In der Kiste des RoboNova ist eine große Sammlung an Bauteilen und ähnlichem von vergangenen Projekten. Im Folgenden wird, die für dieses Projekt benötige, Hardware vorgestellt.

4.1.1 Backplane



Legende:

Grün – Debug Header

Lila – UART Header BLE- /WIFI-Modul

Blau – GND

Rot – V_{CC}

Orange - Signal

Abbildung 8: Backplane RoboNova

Die orginale Backplane des RoboNova wurde durch eine Custom-Backplane ersetzt. Sie wurde von Florian Laufenböck entwickelt.

Auf dem Board ist ein 32bit ARM-Cortex Microcontroller STM32F405RG verbaut. An diesem ist ein MPU-6050 über I^2C und ein RN4020 BLE-Modul über UART angebunden. Die UART Leitungen des BLE-Moduls führen zu dem lila Header. Somit kann entweder das BLE-Modul (Achtung: gekreuzte RX und TX Leitungen) oder ein anders Modul an der UART Instanz genutzt werden. Für die Servomotoren werden 4 UART Instanzen des Microcontroller genutzt. Jede Instanz führt zu einem Header an welchem je 5 Servomotoren angeschlossen werden können.

Aktuell sind 2 Backplanes bestückt. Davon funktioniert bei einer Platine nur ein „Servomotor UART Header“ und bei der anderen gar keiner.

4.1.2 Servomotoren

Abbildung 9:HiTec Servomotor

Für die Gelenke werden HiTec HSR-8498HB Servomotoren genutzt. Diese bieten ca. 190° Drehwinkel. Die original Steuerchips wurden durch ATmega 328P ersetzt. Für den detaillierten Aufbau im Servomotor siehe Dokumentation Servomotoren. Die Chips können mit einem bestimmten Aufbau über UART programmiert werden, somit müssen sie nicht abgelötet werden. Mehr dazu im Kapitel 4.5.

4.1.3 WIFI Modul

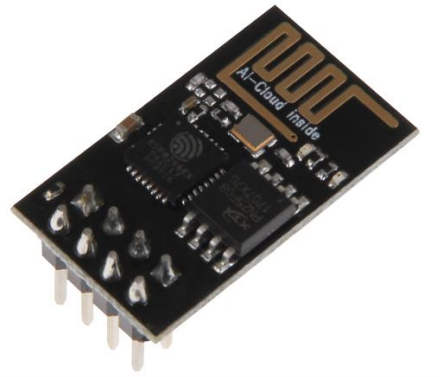


Abbildung 10: ESP6266 Modul

Zur Kommunikation mit dem Backend wurde ein ESP8266 Modul von JOY-IT verwendet. Das Modul wurde für eine unidirektionale Kommunikation zum RoboNova programmiert. Somit wurde das WiFi Modul an den UART Header angeschlossen, welcher für das BLE-Modul gedacht ist.

4.1.4 Jlink Segger Debugger

Abbildung 11: J-Link Segger Debugger

Mit dem j-link Segger Debugger kann man per SWD Interface den Microcontroller flashen und debuggen. Dazu schließt man ihn an dem vorgesehenen Header auf der Backplane an (auf schwarze Markierung achten).

Es gibt 2 Debugger, der Debugger mit den offenen Leitungen scheint nicht zu funktionieren. Der zweite Debugger funktioniert, jedoch muss man drauf achten, dass das Kabel zwischen Debugger und Microcontroller gerade liegt, da es scheinbar einen Wackelkontakt hat.

4.1.5 FTDI FT232RL USB-UART Modul

Abbildung 12: USB-UART Modul

Dieses Modul ist Teil der Toolchain für das Servomotor flashen. Es ermöglicht die Ausgabe von UART Nachrichten über den PC. Dazu schließt man es mit einem Mini-USB Kabel am PC an. Auf dem PC verbindet man sich mit der seriellen Schnittstelle (über das Terminal, Putty,

CoolTerm oder ähnlichem) und kann damit Nachrichten senden und empfangen. Man kann zwischen einem Signalpegel von 5V und 3,3V entscheiden.

4.1.6 Converter Platine

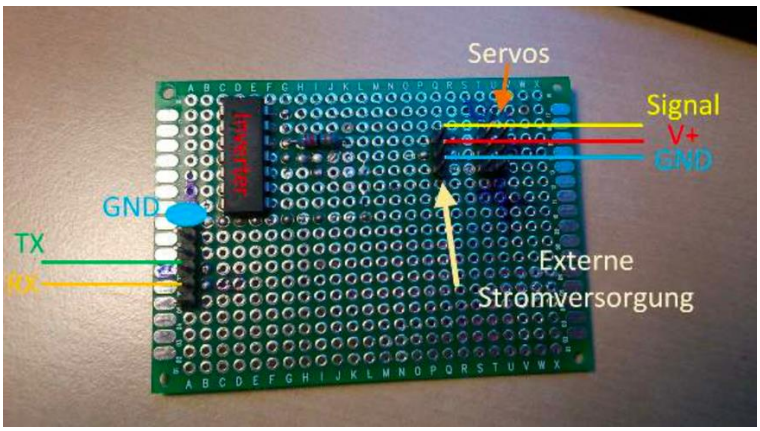


Abbildung 13: Converter Platine

Die Converter Platine wird ebenfalls zum flashen der Servomotoren benötigt. Bauartbedingt nutzt der Servomotor ein invertiertes Signal. Deswegen ist auf der Platine ein HEX Inverter von ST verbaut. Dieser invertiert das eingehende Signal und gibt es an die Signalleitung des Servomotors weiter. Zusätzlich gibt es noch einen Header, um eine Stromversorgung für die Servomotoren anzuschließen.

Die Platine funktioniert nicht. Entweder sind Lötstellen defekt oder der HEX Inverter ist kaputt.

4.2 Aufbau

Dieser Abschnitt beschreibt den Hardwareaufbau bezogen auf den RoboNova.

Für dieses Projekt werden die Servomotoren und das WiFi-Modul genutzt.

Im RoboNova sind 16 Servomotoren verbaut. Je 3 Motoren pro Arm und je 5 Motoren pro Bein. Die große Anzahl ist notwendig, da sich die Servomotoren nur in einer Achse bewegen können. Das menschliche Schulter-, Hüft- und Fußgelenk kann sich aber in mehrere Achsen bewegen, was mit 2 Servomotoren bestmöglich repliziert werden kann.

Durch den Ausfall der UART Header auf einer Backplane können nicht alle Motoren angesteuert werden. Mit einem UART-Splitter werden nur die 6 Servomotoren der Arme an dem einzig funktionierenden UART Header der zweiten Backplane angeschlossen.

Das WiFi-Modul ist an dem UART Header des BLE-Moduls angeschlossen.

4.3 Entwicklungsumgebung

In diesem Abschnitt wird erklärt wie eine neue Entwicklungsumgebung erstellt werden kann.

In den älteren Dokumentationen ist beschrieben wie man eine Entwicklungsumgebung auf Windows mittels STM32 CubeMX und STM32 Cube Workbench erstellen kann. In diesem Projekt wurde statt der Eclipse basierenden Cube Workbench VS Studio Code auf MacOS genutzt.

4.3.1 Downloads und Installationen

Auf MacOS muss man Xcode installieren, da mit diesem der C-Compiler und das make Tool installiert wird. Xcode kann im Appstore installiert werden

Des Weiteren muss man die ARM GCC Toolchain installieren[5].

Für den Debugger wird das J-Link Segger Software Package benötigt.[6]

Zur Konfiguration des Basiscode wird STM32 CubeMX genutzt[7].

Zuletzt benötigt man Visual Studio Code[8] und die Cortex-Debug Erweiterung. Die Erweiterung kann in VS Code über den Extensions Marketplace installiert werden.

4.3.2 Projekt erstellen

Um ein neues Projekt zu erstellen startet man STM32 CubeMX und wählt „New Project from MCU“. In der folgenden Liste wählt man „STM32F405RG“. Da VS Code keinen automatischen make Prozess hat muss in CubeMX unter Project Manager → Project → Toolchain/IDE „makefile“ ausgewählt werden. Für den richtigen Debugmodus muss unter Pinout & Configuration → System Core → SYS → debug „JTAG (4 pins) gewählt werden. Nach diesen Schritten ist die Basiskonfiguration abgeschlossen und das CubeMX Project kann gespeichert und generiert werden.

Die von CubeMX generierte Umgebung wird mit Visual Studio Code geöffnet. Dort kann entweder ein Task für den build-Prozess erstellt werden oder händisch in der Konsole mit „make“ der Code kompiliert werden. Als nächstes muss unter Run → Open configuration eine „cortex-debug“ Konfiguration erstellt werden. Hier sind das richtige Interface (SWD), das Device (STM32F405RG), die Executable (.elf file des build-Prozess) und der Servertype (jlink)

[5] https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads
[6] https://www.segger.com/downloads/jlink/
[7] https://www.st.com/en/development-tools/stm32cubemx.html
[8] https://code.visualstudio.com/download

wichtig. Auf der Wiki Seite der Cortex Debug Erweiterung[9] sind einige Beispiele gegeben wie die Konfiguration angepasst werden kann.

Nach diesen Schritten sollte die Toolchain stehen. Mit CubeMX kann der Microcontroller und dessen Modulinstanzen konfiguriert werden und der Code generiert werden. In VS Code kann Nutzer-Code ergänzt werden, aus dem Projektcode eine Executable erstellt werden und der Microcontroller programmiert und gedebugged werden.

4.4 Software

In diesem Abschnitt wird die gesamte Software des RoboNova erklärt. Es wird gestartet mit der Konfiguration des Microcontroller und dessen Modulinstanzen. Anschließend werden die selbstgeschriebenen Treiber für die Servomotoren und das WiFi Modul vorgestellt.

4.4.1 Microcontroller Konfiguration

Allgemein gesehen wurden für dieses Projekt nur zwei Arten an Konfigurationen benötigt. Es wurden alle Peripheriegeräte über UART angeschlossen. Somit ergibt sich eine Konfiguration für die UART Instanz des WiFi Modul und ähnliche Konfigurationen für die UART Instanzen der Servomotoren.

4.4.1.1 WiFi UART

Das WiFi Modul ist an dem Header angeschlossen, welcher mit den USART1 Pins des Microcontrollers verbunden ist.

Das WiFi Modul sendet Nachrichten mit einer Baudrate von 115200 Bit/s, einer 8 Bit Wordlength, einem Stopbit und sendet keine Paritätsinformation.

Der Treiber für das WiFi Modul nutzt DMA um Nachrichten zu empfangen und in einem Buffer zu speichern. Dafür muss der RX DMA konfiguriert werden und das globale Interrupt des DMA aktiviert werden. Da es sich um den empfangenden DMA handelt wurde Peripherie zu Speicher als Richtung gewählt. Der DMA wird im normalen Modus verwendet und es wird eine Datenbreite von einem Byte gewählt.

4.4.1.2 Servomotor UART

Die UART Instanzen 2, 3, 4 und 6 führen jeweils zu einem der Servomotor Header auf der Backplane. Will man diese nutzen müssen die Instanzen konfiguriert werden.

[9] https://github.com/Marus/cortex-debug/wiki

Das Protokoll der Servomotoren ist so ausgelegt, dass eine Baudrate von 19200 Bit/s, eine 8 Bit Wordlength, 2 Stopbits und keine Paritätsinformationen benötigt werden.

Der Servomotor Treiber benötigt die Aktivierung der globalen Interrupts.

4.4.2 Servomotor Treiber

Der Servomotor Treiber wurde im Rahmen eines vorherigen Projekts von einem anderen Team entwickelt.

Die Implementierung ist dreigeteilt. Auf der abstrakten Schicht gibt es die Konfiguration und das ServoInterface, auf der Hardwareschicht ist der ServoDriver.

Im ServoDriver ist alles was auf Hardware Abstraction Layer (HAL) geschieht implementiert. Dabei werden die von ST gebotenen HAL_UART Treiber genutzt. Der ServoDriver kümmert sich um die HAL_UART Callbacks und das Senden und Empfangen von Nachrichten mit HAL_UART Interrupt Funktionen.

Im ServoInterface werden die Funktionen zur Servomotoren-Ansteuerung implementiert. Die Funktionen erstellen Nachrichten gemäß dem Servomotor Protokoll und senden diese mithilfe der ServoDriver Funktionen. Das ServoInterface bietet Funktionen, um den Servomotor zu aktivieren, seine ID auszulesen, die Kommunikation zu zurückzusetzten und dessen Position setzten und auszulesen.

Das ServoInterface benötigt Informationen an welchen Servomotor die Nachricht gesendet werden soll. Dazu muss es wissen an welcher UART Instanz der Servomotor angeschlossen ist und welche ID dieser hat. Diese Information ist in einer Struktur gespeichert, welche jedem RoboNova Gelenk die Informationen zuordnet. Zusätzlich sind Defines implementiert welche den Code leichter lesbar macht, da jedem Gelenk ein Name gegeben wurde, anstatt nur Nummern zu verwenden.

4.4.3 WiFi Treiber

Der WiFi Treiber besteht aus einer Funktion, da nur das Empfangen von Daten benötigt und implementiert wurde. In dieser Funktion wird die Nachricht aus dem DMA Buffer gelesen und gemäß des Nachrichtenschemas die Informationen extrahiert in eine globale Variable.

4.5 Servomotoren

Jeder Servomotor ist so programmiert, dass er passend zum Protokoll welches, in der Servomotor Dokumentation beschrieben ist, reagiert. Zusätzlich besitzt jeder Servomotor eine

eindeutige ID, dadurch kann er erkennen ob die Nachricht auf dem UART-Bus an ihn gesendet wurde.

4.5.1 Programmieren

Dieses Kapitel basiert auf theoretischem Wissen. Durch die defekte Converter Platine konnte leider nie ein Servomotor in Praxis programmiert werden.

Allgemein werden Microcontroller über Evaluationsboards oder ähnlichem auf denen ein Sockel oder der Controller fest verlötet ist programmiert. Damit die ATmega Chips nicht für jeden Programmiervorgang aus dem Motor ausgelötet werden müssen wurde von Florian Laufenböck ein eigener Bootloader entwickelt. Dieser ermöglicht es über UART die Servomotoren zu programmieren.

Dazu wird das USB-UART Modul und die Converter Platine benötigt. Das USB-UART Modul wird per USB an dem PC angeschlossen und die RX, TX und GND Pins mit den passenden Pins auf der Converter Platine verbunden. Nach Anschluss einer externen Stromversorgung sollte man mit zB. CoolTerm Nachrichten an die Signalleitung des Servomotor Header senden können. In CoolTerm müssen folgende Einstellungen gesetzt werden: Baudrate: 19200; Data Bits: 8; Stop Bits: 2; Parity: none; transmit character delay: 10ms.

Zum programmieren des Servomotors sollte man diese Schritte ausführen:

1. Nur EINEN Servomotor auf der Converter Platine anstecken
2. Die Stromversorgung anschalten
3. Sofort innerhalb 1s beliebig viele 's' verschicken.
4. Mit einer beliebigen Nachricht testen, ob man im Bootloader ist(z.B.0x420x00). Liest man 0x42 zurück, passt alles. Ansonsten die Kommunikation mit 0x00 0x00 0x00 0x00 zurücksetzen und diesen Schritt wiederholen.
5. Mit der Nachricht 0x70 0x00 den Flash-Modus erreichen. Überprüfen ob 0x70 zurückgelesen werden konnte.
6. Die gesamte hex-Datei (vom ersten bis zum abschließenden FF) kopieren und per Copy-Paste im Text-Modus der seriellen Konsole einfügen. Die Übertragung kann durchaus etwas länger dauern, da nach jedem Zeichen 10ms gewartet wird.

7. Ist die Übertragung abgeschlossen, kann durch die Nachricht 0x69 0x00 überprüft werden, ob alles gut gegangen ist. Wird der Wert 0x01 zurückgelesen, passt alles. Wird 0xFF zurückgelesen, sollte man den Aufbau überprüfen und zu Punkt 5) zurückkehren.
8. Den Bootloader mit dem Befehl 0x71 0x00 verlassen. Sicherstellen, dass der Wert 0x71 zurückgelesen werden konnte.

Detailliertere Informationen zum Bootloader kann man in der Servomotor Dokumentation finden.

5 Simulation

Im Folgenden werden der Aufbau und das Einrichten der Simulation beschrieben. Dabei wird die Inbetriebnahme mithilfe des TCP_Test_Servers beschrieben.

5.1 Komponenten

Die Simulation besteht aus 3 Komponenten. Die Blender Projekte für das Mesh und Skelett der Menschen und Roboter Darstellung, dem TCP_Test_Server der für ein Replay von Nachrichten aus dem Backend benutzt werden kann und dem Unity Projekt, das das Simulation Setup beinhaltet und eine TCP-Client Implementierung für die Verbindung zum Backend.

5.2 Einrichten der Simulation

Als erstes müssen die IP-Adresse bzw. der Port des TCP_Test_Servers, die Replay Dateien und der Modus des Testservers festgelegt werden. Für das Festlegen der IP-Adresse muss in der Constants Klasse des TCP_Test_Server der Port auf den gewünschten Port festgelegt werden. Der Server ist dann auf diesem Port lokal erreichbar. Der Modus kann zu Beginn der Program.cs Datei mit

```
#define RoboNovaMode
```

oder

```
#define HumanMode
```

festgelegt werden. Abhängig davon kann auch der Replay Dateipfad in der Constants Klasse angepasst werden. Nach den Änderungen kann der Server gestartet werden und wartet auf einen Client.

Als zweites muss die Simulation eingerichtet werden. Dazu muss das Unity Projekt mit Unity geöffnet werden. Bei der Simulation des Menschen muss für jeden Knochen des AnimationCharacter das „Bone Manipulation Script“ als Komponente hinzugefügt werden.

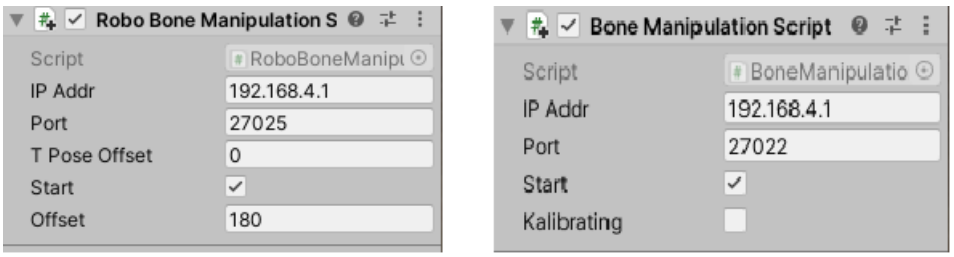


Abbildung 14: Ausschnitt aus der Unity Komponenten Kalibrierung

In dieser Komponente müssen die IP-Adresse und der Port des Servers angegeben werde. Außerdem muss Start angehakt werden, wenn dieser Knochen genutzt werden soll. Für die Simulation des Roboters muss nur bei der Armature des RoboNova das „Robo Bone Manipulation Script“ hinzugefügt werden. Auch hier müssen die Parameter angepasst werden. Wobei die Offsets nur für die Kalibrierung genutzt werden. Für die Simulation des Roboters müssen weitere Anpassungen an Mesh und Skript getätigt werden und ist daher nicht sinnvoll nutzbar abseits von Replays. Zu beachten ist, dass der TCP_Test_Server nur für einen Knochen oder die Simulation des Roboters gedacht ist. Unter dem Tab „Game“ kann zwischen Drei Verschiedenen Perspektiven mithilfe von Display 1, 2 und 3 gewechselt werden.

5.3 Anmerkung

Das Kalibrieren war durch Fehler im Character designe erschwert. Daher muss auf folgendes geachtet werden.

1. **Mesh Genauigkeit an Gelenken erhöhen.**
   Das verhindert ungewollte Verformungen bei Bewegungen der Figur.
2. **Knochen des Meshs genau ausrichten.**
   Das führt zu besser kontrollierbarem Verhalten bei Rotationen der Knochen

6 Stundenlisten

6.1 Stundenliste Maximilian Kritzenthaler

Aktivität	**Stunden**
Wöchentliche Meetings	35
Sprints	50
Entwicklungsumgebung einrichten + Flashen	5
Einlesen ATmega328P, MPU6050, I2C, UART	10
Schaltplan auf Breadboard umsetzen	1
Sensordaten Visualisierung	2
Firmware implementieren + Dataframe entwickeln	25
Debugging	6
Testen	7
Dokumentation erstellen (1 Sensorik)	4
Gesamt:	**145**

6.2 Stundenliste Eduard Schröder

Aktivität	**Stunden**
Wöchentliche Meetings	35
Gemeinsame Arbeitssprints	50
Einarbeitung Sockets	10
Einarbeitung Threads	4
Aufbau der Komponenten	5
Backend Softwareentwicklung	25
Wifi-Modul Softwareentwicklung	2
Einrichtung Banana Pi	5
Dokumentation erstellen(2 Backend)	7
Gesamt:	**143**

6.3 Stundenliste Marco Stoiber

Aktivität	**Stunden**
Wöchentliche Meetings	35
Gemeinsame Arbeitssprints	50
Recherche zu UART, RS232, RS485	4
Recherche zu Wifi-Modul ESP8266	4
Einrichten des ESP8266	5
Einrichten & Programmieren der UART-Kommunikation	4
Entwickeln des Hardware-Layouts (Schaltpläne, Befestigung der Bauteile, etc.)	10
Löten der Verbindungen zwischen Sensoren & Arduino Minis	25
Löten der Verbindungen zwischen Arduino Minis & Raspberry Pi	15
Dokumentation erstellen(3 Kommunikation)	4
Gesamt:	**156**

6.4 Stundenliste Felix Dollinger

Aktivität	**Stunden**
Wöchentliche Meetings	35
Sprints	30
Entwicklungsumgebung erstellen	10
Einlesen RoboNova und Servomotor Ansteuerung	7
RoboNova programmieren und erste Servomotor Ansteuerung	2
Servomotor Treiber anpassen	1
Einlesen und Aufbau für das Servomotor flashen erstellen/nachvollziehen	5
Vergebliche Versuche die Servomotoren zu flashen und Fehlersuche	16
Kommunikation mit Florian Laufenböck zur Fehlersuche bzgl. Servomotor flashen	4
Treiber für WiFi Modul erstellen	5
Fehlersuche aufgrund von (teil)defekter Hardware	10
Dokumentation erstellen(4 RoboNova)	11
Gesamt:	**136**

6.5 Stundenliste Tobias Gilgenreiner

Aktivität	**Stunden**
Wöchentliche Meetings	35
Gemeinsame Arbeitssprints	40
Einarbeitung Blender	20
Einarbeitung Unity	10
Implementieren TCP_Test_Server	5
Implementierung Unity-Skripte	10
Testen	10
Organisatorisches	2
Dokumentation erstellen(5 Simulation)	4
Gesamt:	**136**

7 Abbildungsverzeichnis