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



Inhalt

[1 Sensorik 2](#_Toc48835453)

[1.1 Systemhardware 2](#_Toc48835454)

[1.2 Einrichtung der Entwicklungsumgebung 4](#_Toc48835455)

[1.3 Entwicklung der Firmware 5](#_Toc48835456)

[2 Backend 7](#_Toc48835457)

[2.1 Systemhardware 7](#_Toc48835458)

[2.2 Einrichtung des Access Point 7](#_Toc48835459)

[2.3 Einrichten weiterer Systeme 9](#_Toc48835460)

[2.4 Programmausführung des zentralen Backend 10](#_Toc48835461)

[2.5 Programmausführung weiterer Systeme 10](#_Toc48835462)

[2.6 Zeitlicher Ablauf der Ausführungen 11](#_Toc48835463)

[2.7 Software des WLAN-Modul 11](#_Toc48835464)

[2.8 Mögliche nachträgliche Änderungen 11](#_Toc48835465)

[2.9 Einrichten der Verbindung zwischen Backend und Arduino Mini 12](#_Toc48835466)

[2.10 Klasse zur seriellen Kommunikation 13](#_Toc48835467)

[2.11 Anmerkungen zum Löten der Verbindungen 14](#_Toc48835468)

[3 RoboNova 16](#_Toc48835469)

[3.1 Einrichten des Wifi-Moduls - ESP8266 16](#_Toc48835470)

[4 Stundenlisten 16](#_Toc48835471)

[4.1 Stundenliste Marco Stoiber 16](#_Toc48835472)

# Sensorik

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

## 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.

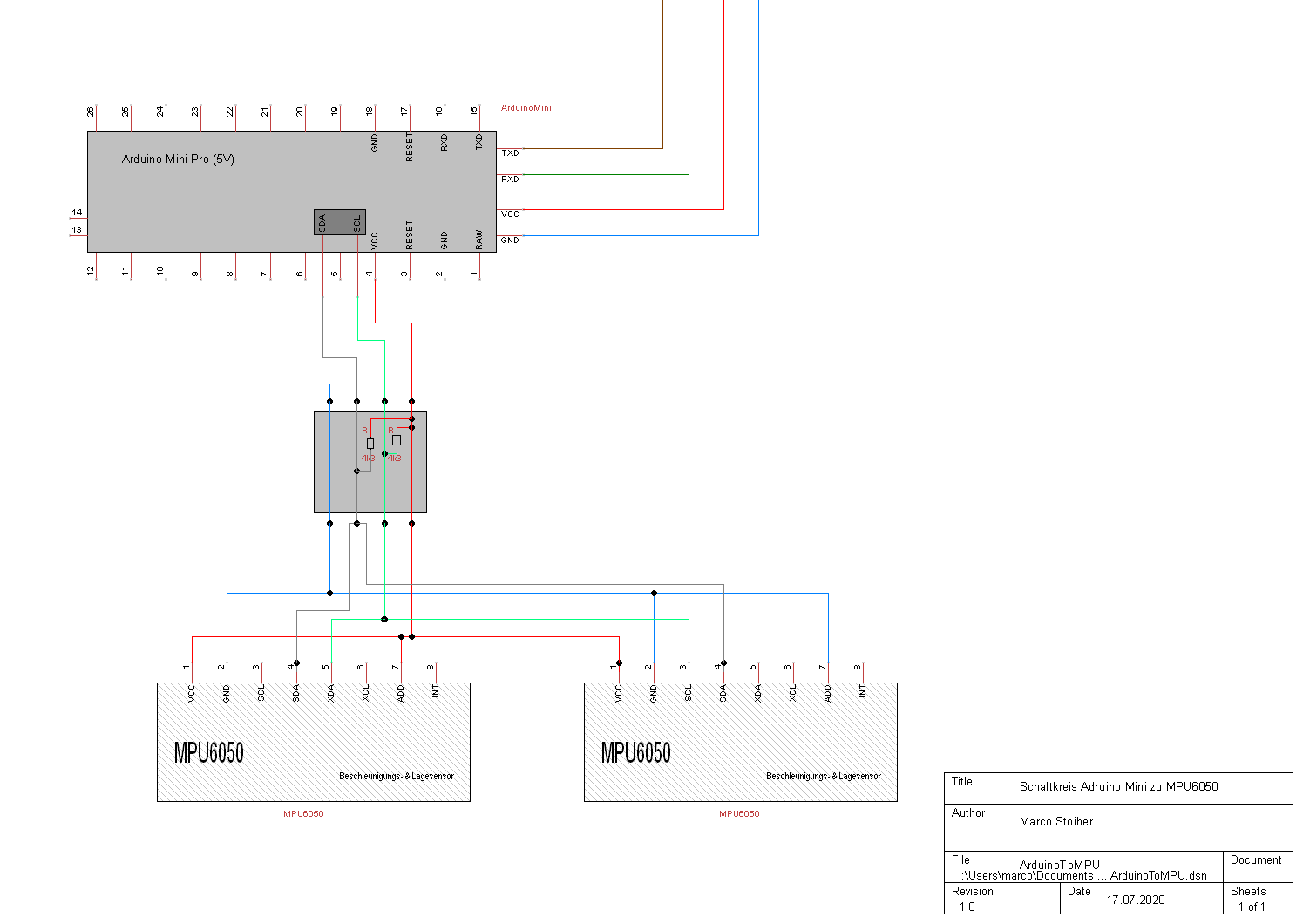
­­

Abbildung : 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:

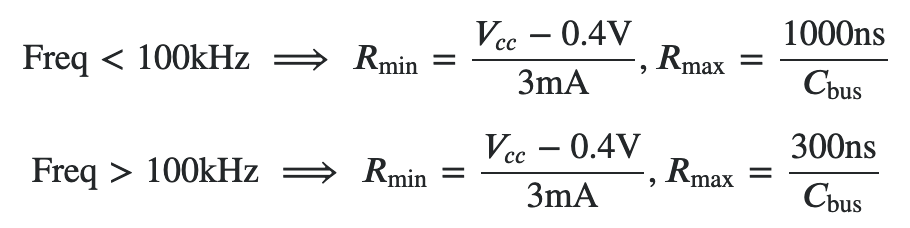


Abbildung :Formel Widerstandberechnung I2C

Wir haben für den I2C die Standard Übertragungsrate von 100kHz verwendet und liegen damitbei 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.

## 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]](#footnote-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.

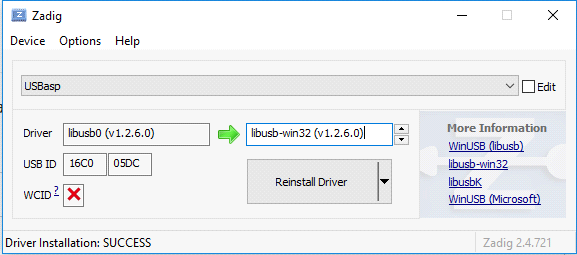


Abbildung :USBasp Treiber Installation

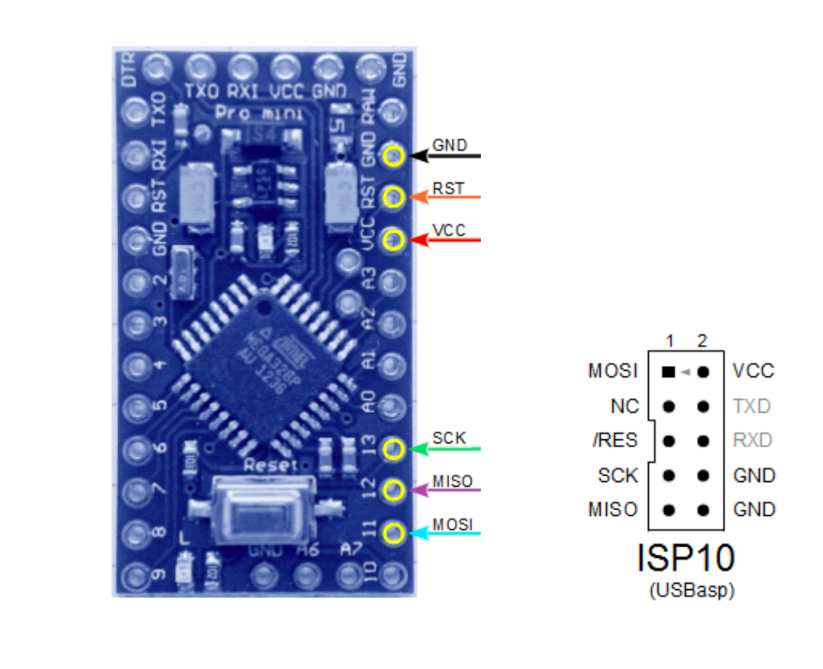
Zum Flashen sind male to female Stecker notwendig, die wie folgt verbunden werden müssen:

Abbildung : Arduino Mini Pro - USBasp Verbindung

## 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.

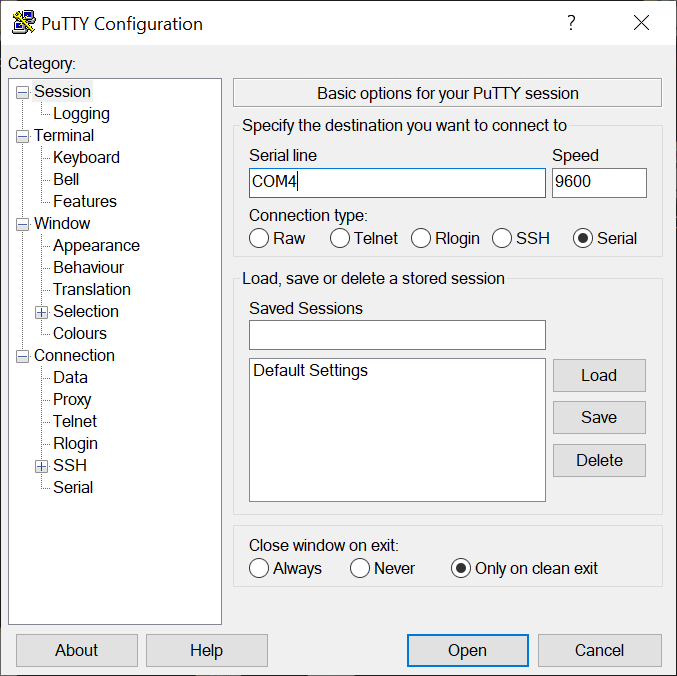
****

Abbildung : 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.

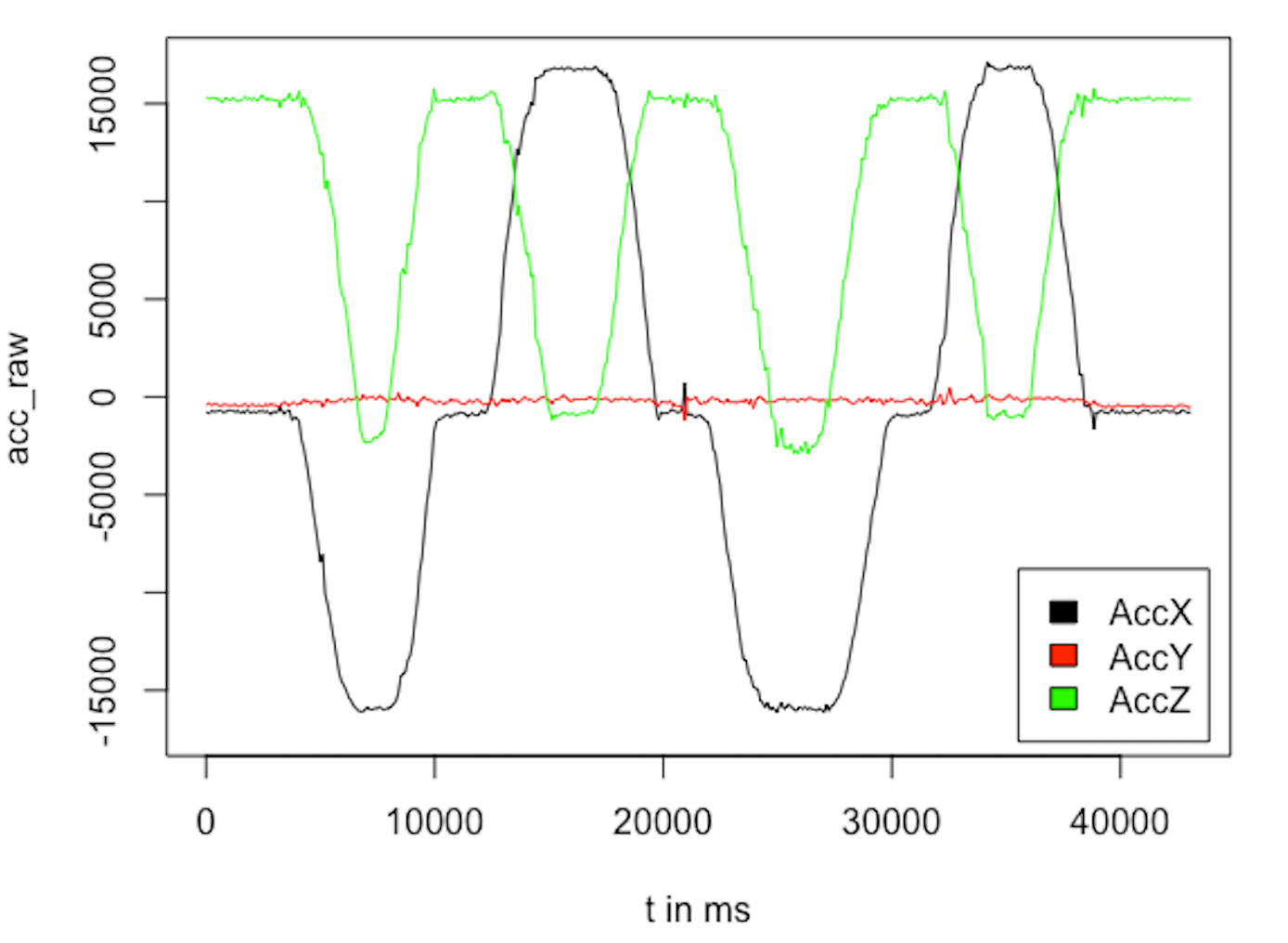


Abbildung : Accelerometer Links-Rechts-Rotation

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.

# 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.

## 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 *1.4**Programmausführung des zentralen Backend*für das Access Point System und in *1.5 Programmausführung weiterer Systeme* für weitere verbundene Systeme erklärt werden.

## 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]](#footnote-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

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

sudo apt install dnsmasq

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

sudo DEBIAN\_FRONTEND=noninteractive apt install -y netfilter-persistent iptables-persistent

1. **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

1. **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

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

1. **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

sudo rfkill unblock wlan

1. **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.

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

sudo systemctl reboot

## 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.

## 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]](#footnote-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.

## 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.

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.

## 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

## 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.

## 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.

1. **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.

## 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.

Auch 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

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

sudo nano /boot/config.txt

Folgende Zeile muss ergänzt werden:

dtoverlay = disable-bt

1. **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 separate aktiviert werden:

dtoverlay = uart2  
dtoverlay = uart3

dtoverlay = uart4

dtoverlay = uart5

1. **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.

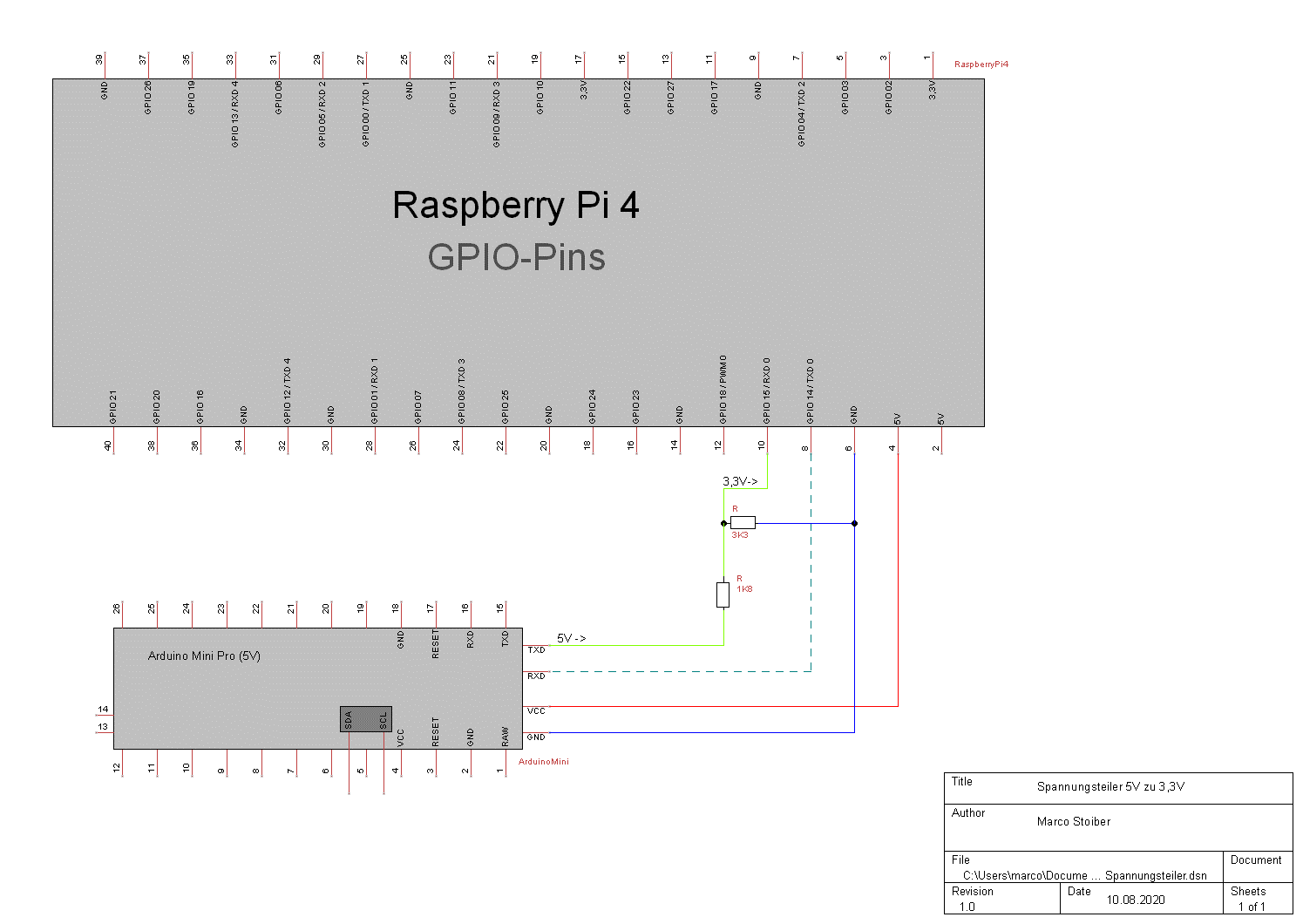
Hierzu wird ein Spannungsteiler wie folgt verwendet:

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

## 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>

1. **Öffnen eines seriellen Ports:**

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

1. **Auslesen eines Zeichens vom seriellen Port:**

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 |

## Anmerkungen zum Löten der Verbindungen

Da Hardwarefehler im Nachhinein oft sehr schwer zu finden sind und man meist 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.

1. **Regelmäßiges durchmessen aller Lötverbindungen.**

Auf diese Weise kann man bereits früh Kurzschlüsse oder falsche gelötete Verbindungen erkennen und ernsthaften Schaden an Bauteilen vermeiden.

1. **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.

# RoboNova

## Einrichten des Wifi-Moduls - ESP8266

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

# Simulation

Im Folgenden werden der Aufbau und das Einrichten der Simulation beschrieben. Dabei wird die Inbetriebnahme mithilfe des TCP\_Test\_Servers beschrieben.

## 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.

## 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.

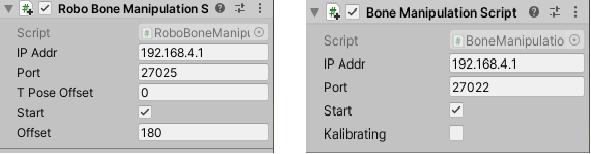


Abbildung : Ausschnitt aus der Unity Komponenten Kalibrierung

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.

## 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.

1. **Knochen des Meshs genau ausrichten.**

Das führt zu besser kontrollierbarem Verhalten bei Rotationen der Knochen

# Stundenlisten

## 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 | 4 |
| **Gesamt:** | **145** |

## 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 | 7 |
| **Gesamt:** | **143** |

## 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 |
| Schreiben der Dokumentation | 4 |
| **Summe:** | **156** |

## Stundenliste Felix Dollinger

## 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 |
| Schreiben der Dokumentation | 4 |
| **Gesamt:** | **136** |

1. <https://medium.com/manash-en-blog/setting-up-atmel-studio-for-usbasp-and-avr-programming-802bb4dcf1e9> [↑](#footnote-ref-1)
2. <https://github.com/raspberrypi/documentation/blob/master/configuration/wireless/access-point-routed.md> [↑](#footnote-ref-2)
3. <https://docs.microsoft.com/de-de/cpp/linux/connect-to-your-remote-linux-computer?view=vs-2019> [↑](#footnote-ref-3)
4. <https://joy-it.net/files/files/Produkte/SBC-ESP8266/SBC-ESP8266-Anleitung-20200626.pdf> [↑](#footnote-ref-4)