Projekt Automatisierungstechnik

Inhaltsverzeichnis

Verwendete Betriebssysteme und Softwareversionen:

PC (SSDToGo): Ubuntu 20.04 LTS mit ROS Neotic (Desktop Full) und MATLAB 2019b (MATLAB 9.7, Simulink 10.0, ROS Toolbox 1.0, Embedded Coder 7.3, Simulink Coder 9.2, MATLAB Coder 4.3, Control System Toolbox 10.7)

BeagleBone: Ubuntu 20.04.2 LTS ohne grafische Benutzeroberfläche, Python 3.5.8 und ROS Noetic. Flashen des eMMC mit Debian Stretch Flasher-Image `bone-eMMC-flasher-debian-9.12-iot-armhf-2020-03-01-4gb` damit SD-Kartenimage Bootpriorität hat.

0 Lernziele Projekt Automatisierungstechnik (PAT)

- Umgang mit komplexen eingebetteten Systemen am Beispiel eines BeagleBone (BB).
- Umgang mit komplexen intelligenten Sensoren und deren Programmierschnittstellen (API).
- Erlernen einfacher Grundlagen im Umgang mit Linux als Betriebssystem eines eingebetteten Systems, insbesondere das Einbinden der Schnittstellen zur I/O-Peripherie.
- Erlernen der Kommunikation zwischen einem Linux-basierten eingebetteten System und folgender Arten von Sensoren: Binäre Sensoren, analoge Sensoren, digitale Sensoren mit UART-, I²C-, SPI- oder Netzwerkschnittstelle.
- Nutzung einfacher Python-Skripte zusammen mit vorhandenen Bibliotheken zum Testen der Sensorfunktion, -kommunikation und der Sensordatenfusion.
- Arbeiten mit der Middleware Robot Operating System (ROS): Dabei dessen Vorteile erfahren wie die Wiederverwendbarkeit von Code, die Peer-to-Peer-Kommunikation der als ROS-Knoten (Prozesse) abstrahierten Komponenten, die umfangreichen ROS-Entwicklungswerkzeuge und Tutorials/Internetdokus.
- Erfahren, was Echtzeitdatenverarbeitung für ein mechatronisches System bedeutet, wodurch diese begrenzt wird und wie sie verifiziert wird.
- Erlernen von Strategien zur Fehlersuche in einem komplexen mechatronischen System.
- Praxis im technischen Dokumentieren von Projektergebnissen und Präsentation derselben in einem Vortrag.
- Erkennen der Vorteile des „Rapid Prototyping“ durch das Verwenden des BeagleBone zusammen mit ROS, MATLAB/Simulink und Python im Vergleich zur Entwicklung unter C mit einer konventionellen IDE wie Eclipse.

- Erkennen der Vor- und Nachteile der modellbasierten Entwicklung von Filter- und Regelungsalgorithmen und deren automatischen Codeerzeugung mit Simulink. Erkennen des Praxisbezugs hiervon.
- Praxiserfahrung mit IP-Kommunikation zwischen Komponenten eines mechatronischen Systems, die teils auf unterschiedlicher Computerhardware basieren.
- Die Wirkung eines Kalman-Filters verstehen und diesen implementieren.
- Ein mechatronisches System bezüglich Hardware als auch Software *modularisieren* können.

Bekannte Bugs und mögliche Lösungswege dazu

Nur bei Nutzung eigenem Ubuntu Betriebssystem, also nicht Nutzung der SSDToGo: Falls beide Python-Versionen Python 2.7 und Python 3 auf dem PC installiert sind, sollte der Befehl `python` den Python3-Interpreter ausführen. Falls Python-Distribution Conda installiert ist, muss dies evtl. mit `conda deactivate` vor dem Verwenden von ROS deaktiviert werden.

Autostart des WLANs am BB funktioniert nicht obwohl unter `connmanctl` der Befehl `config wifi_xxx –autoconnect yes` unter `connmanctl` ausgeführt wurde. Workaround über Bash Skript `wifi_reset.sh`, das jede Minute via `Cron` ausgeführt wird.

Damit der BB von der SD-Karte bootet muss beim Einschalten der Stromversorgung die USR (USER)-Taste auf dem Grove-Cape gedrückt sein. Ansonsten bootet er vom eMMC, wenn sich dort noch die ab Werk alte Debianversion befindet. Nach Installation von z.B. von Debian Stretch 9.12 oder Debian Buster 10.2 als eMMC-Flasher auf dem eMMC bootet der BB standardmäßig von SD-Karte, wenn diese eingesteckt und bootfähig ist.

Python-Pakete sollten wenn nicht anders angegeben nur mit dem Befehl `pip3 install --user` installiert werden. Ansonsten kommt es zu Inkonsistenzen, zwischen der Ubuntu und Python-Paketverwaltung. Siehe `wiki.ubuntuusers.de/pip/` oder `github.com/pypa/pip/issues/5599`. Kann das gleiche Paket via `apt-get` installiert werden, so ist dieser Weg vorzuziehen.

Die für das PAT vorgesehene Grove-Stecker-Schnittstelle UART1/4 auf dem Grove Cape Version 2 funktioniert nicht mehr sobald TX z.B von einem angeschlossenen Arduino belastet wird. Die gleiche UART1/4-Verbindung über die Buchsenleiste funktioniert einwandfrei, wenn der Arduino vom BB via `5VSys` mit Strom versorgt wird. Hingegen funktioniert die UART2 Grove-Stecker-Schnittstelle auf dem BB Board oberhalb des SD-Kartenfachs einwandfrei.

Falls der BB ausschließlich über die USB-Verbindung mit dem PC mit Strom versorgt wird und mehrere Sensoren wie auch der WiFi-Dongle angeschlossen sind, kann dies zu Instabilitäten führen. Denn die USB-Stromversorgung durch den PC ist meist auf nur 500 mA ausgelegt, was in diesem Fall zu Spannungseinbrüchen führen kann. In diesem Fall z.B. den WiFi-Dongle entfernen oder zusätzlich die Stromversorgung über die Powerbank anschließen und einschalten.

Ein- und Ausschalten des BB

Vor dem Einschalten prüfen, ob die SD-Karte im Halter eingerastet ist. Ansonsten bootet der BB vom Internen Speicher mit dem falschen Betriebssystemimage.

Einschalten durch Einstecken der Spannungsversorgung über den Micro-USB-Anschluss bzw. über die Powerbank.

Zum Ausschalten keinesfalls die USB-Spannungsversorgung trennen sondern zuerst das Betriebssystem des BB herunterfahren: Entweder durch Drücken der Power-Taste oder mit dem Befehl sudo `shutdown -h now` im Terminalfenster.

Erst wenn alle blauen LED erloschen sind, USB-Spannungsversorgung trennen.

Oder erneutes Einschalten durch Drücken der Power-Taste, falls USB-Spannungsversorgung nicht zwischenzeitlich getrennt wurde.

1 Einleitung

Sensoren werden meistens über **Mikrocontroller (µC)** ausgelesen oder parametriert.
Im Automobilbereich werden beispielsweise die Raddrehzahlsensoren vom µC des ABS-Steuergeräts ausgelesen, welches die Ventile der einzelnen Bremsen steuert. Im Bildungs- und Hobbybereich verwendet man oft die Arduino-Plattform mit einem ATmega328 µC, der z.B. über den I²C-Bus mit Sensoren kommuniziert. Ein µC wird überwiegend nur für eng begrenzte Aufgaben eingesetzt und besitzt eine limitierte Rechenleistung sowie einen limitieren Speicherplatz. Sein großer Vorteil ist neben den geringen Kosten sein zeitlich deterministisches Verhalten („Echtzeitverhalten“), weshalb er sich für zeit- und sicherheitskritische Anwendungen eignet.

Für komplexe Regelungsaufgaben wie z.B. das autonome Fahren reicht die Rechenleistung eines einfachen µC bei weitem nicht mehr aus. Hier verwendet man komplexe eingebettete Systeme, die aus einem **Mikrocomputer** inkl. Betriebssystem (meistens Linux) bestehen. Einen solchen Rechner nennt man „**Linux Embedded System (LES)**“.

Auch ein PC oder ein Laptop wird als Mikrocomputer bezeichnet. Der Rechner ist hier aber körperlich auf verschiedene Komponenten aufgeteilt, wie der Prozessor, der Grafikchip, der RAM-Speicher und etwa die Festplatte. Außerdem wird ein PC anders als ein µC oder LES universell und nicht nur für *eine* eng eingegrenzte Anwendung verwendet.

Bei den LES befindet sich ein Großteil der Rechnerhardware auf einem hochintegriertem Chip. Daher spricht man hier auch von einem „System on a Chip (SoC)“.
Prominente Vertreter solcher LES sind der Raspberry Pi (RasPi), der Arduino Yun sowie der BeagleBone (BB)[1]. Sie nennt man auch „Einplatinencomputer“.

Diese Systeme beinhalten wie PC einen leistungsstarken Prozessor mit mehreren Hundert MHz Taktfrequenz, einen etwa GByte großen Arbeitsspeicher und als Festplattenersatz einen ebenso großen Flashspeicher (z.B. eMMC). Wie bei einem PC-Mainboard gibt es auch Schnittstellen zur Peripherie (Tastatur, Bildschirm, …) wie HDMI, USB, Ethernet oder WLAN.

Anders als µC - die eigentlich „wahren“ SoC[2] - sind LESs mit einem Betriebssystem ausgestattet. Dabei handelt es sich im Gegensatz zu PCs meistens um ein Linux-Betriebssystem ohne grafische Benutzeroberfläche.

Ein für das Sensor- und Regelungssystemeprojekt (PAT) wesentlicher Vorteil von LES gegenüber einem PC sind dessen sogenannte I/O-Peripherie („Inputs / Outputs“). Diese Ein-/Ausgabepins sind beim RasPi und BB ähnlich wie beim Arduino auf Buchsenleisten hinausgeführt. Erst dadurch ist das Anschließen von Sensoren und Aktoren einfach möglich.

Eigentlich müsste man LES besser „*Linux auf einem eingebetteten System*“ nennen: Denn das eingebettete System profitiert von allen allgemeinen Linux-Vorteilen, wie von dessen Effizienz, von der riesigen Anzahl hochwertiger Open Source Software, vom exzellenten Open Source Support, von den fehlenden Lizenzkosten und nicht zuletzt von der hohen Stabilität dieses Betriebssystems, das sowohl für Konsumprodukte wie Digitalkameras als auch auf großen Servern Verwendung findet.

Da Linux auf vielen verschiedenen SOC verwendet wird, können fertige Applikationen ohne größeren Aufwand auch auf andere Plattformen portiert werden, wenn z.B. die Leistungsfähigkeit des ursprünglichen SOC nicht mehr ausreicht.

Neben den Kosten ist der Haupt**nachteil** von LES die fehlende (harte) Echtzeitfähigkeit:
µC haben den Vorteil, dass deren I/O-Peripherie zeitlich deterministisch (in „Echtzeit“) bedient wird. Bei einem Mikrocomputer ist das aufgrund dessen Betriebssystems normalerweise nicht der Fall. Beispielsweise muss hier der Prozess zur Abfrage eines Sensorausgangs auf einen anderen vorher gestarteten Prozess warten.

Den BB zeichnet gegenüber dem RasPi aus, dass er zusätzlich zu seinem Prozessor zwei „Programmable Real-Time Units (PRU)“ sozusagen als 32 Bit Koprozessoren mit 200 MHz Taktrate besitzt, die für die I/O-Peripherie zuständig sind:
Die PRU des BB sind auf dem SOC Chip zusätzlich integrierte µC, die unabhängig von der Prozessoraus-

1 Anfänglich wurde dieser Mikrocomputer als „BeagleBoard“ bezeichnet. Daher tauchen jetzt in der Literatur die Begriffe „BeagleBone“ und „BeagleBoard“ mehr oder weniger gleichbedeutend auf.
2 Bei µC sind die Rechnerkomponenten in der Regel komplett in einem Chip integriert. Auf den LES ist z.B. der Flashspeicher meistens auf einem getrennten Chip aber auf der selben Platine untergebracht - daher auch die Beizeichnung „Einplatinencomputer“.

lastung die I/O-Peripherie bedienen. Damit ergibt sich auf der Ebene der I/O-Peripherie eine Echtzeitfähigkeit, was im PAT jedoch (noch) nicht genutzt wird.

Im Vergleich zum RasPi hat der BB wesentlich mehr und besser dokumentierte Schnittstellen zur I/O-Peripherie für das Anbinden von Sensoren und Aktoren. Insgesamt ist der BB auch die professionellste Plattform unter den Maker-LES.
Trotzdem setzt sich auch in Mechatronikprojekten immer mehr der RasPi durch. Aufgrund dessen höheren Verbreitungsgrades bietet die Open Source Community hier deutlich mehr Software und Support als für den BB.

Ein wesentliches Lernziel des PAT ist das Erlernen des „Mechatronik Rapid Prototyping“ sowie das „Modularisieren". Hierfür ist der BB mit seiner umfangreichen I/O-Peripherie die ideale Hardwareplattform. Dazu muss aber auch eine „Rapid“ Softwareentwicklung hinzukommen, die für ein Modularisieren geeignet ist.

> *Im Praktikum wird die Middleware ROS1 (ROS-Version „Neotic“, nachfolgend einfach ROS genannt) verwendet, die das mechatronische System in getrennt programmierbare und testbare Komponenten (Module) - sogenannte ROS-Knoten - zerlegt. Es wird vorzugsweise die Programmiersprache Python 3 verwendet. Die Regelung sowie der Kalman-Filter wird über MATLAB/Simulink modellbasiert entwickelt. Dieses Simulinkmodell kommuniziert entweder zur Laufzeit als ROS-Knoten direkt mit den anderen Komponenten oder es wird daraus automatisch C-Code und damit ein ROS-Knoten erzeugt, welcher auf dem BB als Regelungskomponente unter ROS ausgeführt wird.*

Reicht die Performanz der Python-Skripte später nicht aus, so können die ROS-Quellcodes in C übertragen werden, was „zusätzliche Echtzeitfähigkeit“ ergibt.

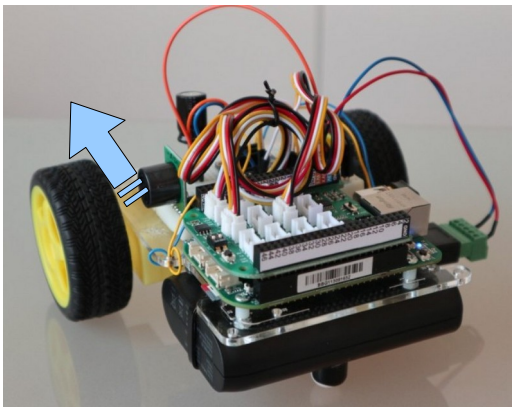
Bis auf Codebestandteile, die die Schnittstellen des BB betreffen, lassen sich die erstellten Softwarekomponenten unverändert auf einen RasPi übertragen. Denn beide LES besitzen das gleiche Betriebssystem, den gleichen Python-Interpreter, einen funktionsgleichen C-Compiler sowie die gleiche ROS-Middleware. Für den RasPI sind dann aber zusätzliche Hardwaremodule z.B. für analoge Schnittstellen nötig.

2 Die Roboterhardware

2.1 RoboToGo

Als Roboterplattform wird das „RoboToGo“-Fahrzeug verwendet. Jedes der beiden Räder wird getrennt von einem Gleichstromantrieb inkl. Getriebe bewegt. Die Räder können bei Bedarf mit Rotationsencodern ausgestattet werden.
Der BB steuert mittels PWM-Signale über einen H-Brücken-Motortreiber die Motoren an. Die Energieversorgung geschieht über eine USB-Powerbank.
Die Kugelrolle (ball caster) unter der USB-Powerbank bildet den dritten Auflagepunkt.

Die Fahrtrichtung des RoboToGo wird am besten so gewählt, dass die Kugelrolle nachläuft. Damit lassen sich Bodenhindernisse wie z.B. Fliesenfugen leichter überwinden.



Die Ansteuerung der Motortreiber (H-Brücke) ist sehr gut im Buch von D. Molloy [1], Kapitel 9 erklärt. Die Platine kann übrigens mit einem Shuntwiderstand bestückt werden, mit den der aktuelle Motorstrom über einen ADC-Eingang des BB erfasst wird.

Mit dem RoboToGo wird das PAT für Studierende „mitnehmbar nach Zuhause".

Achtung: Bei Lastspitzen durch den Motortreiber kann die Versorgungsspannung des BB so stark einbrechen, das dieser einen Reset durchführt.

3 Die BeagleBone Grundlagen

Auf dem BB selbst und im Internet finden sich unzählige Informationen, wobei die erste Anlaufstelle immer die Internetseiten der BeagleBoard.org Foundation sein sollte [2]. Als Fachliteratur wird an erster Stelle auf [1], aber auch auf [3] und [4] verwiesen. Im Internet findet sich unter [5] vom Elektronikhersteller Adafruit umfangreiche Lernmaterialien und Python-Bibliotheken.

Die meisten Internetseiten und Bücher beziehen sich im Schwerpunkt auf die Programmierung des BB über JavaScript oder Python. Nur [1] bzw. [6] behandeln ausreichend die Programmiersprache C und das Einbinden von I/O-Peripherie.

Alle bis hierhin genannte Lehrbücher sind als Printbücher in der Hochschulbibliothek ausleihbar.

Im PAT wird der „BeagleBone Green (BBG)“ verwendet. Der BBG unterscheidet sich vom bekannteren „BeagleBone Black (BBB)“ nur darin, dass er aus Kostengründen keine HDMI-Schnittstelle aber dafür je einen „Grove“-Stecker für UART und I²C Peripherie hat, siehe [7].

Vom BBB und gibt es mehrere Varianten („Revisions“), welche sich z.B. in der Flashspeichergröße unterschieden. Für das PAT sind die Revisions der BBB jedoch unerheblich.

Ein BBG kostet ca. 50 € bei Bezugsquellen wie z.B. [8] oder [9].

3.1 Die BB-Hardware

Das Herzstück des BB ist der Texas Instruments Sitara 1 GHz ARM-A8 Cortex Prozessor. Für ihn wird recht konsistent z.B. bei Dateibezeichnungen die Abkürzung „AM335x“ verwendet.
Der BBG besitzt u.a. folgende Datenschnittstellen:

65 binäre I/Os (GPIO), 8 DAC als PWM Ausgänge, 7 ADC, 2 I²C, 2 SPI, 2 UART, 2 CAN, Micro-SD-Karte, USB Host (USB-A), USB Client (Micro-USB), Ethernet (RJ45).

Als Benutzerinterface besitzt er nur 4 LEDs. Von Außen nach Innen („USR0“ bis „USR3“) sind dies Indikatoren für: „Herzschlag “ des Linux-Kernels[3], Zugriff SD-Karte, Prozessoraktivität, Zugriff eMMC-Speicher[4].
Weiter befindet sich noch ein Power- und einen Reset-Taster auf der Platine.

Im Anhang Kapitel 12.4 findet sich eine detaillierte Übersicht zur BB-Hardware.

3.2 Grove Cape für den BB

Ähnlich wie die sogenannten „Shields“ der Arduino-Plattform oder „Hats“ der Raspberry Pi Plattform gibt es für den BeagleBone „Capes“. Die Capes vereinfachen nicht nur das Anschließen der I/O-Peripherie sondern schützen auch den BB vor Schäden durch falsches Anschließen. Leider ist der BB gegenüber falsches Anschließen wesentlich empfindlicher als ein Arduino Uno:

Achtung:
Werden an den digitalen Schnittstellen Spannungen größer 3,3 V oder an der Analogschnittstelle größer 1,8 V angelegt, dann führt dies wahrscheinlich zur Zerstörung des BB.
Verwenden Sie daher als Eingänge ausschließlich die Grove-Stecker!

Im PAT wird das „Grove Cape (Version 2)“ des Herstellers Seeed [9] verwendet. Zusammen mit den speziell dafür konfektionierten „Grove“-Bauteilen von Seeed ist ein falsches Anschließen nahezu ausgeschlossen.

Auf dem Grove Cape sind die UART-1/2-, I²C-2 -Kommunikation sowie für die ADC AIN 0 bis 4 auf spezielle Stecker herausgeführt. Diese Stecker haben zusätzlich noch einen GND und VCC-Anschluss und sind kompatibel mit der entsprechend vorkonfektionierten Grove-Peripherie. Es können über Adapterkabel aber beliebige Sensoren angeschlossen werden. Über den Jumperstecker neben der Ethernetschnittstelle kann (für das gesamte Cape!) die Versorgungsspannung VCC auf 3,3 V oder 5 V eingestellt werden.

Für die analogen Eingänge „Analog Input“ ist ein Verstärker nachgeschaltet, der die Eingangsspannung über einen Spannungsteiler von 5 V auf 1,8 V reduziert, damit selbst bei einer 5 V Eingangsspannung der ADC nicht zerstört wird. Das heißt aber auch, dass ein 3,3 V Signal auf den ADC diesen nicht voll aussteuert (ca. 2700 LSB statt FSR = 4096 LSB des eigentlichen ADC mit 12 Bit Auflösung).

Im Anhang im Abschnitt 12.1 findet sich ein Schaltplan und das Layout des Grove Capes.
Der SOC des BB selbst hat viel mehr I/O-Möglichkeiten als Pins auf dem beiden Buchsenleisten hat.

3.3 Programmierung des BeagleBone

Das eingebettete System nennt man im Fachjargon auch „Target“, weil diese Hardware das Ziel des in einer Entwicklungsumgebung (IDE) erstellten Programmcodes ist, wenn man diesen am Ende überträgt bzw.

3 Diese LED blinkt in einem Herzschlagrhythmus, und zeigt damit, dass das Linux-Betriebssystem störungsfrei läuft.
4 Im Image des PAT wird die USR3-LED dazu verwendet, nach dem Booten eine bestehende WLAN-Verbindung anzuzeigen.

kompiliert.

Auf unterster Ebene gibt zwei verschiedene Arten, wie ein Programm auf dem BB ablaufen kann:

3.3.1 Skriptprogramme:

Es gibt Skriptprogramme wie Python oder JavaScript (beim BB speziell das „BoneScript“), die nicht direkt auf dem Prozessor sondern auf einer virtuellen Maschine bzw. einem Interpreter ausgeführt werden. Dabei ist die virtuelle Maschine bzw. der Interpreter ein Programm, das direkt auf dem Prozessor läuft. Der Vorteil dieser beiden Skriptprogramme ist deren konsequente Objektorientiertheit und die einfache Portierbarkeit auf eine andere Plattform wie z.B. auf einen RasPi.

BoneScript hat erstens den Vorteil, dass diese Programme direkt über einen Browser ausgeführt werden können, wie in [2] in mehreren Beispielen gezeigt wird. Zweitens ermöglicht diese Skriptsprache einen asynchronen Programmfluss, was im Kontext der Sensoranbindung bedeutet, dass der Programmablauf direkt von einem Sensorsignal beeinflusst werden kann.

Die Skriptsprachen haben aber den Nachteil, dass deren Ausführungsgeschwindigkeit sehr langsam ist. Für ein mechatronisches System wie z.B. die ABS-Regelung beim Auto wären die damit verbundenen Latenzzeiten inakzeptabel.

Im PAT wird trotzdem Python verwendet, da die Echtzeitanforderungen sehr gering sind. Auch für das Testen der Kommunikation mit der I/O-Peripherie ist Python sehr gut geeignet.

Eine kurze und gut verständliche Einführung in Python findet sich z.B. in [10].

Sehr praktisch ist bei Python die interaktive Python-Konsole als Schnittstelle zum Interpreter: Ähnlich wie bei MATLAB kann man in der Kommandozeile dieser „Python Shell“ einzelne Programmbefehle nacheinander eingeben und so testen, bevor man den Quellcode in eine Datei schreibt und dann als Ganzes ausführt. In der Python-Shell kann beispielsweise interaktiv die Kommunikation über eine I²C-Schnittstelle "ausprobiert" werden.

Auf dem BB wird der Python 3 Interpreter mit dem Befehl `python3` aufgerufen.

```
beagle@beaglebone:~$ python3
>>> import Adafruit_BBIO.PWM as PWM
>>> PWM.start("P9_14",12,50,0)
>>> PWM.set_duty_cycle("P9_14",3)
```

Im Beispiel oben wird die Python Shell mit dem Befehl `python3` aufgerufen. Nach dem Import der PWM-Bibliothek wird ein PWM-Signal mit 12 % Duty Cycle und 50 Hz initialisiert, welches in der folgenden Zeile auf 3 % Duty Cycle reduziert wird. Die Python Shell verlässt man mit `Strg + q`. Hiermit wird z.B. ein Modellbauservo oder ein DC-Motor angesteuert.

Achtung: Es gibt leider zwei verschiedene Pythonversionen 2.7 und 3.x, die sozusagen in zwei Paralleluniversen existieren. **ROS Neotic verwendet auf dem PC wie auch auf dem BB ausschließlich Python 3.x.**

3.3.2 Kompilierte Programme

ROS ermöglicht auch die Knoten in C zu coden.

Die Latenzzeiten werden wesentlich kleiner, wenn der Programmcode in der Programmiersprache C erstellt wird. Denn dabei wird eine speziell für den Prozessor ausführbare sogenannte „native“ Programmdatei „kompiliert“ = erzeugt. Über die C-Programmierung kann auch die PRU direkt angesteuert werden, was die Ausführungsgeschwindigkeit nochmals erhöht. Hier sind aber sehr weitgehende Programmierkenntnisse erforderlich.

Ganz so „Schwarz-Weiß“ ist die Abgrenzung der Skriptprogramme zu den C-Programmen jedoch nicht: In der Realität werden Skriptprogramme immer mit Bibliotheken kombiniert, die kompilierten Programmcode enthalten. D.h. der Interpreter ruft Unterprogramme auf, die als nativer Code vorliegen und wahrscheinlich in C programmiert wurden. Nachteilig ist hierbei jedoch, dass dieser C-Code praktisch nicht debugged werden kann, wenn Laufzeitfehler auftreten.

Python-Skripte lassen sich daher nur sehr mühsam debuggen.

4 Das Robot Operating System (ROS)

4.1 Was ist ROS und wofür ist ROS gut?

Ganz einfach gesagt: ROS ist eine sogenannte „Middleware“, die zwischen ausführbaren Programmen und dem Betriebssystem arbeitet. ROS wird dafür verwendet, um ein mechatronisches System auch in in Bezug auf Software in seine einzelnen Komponenten zu ***modularisieren***.

Konkret ist in ROS jede Komponente wie z.B. ein Sensor, ein Aktor oder eine Regelung einem Prozess (genannt „Knoten“ in Form eines C- oder Python-Quellcodes) zugeordnet. Bekanntlich ist bei einem mechatronischen System das Zusammenspiel der einzelnen Komponenten das A und O: Genau hier setzt ROS an, in dem es die Kommunikation der einzelnen Knoten organisiert.

Eine sehr gute Erklärung zum Sinn und Zweck von ROS findet sich in [11]. Auch in der Wiki der offiziellen ROS Internetseiten [12] finden sich exzellente Grundlageninformationen.

Im PAT wird die ROS-Version „Neotic“ verwendet. Genau genommen handelt es sich hierbei um eine Version von "ROS1". Aktuelle ROS-Versionen basieren schon auf der komplett überarbeiteten Version „ROS2“. Ähnlich wie Python 2 vor einigen Jahren ist derzeit ROS1 noch weitaus mehr verbreitet und läuft wesentlich stabiler als das neue ROS2. Es ist aber damit zu rechnen, dass Mitte der 20er Jahre die ROS-Gemeinschaft auf ROS2 umsteigen wird.

4.2 Verwendung von ROS auf dem PC mit Ubuntu 20.04 LTS Betriebssystem

Dieses Kapitel ist nur relevant, wenn Sie im PAT mit Ihrem eigenen PC arbeiten möchten. Die PCs im Sensorlabor sowie die SSDToGo sind komplett fertig konfektioniert für die Arbeit mit ROS, Simulink und BeagleBone. Hier müssen Sie keinerlei weiteren Installationen oder größere Einstellungen vornehmen.

Als Betriebssystem für eine ROS-Installation wird Ubuntu 20.04 LTS empfohlen, Hierfür existieren bewährte Paketquellen für ROS, die die Installation sehr einfach machen. Damit läuft sowohl auf dem PC wie auf dem BB ein Linux-Betriebssystem, was die Arbeit wesentlich erleichtert.

4.2.1 Installation von ROS auf den eigenen PC

Auf den offiziellen ROS Internetseiten [13] sind für verschiedene Linux-Distributionen als auch für Windows und MacOS die Installation sehr gut beschrieben. Von einer Installation auf einem anderen Betriebssystem als Ubuntu 20.04 LTS wird abgeraten, zumal auch dann wesentlich mehr Probleme mit Firewalls aufkommen, die die ROS-Kommunikation behindern. **Installieren Sie für das PAT ausschließlich die ROS1-Version Noetic.**

4.2.2 Installation weiterer nützlicher Tools auf den PC (Ubuntu 20.04)

Bei ROS spielt die Kommunikation zwischen Rechnern eine wichtige Rolle. Mit dem Tool *`Net Tools`*, (auf dem BB schon vorhanden) können mit Befehlen wie *`ifconfig`* die unterschiedlichen PC-Schnittstellen untersucht werden. Installation auf dem PC mit

```
sudo apt-get install net-tools
```

Da im PAT mit ROS jedes Modul des mechatronischen Systems über einen getrennten Prozess abgebildet wird, benötigt man auf dem PC oft vier oder mehr Terminalfenster. Dafür ist das Programm „Terminator“ optimal: Ein Terminal-Multiplexer, der das Verwenden mehrere Konsolen (Terminals) in einem einzigen Fenster ermöglicht, siehe [14]. Die Installation erfolgt mit

```
sudo apt-get install terminator
```

Anschließend kann *`terminator`* mit der Tastenkombination *`strg + alt + t`* gestartet werden.

4.3 Verwendung von ROS auf dem BeagleBone mit Ubuntu 20.04 LTS Betriebssystem

Für die BB im PAT wird wie auf dem SSD-ToGo-PC ein Ubuntu-20.04-Betriebssystem verwendet. Das Ubuntu auf dem BB ist aber sehr abgespeckt und besitzt keine Benutzeroberfläche.

Auf dem PC wie auf dem BB ist ROS Noetic installiert. Auf dem BB befinden sich im Projekt *pat_intro* die in Abschnitt 10.2 vorgestellten Beispiel-ROS-Knoten, die ohne weitere Installationen ausführbar sind.

4.3.1 Das Betriebssystemimage und die SD-Karte

Im PAT wird ausschließlich die SD-Karte als Speichermedium im BB verwendet. Ähnlich wie auf der Festplatte eines PC befindet sich darauf das Betriebssystem, alle installierten Treiber und Programme sowie persönliche Dateien.
Anders als bei einem PC wird das Betriebssystem auf dem BB nicht installiert, sondern es wird ein sogenanntes „Image“ des Betriebssystems auf die SD-Karte des BB bitweise kopiert. Als Image bezeichnet man ein Abbild (1:1 Kopie) von einer oder mehreren Partitionen eines Speichermediums, welches üblicherweise anschließend noch komprimiert wird. Im Gegensatz zu PCs, die unterschiedliche Prozessoren, Grafikchips usw. haben, ist ein solches Klonen der „Festplatte“ bei einem festen BB-Modell (hier BBG) möglich. Denn hier ist die Hardware immer identisch. Umgekehrt kann man den Ist-Zustand eines BB als Image (inkl. der zwischenzeitlich zugefügten Programme und Dateien) konservieren und so eine komplette Back Up Datei erstellen.

Mit dem Begriff „Image“ ist also eine 1:1 Kopie des Speichermediums des BB gemeint. Das aktuelle BB-Image für das PAT finden Sie als Download-Link unter Relax.

4.3.2 Kopieren, Sichern, Wiederherstellen und Klonen von SD-Karten BB-Images

Back Up eines SD-Karten Images auf einen (Linux-) PC:

Zuerst wird ohne SD-Karte im PC mit dem Befehl *lsblk* eine Liste der angeschlossenen Blockgeräte ausgegeben. Diesen Befehl wiederholt man anschließend mit eingesteckter SD-Karte. Im Vergleich beider Ausgaben und auch in Bezug auf die Speicherkartengröße lässt sich damit der Pfad der SD-Karte bestimmen, in diesem Beispiel *dev/mmcblk0*.

Ohne SD-Karte:

```
mackst@tec-04-206-02:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
...
sdb           8:16   0 232,9G  0 disk
├─sdb1        8:17   0   500M  0 part
├─sdb2        8:18   0 231,5G  0 part
└─sdb3        8:19   0   907M  0 part
```

Mit SD-Karte:

```
mackst@tec-04-206-02:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb           8:16   0 232,9G  0 disk
├─sdb1        8:17   0   500M  0 part
├─sdb2        8:18   0 231,5G  0 part
└─sdb3        8:19   0   907M  0 part
mmcblk0     179:0    0  14,9G  0 disk
└─mmcblk0p1 179:1    0   5,3G  0 part /media/mackst/rootfs
```

Der Befehl *sudo dd if=/dev/mmcblk0 of=~/SDCardBackup.img status=progress* überträgt die komplette SD-Karte als Imagedatei *SDCardBackup.img* auf den PC in das Home-Verzeichnis und zeigt während dieser recht langen Zeit den Kopierstatus an.

```
mackst@tec-04-206-02:~$ sudo dd if=/dev/mmcblk0 of=~/SDCardBackup.img status=progress
31116288+0 Datensätze ein
31116288+0 Datensätze aus
15931539456 Bytes (16 GB, 15 GiB) kopiert, 784,569 s, 20,3 MB/s
```

Kopieren eines vorhandenen Images auf eine SD-Karte (unter Linux):
Zuerst wird zur Sicherheit wieder mit dem Befehl *lsblk* der Pfad der SD-Karte ermittelt in diesem Beispiel *dev/mmcblk0*.

Nun kann man im Terminalfenster mit dem Befehl *sudo dd if=~/SDCardBackup.img of=/dev/mmcblk0 status=progress* mit rudimentärer Fortschrittsanzeige die Imagedatei *SDBackup.img* auf die SD-Karte kopieren.

Einfacher und mit grafischer Fortschrittsanzeige funktioniert es mit dem Ubuntu Datei-Explorer: Hier wählt man mit der rechten Maustaste auf der Imagedatei die Option „Mit anderer Anwendung öffnen“, dann

„Schreiber von Laufwerkabbildern“ und wählt in dem erscheinenden Fenster als Ziel die SD-Karte, also das Blockgerät *dev/mmcblk0* aus.

4.3.3 SD-Karte als Massenspeicher versus eMMC

Das PAT verwendet das Linux-Betriebssystem als „Image“ auf einer SD-Karte.
Der eMMC-Speicher wird nicht benötigt, da von der SD-Karte aus gebootet und dort auch die Software abgespeichert wird.

Der BB besitzt sozusagen als SSD-Festplatte einen eMMC (Embedded Multi Media Card). Dieser Speicher ist recht schnell und hat eine Kapazität von 4 GB.

Wenn man auf diesem Speicher jedoch das Betriebssystem „zerschossen“ hat, dann ist es schwieriger auf dem eMMC befindliche Dateien nachträglich zu sichern. Stehen schnelle SD-Karten zur Verfügung, so ist es vorteilhaft diese statt des eMMC als Festplatte zu verwenden. In diesem Fall befinden sich alle Dateien inkl. Dateisystem auf der SD-Karte statt auf dem eMMC. Dieses sogenannte „Betriebssystem-Image“ lässt sich dann einfach auf dem PC mit einem SD-Kartenleser sichern oder auf eine andere SD-Karte klonen.

Wenn man das Betriebssystem auf der SD-Karte zerschossen hat, dann kann man dort befindliche eigene Dateien anders als beim eMMC dadurch retten, indem man die SD-Karte vom BB entfernt und mit einem PC ausliest.
Wenn keine SD-Karte mit funktionierendem Linux-Image eingesteckt ist, dann bootet der BB vom eMMC aus. Andernfalls bootet er (je nach Image auf dem eMMC) manchmal auch vom eMMC aus, falls beim Booten nicht die USER (USR)-Taste gedrückt ist.

Achtung:
Bootet der BB standardmäßig beim Einschalten der Stromversorgung, also z.B. beim Einstecken des USB-Kabels, vom eMMC aus, so muss beim Einschalten die USER (USR)-Taste auf dem Grove-Cape gedrückt sein, damit der BB von der SD-Karte aus bootet.

Tipp:
Wichtig ist, dass man nach dem Booten weiß, ob man nun auf der SD-Karte oder auf dem eMMC arbeitet. Am einfachsten ist es, wenn man einen individuellen Hostnamen verwendet: Dieser erscheint nämlich nur im Promt, wenn von SD-Karte aus gebootet wurde. Ansonsten erscheint der Standard-Hostname *beaglebone*.

Alternativ kann man auch über das Flackern der USR LEDs während des Bootens herausfinden:

USR3 (erste LED neben der Ethernetbuchse): Booten von eMMC.

USR1 (die übernächste LED neben der „Herzschlag LED“): Booten von der SD-Karte.

Nach dem Booten zeigt die USR3-LED jedoch eine aktive WLAN-Verbindung (falls vorhanden) an.

5 Kommunikation zwischen PC und BeagleBone

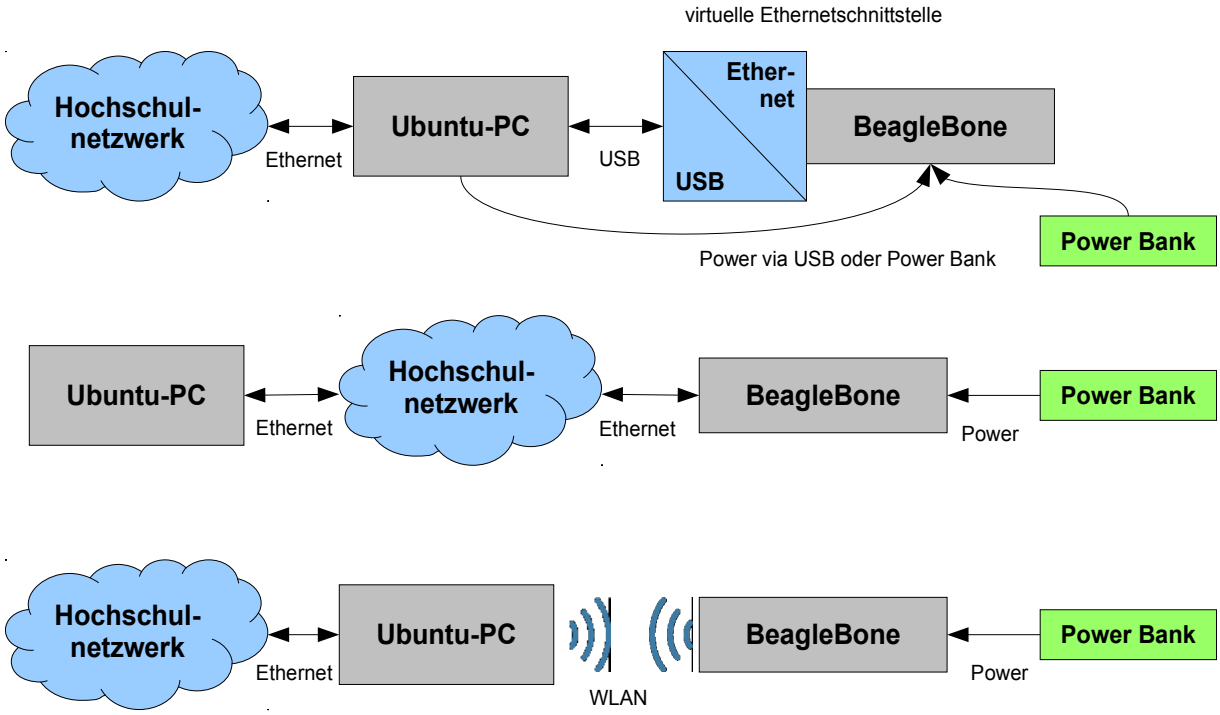


Abbildung 1: Übersicht der im Praktikum verwendeten Kommunikationsarten und Stromversorgungen.

Nachfolgend wird gezeigt wie über ein Terminalfenster auf dem PC eine Verbindung mit dem BB aufgebaut wird. Damit wird der BB durch den PC über Textkommandos in der sogenannten „Kommandozeile[5]“ (Shell oder Bash genannt) sozusagen ferngesteuert: Der PC ersetzt hierfür die Tastatur und den Bildschirm, der sonst am BB angeschlossen sein müsste. Denn auf dem im PAT verwendeten Image des BB gibt es keine Benutzeroberfläche.

Zuerst muss eine Kommunikation zwischen dem BB und dem PC physikalisch aufgebaut werden. Dafür gibt es unterschiedliche Wege (siehe Abb. 1).

Grundsätzlich kann der BB über vier verschiedene Schnittstellen mit dem PC verbunden werden:

1. Serielle Schnittstelle (wird im PAT nicht verwendet)
2. USB-Schnittstelle (emuliert eine Ethernetschnittstelle)
3. Ethernetschnittstelle (entweder im selben Netzwerk wie der PC oder lokal über ein „Coss Over“-Kabel)
4. WLAN-Schnittstelle

Die PC-Kommunikation via serieller Schnittstelle wird im PAT nicht verwendet. (Die serielle UART-Schnittstelle wird im PAT nur intern am BB für die I/O-Kommunikation mit Sensoren bzw. Aktoren verwendet.) Sie ist die einzige Möglichkeit den Bootvorgang des BB zu kontrollieren, da die übrigen Schnittstellen dann noch inaktiv sind.

Bei der ersten Inbetriebnahme verbindet man den BB am besten via USB mit dem PC. Damit ist auch gleichzeitig die Spannungsversorgung gewährleistet, und kein separates Netzteil oder eine Powerbank wird benötigt. Bei erstmaligen Anschließen eines BB an einen PC wird der BB zuerst nur als Massenspeicher ähnlich einem USB-Stick erkannt.

Parallel zu USB kann der BB sich auch via Ethernetschnittstelle z.B. in das Hochschulnetzwerk einbuchen. Dann kann die USB-Verbindung auf die Stromversorgung reduziert und anschließend durch eine Powerbank an der 5 V Buchse bzw. VDD_5V ersetzt werden.

5 Die Kommandozeile nennt man auch „Shell“. Das Programm auf dem BB, welches die Kommandos verarbeitet, wird „Bash“ genannt. Lassen Sie sich nicht irritieren, wenn diese beiden Begriffe oft gleichbedeutend verwendet werden. Das hierzu gehörende Protokoll heißt „SSH“.

Bei einer mobilen Stromversorgung bietet sich eine WLAN-Kommunikation an. Jedoch ist diese im PAT mit vielen WLAN-Teilnehmern erheblich langsamer und instabiler als eine USB-Verbindung.

Tipp:
Für die Einarbeitung am BB ist ist die USB-Verbindung die beste Option. Eine WLAN-Kommunikation sollte nur verwendet werden, wenn eine drahtgebundene Kommunikation nicht möglich ist.
Für die Stromversorgung (z.B. Powerbank) wird dann über den Pin P9_5 oder P9_6 (VDD_5V) in den Power Management Chip des BB eingespeist.

5.1 Kommunikation via USB

Die Kommunikation geschieht hier über eine durch die USB-Schnittstelle emulierte Ethernetschnittstelle.

Achtung:
Häufig findet man Mikro-USB-Kabel, wie sie beispielsweise Powerbanks beiliegen, bei denen nur die Stromversorgung jedoch nicht die beiden Kommunikationsadern angeschlossen sind. Mit solchen Kabeln kann selbstverständlich keine Kommunikation mit dem BB hergestellt werden.

Nach dem Verbinden des USB-Kabels bootet der BB, und nach einigen zehn Sekunden erscheint auf dem PC zuerst das Verzeichnis BEAGLEBONE als Massenspeicher und dann die Meldung über zwei neue Ethernetverbindungen *eth0* und *eth1*. Zu einer der beiden Ethernetverbindungen (Netzwerk *192.168.7.x*) kann der PC jedoch manchmal keine Verbindung aufbauen. Die Fehlermeldungen „Verbindung gescheitert“ dazu können ignoriert werden.

Nun kann über die IP-Adresse *192.168.6.2* eine Verbindung vom PC zum BB aufgebaut werden. Der Status und die Qualität der Verbindung kann vorab in einem Terminalfenster (dafür *strg + alt + t* auf dem PC eingeben) mit dem Befehl *ping 192.168.6.2* getestet werden (Abbruch mit *strg + c*).

Gibt man *192.168.6.2/bone101* in die Adresszeile des Browsers ein, dann wird die *beagleboard.org*-Seite angezeigt. Dabei handelt es sich nicht um eine Seite aus dem Internet, sondern die Seite wird aus lokalen Verzeichnissen des BB aufgerufen. Auf diesen „Internetseiten“ steht alles Weitere, um sich mit dem BB vertraut zu machen.

Über die Adresse *192.168.6.2* kann man sich jetzt auch via SSH-Protokoll[6] auf dem BB in sein Linux-Betriebssystem einloggen (Benutzername *beagle*, Passwort *temppwd*). Dafür öffnet man ein Terminalfenster und gibt den Befehl *ssh beagle@192.168.6.2* ein.

```
mackst@tec-04-206-02:~$ ssh beagle@192.168.6.2
beagle@beaglebone:~$
```

(Beim ersten Mal muss vor dem Passwort noch eine Sicherheitsabfrage mit *yes* beantwortet werden. Wenn vorher ein anderer BB verbunden war, muss zuerst der alte Schlüssel gelöscht werden - siehe Hinweis in der Fehlermeldung nach dem *ssh*-Befehl.)

Das Feedback im Terminalfenster an sich ist Dienstprogramm des BB: Ein solches Programm nennt man „Shell“. Im speziellen Fall hier wird die Shell mit dem Namen „Bash“ verwendet.

Genauso gut kann man aufeinanderfolgende Befehlszeilen aber auch in einer Textdatei abspeichern. Dann kann diese Textdatei als "Shell"- bzw. "Bash-Skript" ausgeführt werden: Hier werden die einzelnen Befehle Zeile für Zeile automatisch abgearbeitet. Solche Skriptbefehle sind auch in der (durch den "." versteckten) Datei *~/.bashrc* enthalten. Diese werden automatisch beim Öffnen eines neuen Terminalfensters ausgeführt.

Achtung:
Zum Ausschalten darf der BB nicht einfach nur von der USB-Schnittstelle bzw. Powerbank (=Spannungsversorgung) getrennt werden, so wie man es bei einem Arduino macht. Da beim BB ein Betriebssystem mit im Spiel ist, muss dieses wie bei einem PC vor dem Trennen heruntergefahren werden. Dies geschieht durch kurzes Drücken der „POWER“ Taste am BB (Taster direkt neben dem Ethernetanschluss).

Falls das Ausschalten über die Power-Taste nicht funktioniert, kann im Terminalfenster der Befehl *sudo shutdown -h now* verwendet werden. Durch langes Drücken (>8 s) der POWER-Taste wird der BB „hart“ heruntergefahren.
Mit dem Befehl *reboot* kann der BB neu gestartet werden. Das selbe bewirkt die RESET-Taste.

6 Dies nennt man auch „SSH Shell“: Man kommuniziert innerhalb eines PC-Terminalfensters mit dem Linux-Betriebssystem des BB über Tastaturbefehle mittels SSH-Protokoll.

Ist der BB „nur“ über USB angeschlossen, dann besitzt er standardmäßig keinen Internetzugang. Dafür kann man ihn z.B. zusätzlich (oder ausschließlich, dann aber mit Powerbank) via Ethernetkabel an einen Switch oder Router anschließen.

Tipp:
Die manchmal recht langen Linuxbefehle aus Webseiten oder aus dieser Anleitung kann man über Copy & Paste in das Terminalfenster übertragen.

Wenn der Terminal-Multiplexer `Terminator` verwendet wird, ist es mit viel Tipparbeit verbunden in jeder einzelnen Konsole die SSH-Verbindung aufzubauen. Dafür gibt es das Bash-Skript `bb_connn_wifi.sh`. Hiermit muss in jeder Konsole nur noch der Befehl `./bb_con_wifi.sh` aufgerufen werden. ***Dabei muss vorher im Bash-Skript jedoch die richtige IP-Adresse bzw. der richtige Hostname editiert werden.*** *Quelltext siehe Abschnitt Fehler: Verweis nicht gefunden.*

5.2 Kommunikation via Ethernet über Switch im selben Netzwerk

Alternativ (und auch zusätzlich!) zu USB kann der BB auch via Ethernet angeschlossen werden. Dabei handelt es sich dann um eine echte und nicht nur emulierte Ethernetschnittstelle wie in Abschnitt 5.1 beschrieben.

Der BB muss zuerst extern mit Spannung versorgt werden, was z.B. über eine Powerbank oder über ein USB-Ladegerät inkl. Ladekabel am USB-Anschluss geschehen kann. (Der USB-Anschluss ist in diesem Fall eine reine Stromversorgungsschnittstelle!) Alternativ kann die Versorgungsspannung auch über VDD_5V (P9_5 bzw. P9_6) eingespeist werden.

Dann wird der BB mit einem Patchkabel über einen Switch mit dem Hochschulnetzwerk verbunden, damit sich im selben Netzwerk wie der PC befindet. Der BB erhält vom Hochschulnetz anschließend eine freie IP-Adresse zugewiesen (wird als „DHCP“ bezeichnet).

Hat der BB eine IP-Adresse im Hochschulnetz erhalten und besitzt er einen eindeutigen Hostnamen (z.B. `einzigartiger_bb`) , so kann von einem anderen PC im Hochschulnetz dieser BB auch ohne IP-Adresse über `einzigartiger_bb.local` erreicht werden (siehe auch Abschnitt 5.2.2).
Besteht kein Zugriff auf den Router des Netzwerks (wie beim Hochschulnetzwerk) und gibt es keinen eindeutigen Hostnamen, dann ist eine zumindest temporäre USB-Kommunikation nötig, um die IP-Adresse des BB herauszufinden.

Achtung:
Je nach Einstellungen des Routers (beim Hochschulnetzwerk auf jeden Fall) erhält der BB jedes Mal eine neue IP-Adresse, wenn er sich z.B. nach einem Reboot erneut im Netzwerk einbucht. Hat er jedoch einen eindeutigen Hostnamen (also nicht „`beaglebone`“ wie voreingestellt), dann kann der BB trotz neuer unbekannter IP-Adresse über seinen Hostnamen `einzigartiger_bb` mit dem Befehl `ssh beagle@einzigartiger_bb.local` verbunden werden [5].

5.2.1 Ermitteln der IP-Adresse des BB

Welche Adresse IP-Adresse der BB erhalten hat, kann man am PC über den Befehl
`ping beaglebone.local` erfahren.

Wenn dies nicht funktioniert, dann loggt man sich am besten über die USB-Verbindung `192.168.6.2` ein und fragt über den Befehl `ifconfig -a` die Netzwerkverbindungen des BB ab, siehe Abschnitt 5.3.1.

5.2.2 Eigenen Hostnamen des BB erstellen

Beim PAT sind ohne besondere Vorkehrungen mehrere BB mit dem selben Hostnamen „`beaglebone`“ im selben Netz. Deshalb sollte der eigene BB einen eindeutigen Hostname erhalten um ihn damit im Netzwerk ausfindig zu machen:

Hierfür müssen im Ordner `/etc` die beiden Dateien `hostname` und `hosts` editiert werden, indem der alte Hostname `beaglebone` durch den neuen ersetzt wird.

```
sudo nano /etc/hostname
sudo nano /etc/hosts
```

Im Anschluss muss ein Reboot mit dem Befehl `sudo reboot` ausgeführt werden. Im Terminalfenster erscheint als neuer Promt anschließend `beagle@` gefolgt vom neuen Hostnamen.

Kurzreferenzen wie der Editor *nano* zu bedienen ist, finden sich im Abschnitt 12.2 oder im Internet.

Ob der BB Verbindung zum Internet hat, kann über den Linuxbefehl
ping www.google.com getestet werden. Damit überprüft man erstens den DNS-Server und zweitens das Zustandekommen einer Kommunikation mit einer IP-Adresse von Google, von deren ständiger Erreichbarkeit auszugehen ist.

5.3 Kommunikation via WLAN

Es ist am einfachsten, den BB via USB mit dem PC zu verbinden, denn in diesem Fall erfolgt sowohl die Stromversorgung als auch die Kommunikation via USB. Diese Verbindung ist optimal vom Einbinden der Sensoren bis zu den Plausibilitätsprüfungen des Gesamtsystems mit aufgebocktem Roboterfahrzeug.

Tipp:
Verwenden Sie wenn immer es geht die USB-Verbindung, da diese leistungsstärker und weniger störanfälliger ist als die WLAN-Verbindung.

Um jedoch das Roboterfahrzeug im fahrenden Betrieb zu optimieren, muss eine Drahtlosverbindung verwendet werden. Zum Glück ist der BB mit einem Linux-Betriebssystem ausgestattet: Denn für den Umstieg auf WLAN ist am BB wie auch am PC nur ein normaler WLAN-Stick bzw. eine WLAN-PCI-Karte nötig.
Der BB kann als WLAN Accesspoint eingerichtet werden, in dessen WLAN sich der PC einbucht. In [15] ist die Vorgehensweise am Beispiel eines RasPi detailliert beschrieben. Diese Art der WLAN-Kommunikation hat sich jedoch als nicht ausreichend stabil erwiesen.

Daher werden im PAT mehrere einfache WLAN-Router (ohne Internetanschluss) verwendet, in die sich jeweils sowohl der PC als auch der BB einbuchen (siehe Abb. 2).

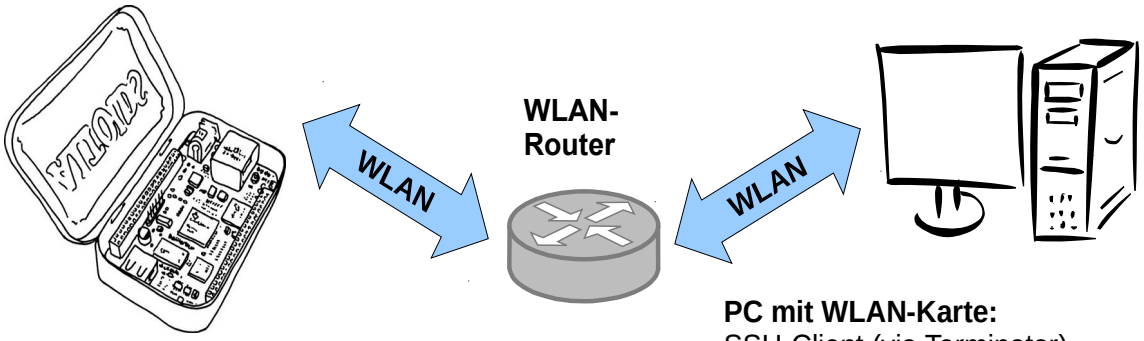


Abbildung 2: Kommunikation des BB via WLAN mit einem PC.

Achtung:
Bitte verbinden Sie den PC und den BB mit dem räumlich nächstgelegenen Router, damit die unterschiedlichen BB gleichmäßig auf die Router verteilt sind.

5.3.1 Mit dem BB in ein vorhandenes WLAN einwählen

Die nachfolgend genannte Prozedur wurde mit einem WLAN-Stick von Edimax, Modell EW-7811Un[7] durchgeführt. Dieser sollte vom BB-Betriebssystem automatisch erkannt und eingebunden werden, da der nötige Treiber (RTL8192CU) dort schon vorhanden ist.
Detailliert ist der Aufbau einer WiFi-Verbindung im Buch von D. Molloy [1] in Kapitel 12 beschrieben.

Zuerst muss sichergestellt werden, dass das Betriebssystem über den USB-Bus den WLAN-Stick während des Bootens überhaupt erkennt und den obigen Treiber geladen hat.
Dazu sollte man den WLAN-Stick erst nach dem Booten einstecken und danach in der Konsole mit dem Befehl *dmesg* nachschauen, ob folgende Meldung vom Kernel ausgegeben wird - die jetzt in der sehr großen Liste an Zeilen ganz unten steht:

```
[ 1106.244567] usb 1-1: new high-speed USB device number 2 using musb-hdrc
[ 1106.394189] usb 1-1: New USB device found, idVendor=7392, idProduct=7811, bcdDevice=
```

7 **Achtung:** Seit 2021 gibt es die Version 2 dieses WLAN-Sticks („EW-7811Un V2“ auf USB-Steckeraußenseite eingraviert). Diese Version funktioniert nicht mit dem aktuellen BB-Image!

```
2.00
[ 1106.394212] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 1106.394223] usb 1-1: Product: 802.11n WLAN Adapter
[ 1106.394232] usb 1-1: Manufacturer: Realtek
[ 1106.394242] usb 1-1: SerialNumber: 00e04c000001
[ 1106.706044] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[ 1106.723692] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[ 1107.165857] rtl8192cu: Chip version 0x10
[ 1107.292687] rtl8192cu: Board Type 0
[ 1107.292942] rtl_usb: rx_max_size 15360, rx_urb_num 8, in_ep 1
[ 1107.293189] rtl8192cu: Loading firmware rtlwifi/rtl8192cufw_TMSC.bin
[ 1107.321018] ieee80211 phy0: Selected rate control algorithm 'rtl_rc'
[ 1107.355821] rtl8192cu: MAC auto ON okay!
[ 1107.390561] usbcore: registered new interface driver rtl8192cu
```

Zusätzlich kann man auch mit *lsusb* prüfen, ob er als USB-Gerät angemeldet ist.

```
beagle@meinbeaglebone:~$ lsusb
Bus 001 Device 002: ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un 802.11n Wireless
Adapter [Realtek RTL8188CUS]
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Mit `ifconfig -a` wird geprüft, welches „Netzwerk Device“ nun dem WLAN-Stick zugeordnet wurde. In den Beispiel unten hat er den Namen „*wlan0*“ erhalten.

Achtung:
Jeder WLAN-Stick (selbst baugleiche) hat eine andere MAC-Adresse. Wenn man verschiedene WLAN-Sticks nacheinander mit dem BB verbindet, dann erhält der erste den Namen „wlan0“, der zweite „wlan1“ usw. Daher ist es wichtig, den Namen des aktuellen WLAN-Sticks zu kennen - denn man weiß ja nicht unbedingt, wie viele andere WLAN-Sticks schon vorher verbunden waren.

```
beagle@beaglebone:~$ ifconfig
eth0: flags=-28669<UP,BROADCAST,MULTICAST,DYNAMIC>  mtu 1500
        ether 84:eb:18:e3:6c:bb  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 55

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2928  bytes 179109 (179.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2928  bytes 179109 (179.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.7.2  netmask 255.255.255.252  broadcast 192.168.7.3
        inet6 fe80::86eb:18ff:fee3:6cbd  prefixlen 64  scopeid 0x20<link>
        ether 84:eb:18:e3:6c:bd  txqueuelen 1000  (Ethernet)
        RX packets 1  bytes 96 (96.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43  bytes 9762 (9.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.6.2  netmask 255.255.255.252  broadcast 192.168.6.3
        inet6 fe80::86eb:18ff:fee3:6cc0  prefixlen 64  scopeid 0x20<link>
        ether 84:eb:18:e3:6c:c0  txqueuelen 1000  (Ethernet)
        RX packets 750  bytes 57653 (57.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 601  bytes 108516 (108.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
wlan0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.178.71  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 fe80::76da:38ff:fe05:5bb  prefixlen 64  scopeid 0x20<link>
        ether 74:da:38:05:05:bb  txqueuelen 1000  (Ethernet)
        RX packets 279  bytes 27841 (27.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 75  bytes 12953 (12.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Im Beispiel oben hat der BB nur über den usb1-Anschluss eine IP-Verbindung (Adresse *192.168.6.2*) und dadurch eine IP-Adresse. Der WLAN-Stick hat vom Router die IP-Adresse *192.168.178.71* erhalten und ist somit mit dem WLAN-Netzwerk verbunden.

Wird der BB zusätzlich über Netzwerkkabel an einem Router angeschlossen, so ändert sich *eth0* im Vergleich zu oben: Nun hält die Schnittstelle eth0 ebenfalls eine IP-Verbindung (*192.168.178.51*).

```
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.178.51  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 fe80::86eb:18ff:fee3:6cbb  prefixlen 64  scopeid 0x20<link>
        ether 84:eb:18:e3:6c:bb  txqueuelen 1000  (Ethernet)
        RX packets 34  bytes 4287 (4.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 67  bytes 11115 (11.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 55
```

Mit dem Befehl *iwconfig* werden anders als bei *ifconfig* nur die Drahtlosverbindungen abgefragt:

```
lo        no wireless extensions.
usb1      no wireless extensions.
can1      no wireless extensions.
eth0      no wireless extensions.
wlan0     IEEE 802.11  ESSID:"Wehlan"
          Mode:Managed  Frequency:2.412 GHz  Access Point: E8:DE:27:55:A1:60
          Bit Rate=1 Mb/s   Tx-Power=20 dBm
          Retry short limit:7   RTS thr=2347 B   Fragment thr:off
          Power Management:off
          Link Quality=70/70  Signal level=-26 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:4   Missed beacon:0
can0      no wireless extensions.
usb0      no wireless extensions.
```

Um dem BB mitzuteilen sich mit einem bestimmten WLAN wifi_xyz zu verbinden, wird das Hilfsprogramm *connmanctl* (unbedingt mit *sudo* starten!) auf dem BB verwendet:

```
sudo connmanctl
connmanctl> scan wifi
Scan completed for wifi
connmanctl> services
    wifi_xyz             wifi_08beac010900_5765686c616e_managed_psk
    FRITZ!Box 6490 Cable wifi_08beac010900_465249545a21426f782036343930204361626c65
```

Auf das jeweilige WLAN wird über dessen Kennung rechts neben der gewünschten SSID zugegriffen. Diese kann mit „TAB-Vervollständigung“ eingegeben werden.

Mit dem Befehl

```
services wifi_08beac010900_5765686c616e_managed_psk
```

werden alle Infos zu dem betreffenden WLAN wie z.B. die Signalstärke angezeigt. Zum Verbinden einer gesicherten WLAN-Verbindung muss vorher ein *agent* eingeschaltet werden:

```
connmanctl> agent on
connmanctl> connect wifi_08beac010900_5765686c616e_managed_psk
Passwort eingeben
connmanctl> agent off
connmanctl> services
*AO wifi_xyz              wifi_08beac010900_5765686c616e_managed_psk
```

Hier bedeutet der * „bevorzugtes WLAN", das A „automatisches Verbinden", das R „Ready" (IP-Adresse von Router erhalten) und das O „Online". R bedeutet auch mit dem WLAN verbunden zu sein, O erscheint erst dann, wenn bestimmte IP-Adressen von *connman* erreicht wurden, was nicht relevant für das PAT ist. Details siehe [16]. Falls kein A erscheint, kann das automatische Verbinden mit folgendem Befehl aktiviert werden:

```
connmanctl> config wifi_08beac010900_5765686c616e_managed_psk autoconnect on
```

connmanctl wird mit dem Befehl *exit* beendet:

```
connmanctl> exit
```

Nun kann man mit *ifconfig wlan0* nachschauen, welche IP-Adresse der BB vom Router erhalten hat:

```
beagle@beaglebone:~$ ifconfig wlan0
wlan0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC>  mtu 1500
        inet 192.168.178.71  netmask 255.255.255.0  broadcast 192.168.178.255
        inet6 fe80::76da:38ff:fe05:5bb  prefixlen 64  scopeid 0x20<link>
        ether 74:da:38:05:05:bb  txqueuelen 1000  (Ethernet)
        RX packets 955  bytes 88593 (88.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 157  bytes 27200 (27.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Jetzt besitzt der BB in diesem konkreten Beispiel also zusätzlich die IP-Adresse *192.168.178.71*, die über WLAN zu erreichen ist. Diese kann nun auch für die SSH-Verbindung statt der USB-IP-Adresse *192.168.6.2* verwendet werden. Nach ca. einer Minute wird die nun bestehende WLAN-Verbindung auch durch die USR3-LED über ein Blinken mit 1 Hz angezeigt.

Mit dem PC muss man sich nun in das selbe WLAN-Netzwerk einbuchen, damit eine WLAN-Verbindung zum BB entsteht. Ob die Verbindung PC-BB steht, prüft man in diesem Beispiel in einem Terminalfenster mit *ping 192.168.178.71* oder (falls eigener Hostnamen vergeben) *ping einzigartiger_bb.local*.

Achtung:
Wenn Sie den BB später ausschließlich über eine Powerbank versorgen möchten, dann muss die WLAN-Verbindung automatisch nach dem Booten hergestellt werden, da für die Kommunikation die USB-Schnittstelle schon mit der Powerbank belegt ist.

Dafür wurde das Bash-Skript wifi_reset.sh erstellt, welches einmal pro Minute von dem Linux-Tool Cron ausgeführt wird. Quelltext siehe Abschnitt 12.12.
Es überprüft, ob eine WLAN-Verbindung besteht. Falls nicht wird über das Tool connmanctl die WLAN-Verbindung neu aufgebaut.

5.4 Bekannte Probleme:

- Oft identifizieren Router einen bestimmten BB über dessen MAC-Adresse und weisen ihm immer die selbe IP-Adresse zu. Hat sich der Hostname geändert, dann kann es bei erneutem Verbinden mit dem WLAN Router zu Problemen kommen, weil sich die Kombination aus MAC-Adresse und Hostname geändert hat.
- Wenn die WLAN-Verbindung nicht aufgebaut werden kann, hilft manchmal ein Neustart der connman Services: *sudo systemctl restart connman*
- Das Hilfsprogramm *connmanctl* speichert die bisher hergestellten WLAN-Verbindungen unter *var/lib/connman*. Dort existiert für jede Kennung ein Verzeichnis. Bei Verbindungsproblemen hilft es manchmal das entsprechende Verzeichnis zu löschen, den BB neu zu starten und die Verbindung von Anfang an neu herzustellen.
- Zusammen mit dem WLAN-Dongle kann der BB kurzzeitig einen hohen Strom benötigen. Dies kann je nach Stromversorgung bzw. angeschlossenen Verbrauchern zu einem Absinken der USB-Spannung SYS_5V auf unter 4,5 V und damit zu Instabilitäten der WLAN-Verbindung führen.
- Die Version 2 des Edimax WLAN-Sticks („EW-7811Un V2“ auf USB-Steckeraußenseite eingraviert). funktioniert nicht mit dem aktuellen BB-Image.

Tipp:
Manchmal ist es schwierig zu erkennen, ob sich der BB überhaupt in das (richtige!) WLAN eingebucht hat. Wenn Sie ein Smartphone dabei haben, dann können Sie sich mit diesem in das selbe WLAN einbuchen und mit einer Netzwerk-Scanner App (z.B. Fing) nachschauen, ob Ihr BB im selben Netz zu finden ist. Hat

Ihr BB noch keinen eigenen Hostnamen, so können Sie Ihren ihn über dessen MAC-Adresse (des WLAN-Dongles) identifizieren und mit *ping* die Verbindung testen.

6 Umgang mit Linux

Kommt man aus der Welt der Windows/Apple-PCs oder Androidsmartphones, so kann man sich eine Welt ohne grafische Benutzeroberfläche gar nicht mehr vorstellen.

Bei Linux Embedded Systems macht eine solche Benutzeroberfläche jedoch keinen Sinn, da sie nur unnötig Rechenleistung verbraucht. Tatsächlich ist der Umgang mit einer sogenannten "Konsole", bei der in einem Terminalfenster die Befehle in Form von Text statt Mausklicks eingegeben werden, langfristig wesentlich effizienter.

Tipp:
Eine nützliche Funktion ist das automatische Vervollständigen („Tab Completion") von Verzeichnis- oder Dateinamen:
Bei der Eingabe eines Pfades kann man mit der Tab-Taste die Namen vervollständigen, wenn man die ersten Buchstaben eingegeben hat. Dies funktioniert auch bei allen ROS-Tools.
Auch kann ein aus einem PDF (wie dieses hier) kopierter Befehl mit Copy/Paste in die Linuxkonsole eingefügt werden.

Die im Quellenverzeichnis genannten Fachbücher [17], [3] und [4] enthalten jeweils die wichtigen Infos und Befehle zum Thema Linux. In der Praxis sind Cheat Sheets für Linux sehr nützlich, ein Beispiel hierfür finden Sie im Anhang im Abschnitt 12.3.

6.1 Dateimanagement auf dem BB, Übertragen von Dateien zwischen PC und BB

Vom PC aus kommunizieren Sie mittels SSH-Shell mit dem Linux-Betriebssystem auf dem BB. Dies geschieht wie oben beschrieben über Tastaturbefehle im Terminalfenster. Damit können Dateien innerhalb des BBs verschoben, umbenannt, gelöscht usw. werden.

Möchte man zwischen dem PC und dem BB Dateien übertragen, so kann dies textbasiert innerhalb eines Terminalfensters gemacht werden oder mit einem PC auch grafikbasiert beispielsweise bei Ubuntu über den Standard-Dateimanager *Nautilus*.

6.1.1 Dateiübertragung zwischen PC und BB im Terminalfenster

Hierfür gibt es das Tool *sftp* (Secure File Transfer Protocol).

Bei *sftp* kennt im Prinzip die gleichen Kommandos wie die Bash-Shell (z.B. *pwd*, *ls*, *cd*, *mkdir*, *rm*). Der PC wird als „local machine“ bezeichnet, der BB ist die „remote machine“.
Die Logik hinter *sftp* ist folgende: Wird dem Befehl ein „l“ für „local“ vorangestellt, dann bezieht sich dieser auf den PC. Wenn nicht, dann bezieht sich der Befehl auf den BB.
Zusätzlich gibt es noch die beiden Kommandos *put* und *get*: *put* bewirkt einen Upload vom PC auf den BB und *get* einen Download vom BB auf dem PC, wie folgendes Beispiel zeigt:

```
mackst@tec-04-206-02:~$ sftp beagle@192.168.6.2
sftp> pwd
Remote working directory: /home/beagle
sftp> lpwd
Local working directory: /home/mackst
sftp> put dateiAufPC.py
Uploading dateiAufPC.py to /home/beagle/dateiAufPC.py
dateiAufPC.py                                 100%  716   334.1KB/s   00:00
sftp> get dateiAufBB.py
Fetching /home/beagle/dateiAufBB.py to dateiAufBB.py
/home/beagle/dateiAufBB.py                    100%  716   226.5KB/s   00:00
sftp> exit
mackst@tec-04-206-02:~$
```

Es können auch Dateien von einem USB-Stick vom und zum BB kopiert werden. Dazu wird nach dem Einstecken des USB-Sticks mit dem Befehl *fdisk -l* nachgeprüft, ob dieser als */dev/sda1* erscheint.
Nun muss der USB-Stick noch „gemountet“ werden, d.h. es muss ein Verzeichnis angelegt werden, über das auf ihn zugegriffen werden kann. Dafür wird z.B. das Verzeichnis */media/usb* angelegt (*mkdir /me-*

dia/usb) und anschließend mit dem Befehl *mount /dev/sda1 /media/usb* mit dem USB-Stick verbunden.

6.1.2 Dateiübertragung zwischen PC und BB grafikbasiert

Die Übertragung an sich findet hier im Hintergrund auch über die Protokolle *SSH* und *sft* statt.

Bei einem Ubuntu-PC kann der Standard-Dateimanager *Nautilus* schon mit den o.g. Protokollen umgehen. Um eine Verbindung mit dem BB aufzubauen, wird hier „+ Andere Orte" ausgewählt. Dann wird in der Eingabezeile „Mit Server verbinden" die SSH-Adresse des BB *sftp://beagle@192.168.6.2* eingegeben. Nach einem Klick auf „Verbinden" wird das Passwort abgefragt und anschließend erscheint in *Nautilus* das Dateisystem des BB, mit dem nun genau so gearbeitet werden kann wie mit dem lokalen Dateisystem des Ubuntu-PC.

Für Windows-PC muss als sogenanntes „Frontend" die Software WinSCP installiert werden.

7 Die Ein- und Ausgänge des BB

Achtung:
Gegenüber früheren BB-Betriebssystemversionen hat sich der Umgang mit den I/Os seit der Kernelversion 3.14 grundlegend geändert: Für die Pinkonfiguration wird nicht mehr das Tool *capemanager* sondern das Tool *config-pin* [18] verwendet. Sogenannte "Slots" für (virtuelle) Capes werden durch aktuelle Kernelversionen nicht mehr unterstützt.
Dies ist bei älteren Informationen (vor 2018) aus dem Netz unbedingt zu beachten. Nur die Second Edition des Buchs von C. Molloy [1] ist in Bezug auf I/Os und IP-Kommunikation auf dem aktuellen Stand.

Was das Anschließen von I/O-Peripherie wie Sensoren betrifft, ist dieses Kapitel von sehr großer Bedeutung. Denn viele Kompilier- und Laufzeitfehler beruhen auf Konflikten bei der Pinkonfiguration.

Grundsätzlich sind auf den beiden zweireihigen Buchsenleisten P8 und P9, den sogenannten „Cape Expansion Headers“ Versorgungsspannungen (3,3 und 5 V), Masse, Resets, Analogeingänge und GPIO herausgeführt.

Die Abkürzung GPIO bedeutet „General Purpose Input Output“. In der Literatur werden damit aber oft nur die binären (digitalen) I/Os bezeichnet. GPIO können aber auch als CAN-, I²C-, UART-, SPI-Schnittstelle oder oder als Timer konfiguriert werden.

Die Zuordnung der unterschiedlichen GPIO-Funktionen zu den einzelnen Buchsen von P9 und P8 wird "Pinkonfiguration" genannt. Allgemein spricht man hier von „Pin Muxing“, zu Deutsch „Anschlussmultiplexing“.

Eigentlich bräuchte man bei Linux für unterschiedliche Pinkonfigurationen unterschiedliche Kernels (=Betriebssystemversionen). Denn normalerweise ist die Pinzuordnung im Betriebssystem = Kernel festgelegt. Aufgrund der vielen verschiedenen ARM-Prozessoren wären folglich viele verschiedene Kernelversionen nötig.
Um dies zu umgehen, wurde der sogenannte "**Device Tree (DT)"** eingeführt: Hier wird einem einheitlichen Kernel beim Bootvorgang die gewünschte (nicht einheitliche) Pinkonfiguration mitgeteilt und gleichzeitig auch die nötigen Kernelmodule dafür geladen. Nach dem Booten hat also jeder Pin eine Default-Konfiguration. Auch nach dem Booten kann die Pinkonfiguration mit dem Tool `config-pin` noch geändert werden. Man spricht dann von einem **"Device Tree Overlay (DTO)"**, der geladen wird.

Der Zugriff auf die binären I/O (Schalteingänge, Schaltausgänge), den ADC und den PWM-Ausgang ist bei allen LES über das **Sys File System** möglich (siehe Abschnitt 7.2):

Das Setzen eines binären Ausgangs oder das Auslesen einer gewandelten Eingangsspannung ist somit nichts Anderes als das Schreiben oder Lesen einer entsprechenden virtuellen Datei im Linux-Dateisystem. Dieses virtuelle Dateisystem wird „Sys File System" genannt.

Daher benötigt man für die binären I/Os, den ADC und PWM (=DAC) sowohl für die Programmiersprache C als auch für Python prinzipiell keinen Treiber, denn es sind nur Schreib- oder Lesebefehle auf Dateien notwendig. Diese Programmierschnittstelle wird oft auch als "SysFS-API" bezeichnet.

Damit die entsprechenden virtuellen Dateien überhaupt im Linux-Dateisystem auftauchen, muss vorher der DT (beispielsweise für den ADC oder den PWM) geladen werden.

Geschieht dies beim Booten, dann spricht man von "Device Tree". Geschieht dies mit dem Tool *config-pin* nach dem Booten, dann spricht man von "Device Tree Overlay".

Im PAT werden die benötigten Schnittstellen nach dem Booten innerhalb der Programme zu deren Laufzeit oder über automatisch ausgeführte Bash-Skripte mittels einzelner *config-pin* Befehle konfiguriert. Die analogen Eingänge, der I²C Bus 2 sowie die binären I/O sind schon durch den DT direkt nach dem Booten konfiguriert.

Die Schnittstellen SPI, I²C, UART, USB und Ethernet benötigen zusätzlich Treiberbibliotheken. In Python verwendet man der Einfachheit halber gemeinsame Bibliotheken für alle Schnittstellen (GPIO bis I²C), die teils sogar wie die BBIO-Bibliothek das Laden der nötigen Pinkonfiguration via `config-pin` mit übernehmen.

In dem Buch von D. Molloy [1] sind in Kapitel 6 die Ein- und Ausgänge des BB sowie das Tool `config-pin` recht gut beschrieben. Außerdem findet sich auf seinen Internetseiten zu Kapitel 6 des Buchs („Interfacing to BeagleBone Busses“) [19] tabellarische Übersichten zu den erlaubten und möglichen Pinbelegungen, die

auch in Abschnitt 12.5 abgebildet sind.

Recht hilfreich sind auch Aufkleber mit den Pinnummern, die man an die beiden Buchsenleisten klebt. Man kann sie als PDF-Datei unter [20] oder [21] herunterladen.

Im PAT werden jedoch diese Pins nicht direkt über die Buchsenleisten sondern geschützt über die Grove-Stecker des Grove-Capes kontaktiert.

Achtung:
Leider ist die ganze Geschichte mit den DT und DTO in der Realität nicht so sauber implementiert wie oben beschrieben. Bei einigen Images ist die Pinzuordnung dann doch Teil des Kernels und das zusätzliche Laden eines DT bzw. DTO hat nicht den gewünschten Effekt.
Bestimmte Pins wie z.B. P8_20 bis P8_25, die Schnittstellen zum eMMC (siehe Abschnitt 12.5) dürfen nicht umkonfiguriert werden.
Beim Verwenden von `config-pin` besteht immer die Gefahr, eine vorher gesetzte Pin-Konfiguration ungewollt zu überschreiben.
Daher wird empfohlen, die Pin-Konfiguration des BB-Images für das PAT so zu verwenden wie sie in Abschnitt 7.1 dokumentiert ist.

7.1 Die Standard Pinkonfiguration für das PAT

Verwenden Sie möglichst diese Pinbelegung und Kontaktierung über die Grove-Stecker während der gesamten Projektarbeit. Damit haben Sie schon eine Hauptfehlerquelle beseitigt!

GPIO siehe Abschn. 12.5	Entsprechende Pins der Buchsenleiste am BB	Beschriftung Grove-Stecker auf Cape Version 2	Primäre Verwendung im PAT und Alternative dazu
30, 31	P9_11, P9_13, VCC, GND UART4: P9_11 RX, P9_13 TX	UART/Digital links (UART4)	bin. I/O, alt. UART4 (Nur über Buchsenleiste[8])
14, 15	P9_26, P9_24, VCC, GND UART1: P9_26 RX, P9_24 TX	UART/Digital rechts (UART1)	bin. I/O, alt. UART1 (Nur über Buchsenleiste), I²C, CAN
-	P9_37-40, VCC, GND	Analog Input	Analog In
51	P9_16, NC, VCC, GND	GPIO 51	PWM, alt. bin. I/O
50	P9_14, P9_16, VCC, GND	GPIO 50	PWM, alt. bin. I/O
117	P9_25, P9_14, VCC, GND	GPIO 117	bin. I/O bevorzugt Ausgang
115	P9_27, P9_25, VCC, GND	GPIO 115	bin. I/O bevorzugt Eingang
13 CL, 12 DA	P9_19 CL, P9_20 DA, VCC, GND	I2C2 (Hub für 4 Slaves)	I²C2, alt. CAN, Timer
84 RX, 85 TX	UART2: P9_22 RX, P9_21 TX SPI: P9_22 CL, P9_21 D0	Oberer Stecker auf BBG-Board neben USER-Button bzw. über SD-Karte (UART2)	UART2, SPI0 zusammen mit P9_17 CS, P9_18 D1, PWM

7.2 Das SysFS

Unter Linux wird auf die binären I/Os, den ADC und den PWM (=DAC) über Lese- und Schreibbefehle auf virtuelle Dateien (das System File System kurz SysFS) zugegriffen. Vorher muss jedoch die benötigte Pinkonfiguration (z.B. mit dem Tool `config-pin`) gesetzt sein. Näheres zu `config-pin` siehe Abschnitt 12.15.

Jeder Schnittstelle ist im Verzeichnis in `/sys/class/` ein entsprechender Ordner zugeordnet. In dessen Dateien kann man Werte hineinschreiben um z.B. einen binären Ausgang auf High zu setzen. Genauso kann man eine Datei auslesen, um den Wert des ADCs zu erhalten. Das Datei System nennt man „SysFS“. Es ist eine Besonderheit des Linux-Betriebssystems.

Das SysFS ist also ein virtuelles Dateisystem, über das man **als User** direkt auf Betriebssystemfunktionen

8 Betrieb von UART1 und UART4 über Grove-Stecker funktioniert aufgrund Hardwarefehler am Grove-Cape Version 2 nicht zuverlässig. Diese beiden UART daher direkt über die Buchsenleiste kontaktieren.

zugreift, um über die I/Os zu kommunizieren.
Dieser Umgang mit den I/Os durch das Betriebssystem ist also grundsätzlich bei allen LES gleich. Schreibt man ein C-Programm, dass in dieser Weise die Schnittstellen anspricht, so ist es einfach auf ein anderes LES portierbar. Daher unterscheidet sich der C- oder Pythonquellcode für einen BB nicht wesentlich von dem für einen RasPi abgesehen von einigen hardwarespezifischen Bibliotheken.

In den Beispielprogrammen des PAT erfolgt der Zugriff auf die I/Os teils über dieses SysFS. Die folgenden Beispiele verdeutlichen, wie in einem Terminalfenster mit Linuxbefehlen auf das SysFS zugegriffen wird.

Beispiel 1:

Binärer Eingang P9_25 (GPIO 117, siehe Abschnitt 12.5):

```
cd /sys/class/gpio
ls
```

Der obige Befehl listet die "exportierten" GPIO auf. Nach dem Booten des PAT-Image gehören fast alle GPIO dazu. Maximal erscheinen hier 4 x 32 GPIO der vier GPIO-Chips 0, 32, 64 und 96. Es sind aber nicht alle dieser möglichen GPIO überhaupt auf die Buchsenleisten herausgeführt oder im DT nach dem Booten als GPIO konfiguriert.

Auf einen dieser GPIO wird mit folgenden Schreib-/Lese-Befehlen zugegriffen (hier Konfiguration als Input und Auslesen des Eingangssignals):

```
cd gpio117
echo in > direction
cat value
```

Bestimmte Parameter der GPIO wie Pull-Ups können nur mit dem Tool *config-pin* eingestellt werden. Ansonsten bewirken die oben dargestellten Dateioperationen im SysFS aber das selbe wie entsprechende Befehle über das Tool *config-pin*.

Achtung:

Bestimmte binäre I/Os können grundsätzlich nicht ohne Weiteres verwendet werden. Diese sind in den Tabellen in Abschnitt 12.5 rot unterlegt.

Beispiel 2: Analoger Eingang AIN0 an P9_39

```
cd /sys/bus/iio/devices/iio\:device0
cat in_voltage0_raw
```

Die Ausgabe dieses Befehls ist die Eingangsspannung am analogen Eingang AIN0 in LSB (FSR 1,8 V).

Beispiel 3: PWM Signal an P9_14 einstellen

Zunächst muss der Pin mit dem Tool *config-pin* als PWM-Ausgang konfiguriert werden, da er über den DT standardmäßig als binärer I/O konfiguriert ist. (Im PAT-Image wird dies automatisch beim Booten des BB durchgeführt.)

Um dann die PWM-Einstellungen im SysFS vorzunehmen muss man herausfinden, zu welchem PWM-Hardware-Namen und -Kanal dieser Pin gehört. Die Zuordnung des PWM-Chips im SysFS ist leider nicht eindeutig. Hier ist es das Verzeichnis *pwmchip4*.

```
config-pin p9.14 pwm
cd /sys/class/pwm/pwmchip4
echo 0 > /sys/class/pwm/pwmchip4/export
cd pwm-4\:0
echo 4000 > period
echo 1000 > duty_cycle
echo 1 > enable
```

Die PWM-Periode und -Duty Cycle sind jeweils in µs angegeben.
Im SysFS gibt es mehrere PWM-Chips mit jeweils zwei Kanälen. Dabei können via *config-pin* für jeden Kanal jeweils zwei Pins ausgewählt werden. Wählt man z.B. PWM-Chip 4 und Kanal 0A aus, so wird über das SysF *sys/class/pwm/pwmchip4/pwm-4:0* der Pin P9_14 angesteuert sofern dieser vorher mit *config-pin* für PWM konfiguriert wurde.

Hardwarename	PWM Chip des BB	Kanal	Pin des BB
Ehrpwm1a bzw. ehrpwm1b	*pwmchip4*	4A mit *pwm-4:0* bzw. 4B mit *pwm-4:1*	P9_14 bzw. P9_16

Für die Schnittstellen I²C, SPI und UART gibt es entsprechende SysFS-Dateien für das Lesen und Schreiben. Jedoch werden zusätzlich Treiberbibliotheken benötigt. Details siehe Buch von D. Molloy, Kapitel 6 [1].

7.3 Die I²C-Busse

Eine sehr gute Anleitung zur Verwendung der I²C-Busse auf LES findet sich in [22].

Folgende Pakete werden für die I²C-Kommunikation benötigt. Mit der folgenden Befehlsfolge werden sie aktualisiert bzw. installiert (nur auf eigenem PC nötig):

```
sudo apt-get update
sudo apt-get install i2c-tools       # I2C-Toolkit fuer die Kommandozeile
sudo apt-get install python3-smbus   # Python-Bibliothek fuer I2C
sudo apt-get install libi2c-dev      # Bibliothek fuer C
```

Mit dem Tool *i2c-tools* kann die I²C Kommunikation zu angeschlossenen Peripheriegeräten (Slaves) wie Sensoren überprüft werden.

Mit dem Befehl *i2cdetect -y -r 2* wird der I²C-Bus 2 nach angeschlossenen Geräten durchsucht.

```
beagle@beaglebone:~$ i2cdetect -y -r 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

Oben ist die Ausgabe dieses Befehls im Terminalfenster dargestellt: Es wurde an der Adresse 0x70 ein Gerät gefunden – in diesem Fall ein SRF02 Ultraschallsensor an den Pins P9_20/_19.

Die Zeichen „UU“ in der Tabelle bedeuten, dass diese Adressen schon im Zugriff eines Treibers sind und daher nicht ausgelesen werden.

Mit dem Befehl *i2cdetect -l* (Achtung nicht „-1“ sondern „-l“) werden die verfügbaren Busse aufgelistet. Beim BB sind dies nach dem Booten i2c-0, i2c-1 und i2c-2.

Achtung:
Auf den ebenfalls angezeigten Bus i2c-0 kann der BB nur intern zugreifen. De Bus i2c-1 wird am Grove Cape Version 2 nicht als Stecker herausgeführt.

Die folgende Tabelle gibt die verschiedenen Bezeichnungen der I²C-Busse sowie deren zugeordnete Pins an.

Grove Cape	SysFS	SDA/SCL Pin	SysFS
I2C2	i2c-2	P9_20/_19	*/dev/i2c-2*

Weiter können mit *i2c-tools* alle Register eines angeschlossenen Slaves ausgelesen werden. Für den SRF02-Sensor würde dies mit dem Befehl *i2cdump -y 2 0x70* ausgeführt werden, um z.B. die Parametrierung zu kontrollieren. Auch kann man direkt per Kommandozeile mit dem Slave kommunizieren.

Bei den meisten Sensoren kann man die Adresse deren I²C-Schnittstelle ändern, was z.B. nötig wird, wenn man am selben I²C-Bus zwei gleiche Sensoren verwenden möchte. In dem Python-Skript *SFR02Address-Change.py* ist dies exemplarisch für den Ultraschallsensor SRF02 gezeigt.

7.4 Die UART-Schnittstelle

Im SysFS Dateisystem erscheint die UART2-Schnittstelle als */dev/ttyO2*. Achtung: Das zweitletzte Zeichen im Pfad ist ein "O" und keine Null!
Die hier zugeordneten Pins P9_22 und 21 werden nicht über das Grove Cape sondern auf der BBG-Platine oberhalb des SD-Kartenslots herausgeführt. Für die Pythonprogrammierung wird die Bibliothek *pySerial* verwendet. Die Programmierung in C funktioniert ohne spezielle Schnittstellenbibliothek über das SysFS, siehe [23].

Aus einem Terminalfenster lässt sich die UART-Verbindung auf dem BB mit dem Tool *minicom* unkompliziert testen. Für die UART2-Schnittstelle (*/dev/ttyO2*) bei 115200 Baud und 8-N-1-Konfiguration wird *minicom* mit *minicom -b 115200 -o -D /dev/ttyO2* aufgerufen. Für einen Funktionstest eignet sich ein Arduino mit der in Abschnitt 12.11 dargestellten Firmware, die die Zeichen vom BB als Echo an diesen zurück sendet.

7.5 Hardwarespezifisches

Die Analogeingänge haben unter den GPIO eine Sonderstellung und sind streng genommen ja auch gar keine "General" PIO: Sie befinden sich immer an den selben Anschlussbuchsen, die auch nicht durch andere Schnittstellen belegt werden können. Der Grund hierfür ist die nötige Analogelektronik, die nicht einfach über einen Multiplexer weitergeleitet werden kann. Sie sind im Image des PAT automatisch durch den DT konfiguriert.

Achtung:

Die 5 V Spannung an P9_5 bis P9_8 oder über das Grove Cape ist sehr unsauber, d.h. mit vielen Störspannungen überlagert. Soll sie Sensoren versorgen, dann muss diese Spannung mit einem Kondensator geglättet werden. Die 3,3 V Spannung am BB enthält erheblich weniger Störanteile. Verbraucher mit hohem Störpotential wie z.B.die Motortreiber dürfen nicht mit 3,3 V betreiben werden, da der BB dadurch gestört wird. Zudem ist die 5 V Spannungsversorgung wegen des fehlenden Spannungsreglers leistungsfähiger.
Der Maximalstrom für die 5 V Spannung beträgt bei Speisung über VDD_5V 1 A und bei Speisung über den Micro-USB-Anschluss bzw. SYS_5V 500 mA.

8 Einbinden von Sensoren und Aktoren über Python-Skripte

Python ist genial und ersetzt immer mehr MATLAB. Eine gute Übersicht, was man inzwischen alles mit dieser kostenlosen Open Source Programmiersprache machen kann, finden Sie z.B. in [24] oder in [25].

Für das Einbinden von Sensoren und Aktoren (allgemein I/O-Peripherie) in ein LES ist die Programmiersprache Python eigentlich nicht geeignet. Denn als Skriptsprache weist sie eine vergleichbar geringe Ausführungsgeschwindigkeit auf, bei der nur bedingt von Echtzeitfähigkeit gesprochen werden kann. Ideal verwendet man hier C-Programme, zumal damit auch die PRU des BB mit eingebunden werden kann. Siehe hierzu Kapitel 8 in [3].

In [1] wurde für den selben Algorithmus die Programmlaufzeit auf dem BB für verschiedene Programmiersprachen verglichen. Weiter wurde für ein zweites Benchmarking ein einfaches Toggeln des GPIO45 (P8_11) einmal mit C und einmal mit Python programmiert, und anschließend mit einem Oszilloskop die maximal mögliche Togglefrequenz gemessen. Wie in der nachfolgenden Tabelle zu sehen ist, liegt Python bei beiden Vergleichstests weit abgeschlagen.

Programmiersprache	**C/C++**	**Java**	**Python**
Programmlaufzeit (s)	33	39	1063
Frequenz Pin Toggle (kHz)	93	-	5,3

Python wird mit dem Fokus auf die einfache Erlernbarkeit und einfache/klare Syntax von einer gemeinnützi-

gen Organisation entwickelt[9]. Python ist somit sehr viel einfacher als C. Zudem existieren sehr viele Bibliotheken und Beispielprogramme für das Einbinden der Peripherie am BB [5]. Bei fast allen LES wird für das Rapid Prototyping im Sinne von „Rapid“ die Programmiersprache Python eingesetzt. Ein gutes Lehrbuch zu Python ist z.B. [26].

Achtung:
Der wohl auffallendste Unterschied zwischen Python und etablierten Programmiersprachen wie Java oder C ist die Tatsache, dass zusammenhängende Codeblöcke nicht über Klammern sondern über Einrückungen definiert werden. Die Formatierung des Quellcodes bestimmt also dessen Inhalt mit. Dabei ist es wichtig, bei den Einrückungen nicht Tabs und Leerzeichen zu mischen! **Verwenden Sie am besten ausschließlich Leerzeichen.**

Python spielt auch beim RasPi eine wichtige Rolle. Da dessen Hardware I/O bis auf die fehlenden ADC recht ähnlich zu denen des BB ist, können RasPi-Python-Skripte einfach auf dem BB portiert werden. Dies betrifft z.B. Beispielprogramme für das Einbinden von Sensoren, siehe [27].

Im PAT wird Python hauptsächlich verwendet, um die I/O-Peripherie vor dem Einsatz zu testen und anschließend Python-ROS-Knoten dafür zu coden. Die interaktive Python-Shell startet man mit dem Befehl *`python3`* in einer Konsole. Beendet wird der Python-Interpreter mit *`strg + d`* oder *`strg + c`*, wenn ein Skript ausgeführt wird.

Achtung:
Im PAT wird ausschließlich Python 3 verwendet. Oft findet man aber Programmbeispiele noch als Python <=2.7 Code. In diesem Fall müssen folgende Inkompatibilitäten berücksichtigt werden: Ganzzahlmultiplikation, Verwendung von Umlauten, *`print`*-Anweisung. Näheres siehe [28].

8.1 Phyton-Bibliotheken

Beide Pythonversioinen sind auf dem BB-Image für das PAT installiert. Darüber hinaus sind die nötigen Python-Bibliotheken vorhanden, um mit der I/O-Peripherie zu kommunizieren.

Die Fa. Adafruit stellt für die Kommunikation über die GPIO-Schnittstellen ADC, PWM, I²C, SPI und UART die universelle Bibliothek *`Adafruit_BBIO`* für den BB bereit [5]. Statt der *`Adafruit_BBIO`* können auch die jeweiligen Originalbibliotheken wie *`spidev`*, *`pySerial`* oder *`python-smbus`* (für I²C) verwendet werden.

Achtung:
Zum Anschließen der Peripherie muss unbedingt das Grove Cape aufgesteckt werden, da sonst bei falschem Anschließen der BB zerstört wird.
Es sind jedoch auch dann nur die vierpoligen Grove-Stecker geschützt! Die beiden Buchsenleisten werden vom Grove Cape auf dessen eigene identische Buchsenleisten nur durchkontaktiert.
Der BB muss immer ausgeschaltet sein (nicht ausstecken, sondern über POWER-Taste herunterfahren!) bevor neue Peripherie angeschlossen wird. Den BB erst dann wieder einschalten, wenn Sie die Verdrahtung überprüft wurde.
Außerdem darf an den Anschlüssen des BB keine Spannung anliegen, wenn dieser ausgeschaltet ist.

Zum Öffnen der Beispielprogramme bzw. zum Schreiben eigener Pythonprogramme wird im PAT der Editor *`Nano`* verwendet. Dieser ist rein textbasiert und am Anfang etwas gewöhnungsbedürftig. Hilfreich ist hierbei ein Cheat Sheet, siehe Abschnitt 12.2.

Um eine Quellcodedatei *`myCode.py`* zu erstellen, wird in der Linuxkonsole der Befehl *`nano myCode.py`* eingegeben, der den Texteditor *`Nano`* öffnet. Darin gibt man den Quellcode ein.

Anschließend wird die fertige Skriptdatei mit dem Befehl *`python3 myCode.py`* im Interpreter ausgeführt. Wenn das Skript eine Endlos-Whileschleife beinhaltet, wird der Interpreter mit *`Strg + c`* beendet.

Die erste Zeile im Python-Quellcode gibt das Verzeichnis des Interpreters an, man nenn sie „Shebang“. Wenn man vorher mit dem Befehl *`chmod +x myCode.py`* der Quelldatei Ausführungsrechte gegeben hat, dann kann man auch nur mit dem Befehl *`./myCode.py`* das Programm starten. Die zweite Zeile sorgt dafür, dass deutsche Umlaute zu keiner Fehlermeldung führen.

Folgende Beispielprogramme befinden sich auf dem Image des PAT bzw. im GitHub Repository:

9 Der Name „Python“ wurde der Komikergruppe „Monty Python“ entlehnt – im Sinne der fehlenden Ehrfurcht gegenüber der Programmiersprache C.

Programmname	Funktion	Bemerkung
myBlink.py	Toggeln des GPIO 117 (P9_25)	via Bibliothek *Adafruit_BBIO*
myBlink_SysFS.py	Toggeln des GPIO 117 (P9_25)	via SysFS
myBlink_SysFS_OS.py	Toggeln des GPIO 117 (P9_25)	via SysFS, byteweises Schreiben
myGPIOread.py	Auslesen GPIO 115 (P9_27)	via SysFS
myADC.py	Auslesen des AIN0 (P9_39)	via Bibliothek *Adafruit_BBIO*
myADC_SysFS.py	Auslesen des AIN0 (P9_39)	via SysFS
mySRF02_smbus.py	Auslesen Abstandswert in cm Ultraschallsensor via I²C-Bus 2	via Bibliothek *smbus, Ändern I²C-Adresse mit SRF04AddrChange.py*
myKalmanSRF02.py	Einfacher Kalmanfilter mit den Echolaufzeitdaten des SRF02	via Bibliothek *smbus, Ändern I²C-Adresse mit SRF04AddrChange.py*
myPWM.py	PWM-Signal auf GPIO 50 (P9_14)	via Bibliothek *Adafruit_BBIO*, für Modellbauservo gedacht
myMot.py	PWM-Signal auf GPIO 50 (P9_14), GPIO 51 (P9_16), Richtungssignal via GPIO 117 (P9_25), GPIO 115 (P9_27)	via Bibliothek *Adafruit_BBIO*, für beide DC-Motoren des Robo-ToGo gedacht
myPWM_SysFS.py	PWM-Signal auf GPIO 50 (P9_14)	via SysFS, für Modellbauservo gedacht
myUART.py	Serielle Kommunikation mit Arduino (Firmware *UARTComm_BB_Ardu.ino*) über UART2-Schnittstelle (RX: P9_22, TX: P9_21)	via Bibliothek *pySerial*
myHMC5883L.py	Auslesen des AMR-Sensors myHMC5883L	via Bibliothek *smbus*
myVL53L1X.py	Auslesen des TOF-Sensors myVL53L1X	via Bibliothek *smbus,* benötigt *VL53L1XRegAddr.py* im selben Verzeichnis für Registeradressen
myAPDSCo.py, myAPDSPr.py, myAPDSGe.py, myAPDSInterrGe.py	Auslesen von Farbe, Abstand, Geste bzw. Geste via Interrupt des Gestensensors APDS9960	via Bibliothek *smbus*

8.2 Beispielprogramm myBlink.py: LED blinken lassen.

Wie auch bei der Arduinoprgrammierung fängt man beim Physical Computing erst einmal ganz klein an: Man lässt eine LED blinken – d.h. ein GPIO wird ein- und ausgeschaltet.
Der Ausgang P9_25 (GPIO 117) wird mit einer Frequenz von 1 Hz ein- und ausgeschaltet.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

import Adafruit_BBIO.GPIO as GPIO
import time
import sys
pin = "P9_25"
GPIO.setup(pin, GPIO.OUT)
print("Pin enabled...")

try:
    while True:
```

```
        GPIO.output(pin, GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(pin, GPIO.LOW)
        time.sleep(0.5)
except KeyboardInterrupt:
    print("")
    GPIO.output(pin, GPIO.LOW)
    print("...Pin disabled.")
    print("Byebye…")
```

Im Quellcode oben wurde die Adrafruit_BBIO-Bibliothek verwendet. Die gleiche Programmfunktion kann aber auch über direktes Schreiben auf das SysFS erreicht werden, siehe Quellcodes *myBlink_SysFS_OS.py* oder *myBlink_SysFS.py*.

8.3 Beispielprogramm myGPIOread.py: GPIO auslesen

Hier wird ein direktes Schreiben auf das SysFS verwendet um den Status des GPIO-Pins P9_27 auszulesen.

```
import time
import sys
setup = open('/sys/class/gpio/gpio115/direction', 'w')
setup.write('in')
setup.close()
time.sleep(1)
print('GPIO 115 als Eingang gesetzt')
gpio = open('/sys/class/gpio/gpio115/value', 'r')

try:
    while True:
        value=gpio.read().strip()
        print('GPIO-Zustand: {}'.format(value))
        gpio.seek(0)
        time.sleep(1)
except KeyboardInterrupt:
    print("")
    print("Byebye...")
    gpio.close()
```

8.4 Beispielprogramm myADC.py: Analogspannung auslesen (ADC)

An dem Steckplatz „Analog Input“ des Grove Capes wird ein Potentiometer angeschlossen. Dieses wird mit VCC 3,3 V und GND spannungsversorgt. Sein Mittenabgriff liegt auf Eingang AIN0 (P9_39).

Es wird wieder die Bibliothek *Adafruit_BBIO* verwendet. Das Skript *myADC.py* liest die Spannung am BB Pin P9_39 jede Sekunde aus, bis das Programm mit *Strg + c* aus der Linuxkonsole heraus beendet wird.

```
import sys
import Adafruit_BBIO.ADC as ADC
import time
ADC.setup()

try:
    while True:
        #read returns values 0-1.0
        valueNorm = ADC.read("P9_39")
        #read_raw returns non-normalized value
        valueRaw = ADC.read_raw("P9_39")
        time.sleep(1)
        print("Normierter Wert: {}".format(valueNorm))
        print("Rohwert: {}".format(valueRaw))
except KeyboardInterrupt:
    print("")
    print("Byebye...")
```

Der vom AD-Wandler gemessene Wert wird in LSB (12 Bit bezogen auf U_{Ref} = 1,8 V) bzw. auf 1 normiert ausgegeben. Am Grove Cape wird unabhängig von Stellung des VCC-Schalters die Eingangsspannung am

Grove Stecker um den Faktor 1,8/5 reduziert. Somit entsprechen 3,3 V ca. 2700 LSB.

Auch hier gibt es eine Version *myADC_SysFS.py* die die ADC-Werte direkt über das SysFS ausliest.

8.5 Beispielprogramm mySRF02_smbus.py: Sensordaten via I²C von einem SRF02-Ultraschallsensor „zu Fuß“ auslesen.

Dieses Skript *mySRF02_smbus.py* verwendet für die I²C Kommunikation nicht die Bibliothek *Adafruit_BBIO*, denn unter Python 3 funktioniert diese Bibliothek für die I²C-Kommunikation nicht fehlerfrei. Stattdessen wird die Pythonbibliothek *smbus* verwendet. Der Ultraschallsensor ist am I²C-Bus 2 angeschlossen (P9_19 SCL, P9_20 SDA).

```
import sys
import smbus
import time
i2c = smbus.SMBus(2)
print('i2c-bus opened...')

try:
    while True:
        i2c.write_byte_data(0x70, 0, 0x51)
        time.sleep(0.2)
        wert = i2c.read_word_data(0x70, 0x02)
        wert = ((wert << 8) & 0xFF00) + (wert >> 8)
        print(wert)
        time.sleep(1)
except KeyboardInterrupt:
    print('')
    i2c.close()
    print('...i2c-bus closed.')
    print('Byebye…')
```

Tipp:
Falls die Kommunikation nicht funktioniert, dann mit dem Tool *i2c-tool* überprüfen, ob und unter welcher Adresse und Busnummer der Sensor über den I²C Bus erreichbar ist. Mit dem Python-Skript *SRF02AddrChange.py* kann die I²C-Adresse des Sensors geändert werden. *Die neue Adresse muss dabei unbedingt innerhalb des gültigen Adressbereichs liegen ansonsten ist der Sensor anschließend nicht mehr zugänglich = Elektroschrott*.

Machen Sie sich auch mit den Messeigenschaften des Ultraschallsensors vertraut: Dieser liefert z.B. Fehlmessungen bei Reichweiten ca. <100 mm oder beim Antasten glatter ebener Flächen, die den Ultraschall spiegelnd reflektieren.

8.6 Beispielprogramm myPWM.py: PWM-Signal erzeugen

Für das Erzeugen eines PWM-Signals kann man entweder die Bibliothek *Adafruit_BBIO* verwenden oder direkt auf das SysFS schreiben. Im nachfolgendem Codebeispiel wurde die Bibliothek verwendet. Das Codebeispiel *myPWM_SysFS.py* verwendet keine Bibliothek.

```
import Adafruit_BBIO.PWM as PWM
import sys
import time

pin = "P9_14"
FREQ = 50
MINDUTYC = 0.03
MAXDUTYC = 0.12
POL = 0
dutyC = MINDUTYC
PWM.start(pin, ((MINDUTYC+MAXDUTYC)/2*100), FREQ, 0)

try:
    while True:
        PWM.set_duty_cycle(pin, dutyC*100)
        time.sleep(1)
        dutyC = dutyC + 0.01
```

```
        if (dutyC > MAXDUTYC):
            dutyC = MINDUTYC
#Falls Strg+c gedrueckt wird
except KeyboardInterrupt:
    PWM.cleanup()
    print("")
    print("Byebye...")
```

8.7 Beispielprogramm myMot.py: Ansteuerung von DC-Motoren über einen H-Bücken-Treiber

Dieses Programm ist für dem RoboToGo-Roboter gedacht. Über dessen Motortreiberplatine TB6612 werden die beiden DC-Motoren getrennt jeweils über ein PWM-Signal mit 250 kHz angesteuert. Zwei weitere GPIOs als Schaltausgänge steuern die (gemeinsame) Drehrichtung der Motoren. Der Motortreiber muss über VDD_5V also über die Powerbank und nicht via USB mit Strom versorgt werden, da hier anders als beispielsweise bei einem kleinen Modellbauservo höhere Ströme fließen. Zur Entkopplung der stark schwankenden Last durch den Motortreiber sollte an dessen Spannungseingang ein Keramikkodensator mit ca. 1 mF geschaltet werden.

Tipp:
Falls die Ansteuerung der Motoren nicht funktioniert, dann überprüfen ob beim TB6612 der Stand By Eingang auf Low liegt (nur manche Break Out Platinen haben einen Pull Down) und ob die PWM-Frequenz für die gewählte Break Out Platine nicht zu hoch ist.

```
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.GPIO as GPIO
import sys
import time

pin_motL = "P9_14" # GPIO 50 PWMA linker Motor
pin_motR = "P9_16" # GPIO 51 PWMB rechter Motor
pin_in1 = "P9_25" # GPIO 117
pin_in2 = "P9_27" # GPIO 115
FREQ = 250000
DUTYC = 0.6

PWM.start(pin_motL, (DUTYC*100), FREQ, 0)
PWM.start(pin_motR, (DUTYC*100), FREQ, 0)
GPIO.setup(pin_in1, GPIO.OUT)
GPIO.setup(pin_in2, GPIO.OUT)

try:
    while True:
        GPIO.output(pin_in1, GPIO.HIGH)
        GPIO.output(pin_in2, GPIO.LOW)
        PWM.set_duty_cycle(pin_motL, DUTYC*100)
        PWM.set_duty_cycle(pin_motR, DUTYC*100)
        time.sleep(2)
        PWM.set_duty_cycle(pin_motL, 0)
        PWM.set_duty_cycle(pin_motR, 0)
        time.sleep(0.5)
        GPIO.output(pin_in1, GPIO.LOW)
        GPIO.output(pin_in2, GPIO.HIGH)
        PWM.set_duty_cycle(pin_motL, DUTYC*100)
        PWM.set_duty_cycle(pin_motR, DUTYC*100)
        time.sleep(2)
        PWM.set_duty_cycle(pin_motL, 0)
        PWM.set_duty_cycle(pin_motR, 0)
        time.sleep(0.5)
except KeyboardInterrupt:
    PWM.cleanup()
```

8.8 Beispielprogramm myUART.py: UART-Kommunikation (mit Arduino)

Als Kommunikationspartner dient hierfür ein Arduino mit der in Abschnitt 12.11 dargestellten Firmware

UARTComm_BB_Ardu.ino.

Konkret in diesem Beispiel verwendet der BB die UART2-Schnittstelle, welche im Pythonprogramm an Anfang mit dem Tool *config-pin* und den Befehlen *config-pin -a p9.21 uart* bzw. *config-pin -a p9.22 uart* konfiguriert wird.

```
from subprocess import call
import serial
import sys
from time import sleep

command = "config-pin -a p9.21 uart"
call(command, shell= True) # execute command in Bash Shell
command = "config-pin -a p9.22 uart"
call(command, shell= True)

ziffer = 0
print('Vorher Arduino Reset drücken - Abbruch mit strg + c')
print('...zwei Sekunden warten...')
sleep(2)

try:
    port = serial.Serial(port='/dev/ttyO2', baudrate = 115200, timeout = 1)
    sleep(1)
    print('Arduino sagt: ', port.readline().decode("utf-8"))
    sleep(0.5)
    port.readline()
    port.flushInput()
    print('Messungen werden vom Arduino angefordert...')
    while True:
        port.write(str(ziffer).encode())
        result = port.readline()
        print('Arduino sagt: {}'.format(str(result)))
        sleep(1)
        ziffer = (ziffer +1)%10
except KeyboardInterrupt:
    print()
    print('Programm mit strg + c abgebrochen bzw. Laufzeitfehler')
finally:
    port.close()
    print('Schnittstelle geschlossen')
```

9 Einbinden von Sensoren und Aktoren über C/C++ Programme

Python ist eine sehr gute Programmiersprache, um im Rapid Prototyping eine Idee zu testen. Hat sich die Idee bewährt, dann wird der Pythonquellcode oft in C umgeschrieben. Mit C und seiner objektorientierten Variante C++ können Programme systemnah geschrieben werden. Die Programme haben bei C einen direkten Zugriff auf die Hardware, die Firmware, den Kernel und auf das Betriebssystem.

Anders als Python wird ein C-Programm vom Compiler in Maschinensprache übersetzt. Solche "nativen" Programme haben eine um Größenordnungen höhere Ausführungsgeschwindigkeit als Python-Skripte.

Gerade auf dem BB mit nur einem Prozessorkern bei 1 GHz Taktfrequenz und mit nur 1 GB RAM Arbeitsspeicher ist dies ein wichtigeres Thema als bei einem aktuellen Smartphone mit 8 x 2 GHz und 4 GB.

Auf jedem Linuxsystem ist ein C-Compiler (*gcc*) präsent. Im PAT wird gcc für C-Code verwendet. Für C++-Code wird der ebenfalls schon vorhandene Compiler *g++* verwendet.

Einen Quellcode *meinProgramm.c* wird mit *gcc meinProgramm.c -o meinProgramm* kompiliert. Die Option *-o* bewirkt, dass das ausführbare Programm den Namen *meinProgramm* erhält. Ohne diese Option erhält es den Namen *a.out*.
Aufgerufen wird das Programm dann mit *./meinProgramm*.

Mit dem Compiler *g++* wird in gleicher Weise ein ausführbares Programm erzeugt.

Folgende Beispielprogramme befinden sich auf dem Image des PAT bzw. im GitHub Repository:

Programmname C	**Funktion**	**Bemerkung**
myBlink.c	Toggeln des GPIO 117 (P9_25)	via SysFS, byteweises Schreiben
myGPIOread.c	Auslesen GPIO 115 (P9_27)	via SysFS, byteweises Lesen
myGPIOset.c	Setzen GPIO 117 (P9_25)	via SysFS, blockweises Schreiben
myADC.c, myADC.cpp	Auslesen des AIN0 (P9_39)	via SysFS, blockweises Lesen
mySRF02.c	Auslesen Abstandswert in cm Ultraschallsensor via I²C-Bus 2	Ohne SMBUS
myPWM.c	PWM-Signal auf GPIO 50 (P9_14)	via SysFS, blockweises Schreiben, für Modellbauservo gedacht
myUART.c	Serielle Kommunikation mit Arduino (Firmware *UARTComm_BB_Ardu.ino*) über UART2-Schnittstelle (RX: P9_22, TX: P9_21)	via Bibliothek termios

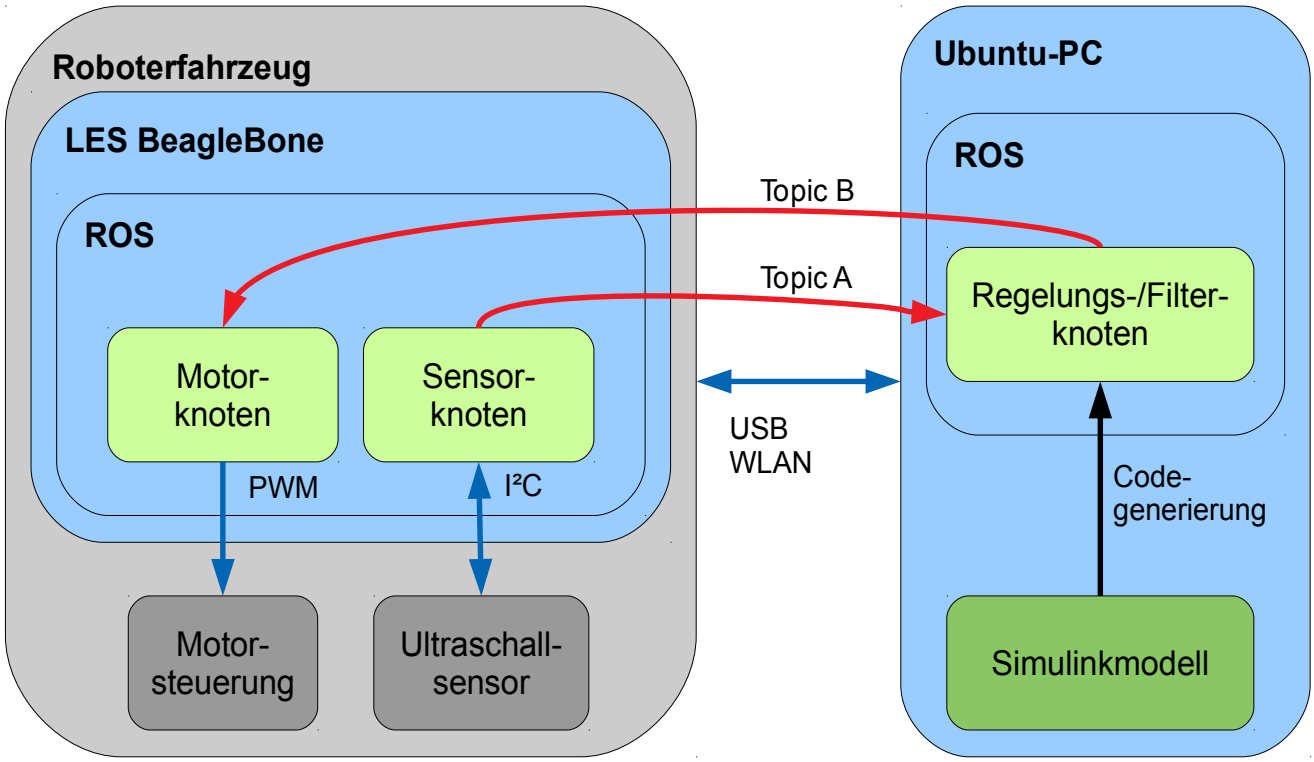
10 Mit ROS-Knoten arbeiten und selbst ROS-Knoten erstellen

Dieses Kapitel behandelt anhand von Beispielen den Umgang mit ROS zuerst jeweils nur innerhalb des BB bzw. des PC. Für die PC-Plattform wird zusätzlich dargestellt, wie dort ein ROS-Knoten über ein Simulinkmodell erstellt wird.

In den letzten Abschnitten wird gezeigt, wie ROS-Knoten verteilt auf dem BB und PC kommunizieren und damit softwareseitig ein mechatronisches System abbilden.

Dies ist letztendlich das Ziel des PAT: Das Roboterfahrzeug als modularisiertes mechatronisches System besitzt die Komponenten *Sensordatenerfassung, Motorsteuerung* und *Kalman-Filter / Regelung / Sensordatenfusion*. Jeder dieser Komponenten (=Module) ist ein ROS-Knoten (=Modul) also ein Prozess zugeordnet.
Dabei werden die Knoten Sensordatenerfassung und Motorsteuerung auf dem BB ausgeführt. Die Knoten Kalman-Filter / Regelung /Sensordatenfusion werden (vorzugsweise mit Simulink) auf dem PC erstellt und dort auch ausgeführt.

Wie in der Grafik dargestellt liefert der Sensorknoten Abstandsmesswerte (Topic A) an den Filter-/Regelungsknoten. Dieser wiederum liefert Solldrehzahlen der Räder (Topic B) an die Motorsteuerung. Hier ist exemplarisch nur der Ultraschallsensor dargestellt.



Achtung:
Die meisten Probleme beim Umgang mit ROS im PAT werden durch falsch gesetzte IP-Adressen verursacht:

Es gibt generell nur einen ROS-Master.

Den verschiedenen anderen am ROS-Netzwerk beteiligten Rechnern („Hosts“) muss die IP-Adresse des Masters bekannt sein. Die Rechner haben jeweils eine eigene IP-Adresse, die dem ROS auf dem jeweiligen Rechner auch bekannt sein muss.
Für die Bekanntgabe dieser IP-Adressen müssen auf dem beteiligten Rechnern jeweils die Umgebungsvariablen `ROS_MASTER_URI` und `ROS_HOSTNAME` gesetzt sein, erstere für die Adresse des Masters und letztere für die Adresse des jeweiligen Host-Rechners. Das Setzen der Umgebungsvariablen geschieht über das Linux Kommando `export`.

Ist beispielsweise der BB via USB an den PC angeschlossen, dann hat der BB die IP-Adresse 192.168.6.2 und der PC 192.168.6.1. Der ROS-Master soll in diesem Beispiel auf dem PC laufen.

Nötige *export*-Kommandos auf dem BB:

export ROS_MASTER_URI=http://192.168.6.1:11311
*(*alternativ *export ROS_MASTER_URI=http://RechnerName.local:11311*)

export ROS_HOSTNAME=192.168.6.2
(alternativ export ROS_HOSTNAME=beaglebone.local)

Nötige *export*-Kommandos auf dem PC:

export ROS_MASTER_URI=http://192.168.6.1:11311
(alternativ export ROS_MASTER_URI=http://RechnerName.local:11311)

export ROS_HOSTNAME=192.168.6.1
(alternativ export ROS_HOSTNAME=RechnerName.local)

Die IP-Adresse des Rechners kann alternativ auch mit über dessen Rechnernamen in der Form *RechnerName.local* angegeben werden. Im Fall des BB lautet der Rechnername *beaglebone*.
Ob dies generell funktioniert hängt vom verwendeten Router ab und sollte vorher mit dem Kommando *ping RechnerName.local* getestet werden.

Durch die Datei *~/.bashrc* auf dem BB bzw. auf dem PC werden diese Kommandos automatisch ausgeführt, wenn ein neues Terminalfenster geöffnet wird.

Gibt ROS eine Fehlermeldung aus, eine URI oder IP-Adresse wäre nicht erreichbar, dann liegt dies meistens daran, dass in der Datei *~/.bashrc* die o.g. Umgebungsvariablen falsch gesetzt sind oder der Router die *.local* Adresse nicht auflösen kann.
Wurde eine Adresse in *~/.bashrc* geändert, muss zuerst das Kommando *source ~/.bashrc* ausgeführt werden, damit die Umgebungsvariablen vom Betriebssystem entsprechend aktualisiert werden.

Manchmal blockiert eine Firewall zwar nicht eine IP-Adresse sondern nur den Zugriff einiger Ports davon. Dies äußert sich beispielsweise so, dass mit *rostopic list* der Topic angezeigt wird, jedoch nicht mit *rostopic echo…* ausgegeben werden kann.
Dann ist es hilfreich wie in *wiki.ros.org/ROS/NetworkSetup* beschrieben die Kommunikation Schritt für Schritt zu testen. Siehe hierzu auch [29].

10.1 ROS und ROS-Knoten ausschließlich auf dem PC

Um sich mit ROS (und Linux) vertraut zu machen, ist es am einfachsten mit dem PC zu beginnen. Denn die *ros.org* Internetseiten liefern ein sehr gutes Tutorial für den Einstieg [30].
Das Lernziel hierbei ist es, einen Publisher- und eine Subscriber-Knoten selbst coden und ausführen zu können. Es wird empfohlen dies in der Programmiersprache Python zu machen. Liegen ausreichende Programmierkenntnisse in C vor, dann können die Knoten auch in dieser Sprache geschrieben werden. Das *ros.org* Tutorial behandelt sowohl Python als auch C/C++.

Für die Einarbeitung in ROS sind folgende Abschnitte des ros.org Tutorials empfehlenswert:

1. Navigating the ROS Filesystem (catkin)
2. Creating a ROS Package (catkin)
3. Building a ROS Package (catkin)
4. Understanding ROS Nodes
5. Understanding ROS Topics
6. Understanding ROS Services and Parameters
7. Writing a Simple Publisher and Subscriber (Python) (catkin)
8. Examining the Simple Publisher and Subscriber

10.2 ROS und ROS-Knoten ausschließlich auf dem BeagleBone

Auf dem BB-Image des PAT sind die nachfolgenden Schritte schon ausgeführt worden. Hier kann direkt auf das ROS-Package *srp-init* zugegriffen werden.

Es werden vier Terminalfenster auf dem PC benötigt: Dazu mit *strg + alt + t* den Terminalmultiplexer Terminator öffnen und anschließend mit *strg + shift + o* bzw. *strg + shift + e* das Fenster horizontal bzw. vertikal teilen. In jedem der vier Terminalfenster *./bbcon_usb.sh* ausführen und damit eine SSH-Verbindung via USB mit dem BB herstellen.

Aus dem Package *pat_intro* können nun die verschiedenen Knoten gestartet werden:

Knotennamen	**Publisher von**	**Subscriber von**
adc_talker.py	ADC-Wert Pin 9_39 in LSB 12 Bit */adc_val UInt16*	
gpio_listener.py		*/range_val UInt16*, setzt Pin P9_25 auf High falls Wert < 30
srf02_talker.py	Abstandsmesswert Ultraschallsensor SRF02 an I²C Bus 2 in cm */range_val UInt16*	
gpio_talker.py	Eingangswert GPIO Pin P9_27 */gpio_val UInt8*	
servo_listener.py		*/range_val UInt16* Im Bereich 0..100 cm Stellung Servoposition via P9_14 proportional zum Abstandsmesswert
motor_listener.py		*/range_val UInt16* Im Bereich <30 cm wird der Duty Cycle des PWM-Signals um 20 % reduziert.
simulink_listener_talker.py	*/volt_tresh*	*/adc_val_mv*

10.2.1 Beispiel ROS-Netzwerk in getrennen Terminalfenstern

Zunächst wird gezeigt, wie der ROS-Master und jeder Knoten einzeln in einem jeweils eigenem Terminalfenster gestartet wird:

Knoten *srf02_talker* (Ultraschallsensor) ausführen und dessen Topic */range_val* ausgeben. Danach zweiten Knoten *gpio_listener* (GPIO steuern) ausführen, damit LED an P9_25 aufleuchtet falls Abstand < 30 cm ist.

Terminal 1: *roscore # ROS-Master starten*

Terminal 2: *rosrun pat_intro srf02_talker.py #Knoten starten*

Terminal 3: *rostopic list #Vorhandene Topics auflisten*

rostopic echo /range_val #Topic /range_val ausgeben

strg + c #Topicausgabe stoppen

rosrun pat_intro gpio_listener.py #Knoten starten

Zum Beenden in jedem Terminalfenster *strg + c* eingeben.

10.2.2 Beispiel ROS-Netzwerk mit *roslaunch* in einem gemeinsamen Terminalfenster via Launchdatei

Besteht das ROS-Netzwerk aus vielen Knoten, dann wird es schnell unübersichtlich, wenn wie im vorangegangenen Abschnitt für jeden Knoten ein gesondertes Terminalfenster geöffnet wird. Mit dem ROS-Tool *roslaunch* können alle benötigte Knoten und auch der ROS-Master mit *einem* Befehl in einem *gemeinsamen* Terminalfenster gestartet werden.

Dazu wird im Paketordner *pat_intro* ein neuer Ordner mit dem Namen *launch* angelegt. Darin wird eine sogenannte Launchdatei – hier die Datei *pat_srf02_gpio.launch* – mit folgendem Inhalt erstellt:

```
<?xml version="1.0" encoding="utf-8" ?>
<launch>
  <node name="mein_srf02_talker" pkg="pat_intro" type="srf02_talker.py" />
  <node name="mein_gpio_listener" pkg="pat_intro" type="gpio_listener.py" />
</launch>
```

Wird diese Launchdatei mit dem Befehl *roslaunch pat_intro srp_srf02_gpio.launch* gestartet, dann werden die drei getrennten Ausführungen des vorherigen Abschnitts (*roscore* und zweimal *rosrun*) in einem Rutsch und im selben Terminalfenster ausgeführt. Das damit erzeugte ROS-Netzwerk ist dasselbe wie im Abschnitt zuvor.

Lediglich haben die beiden Knoten nun noch extra Namen erhalten (*mein_…*). Für den Fall, dass in einer Launchdatei ein Knoten mehrmals benötigt wird, kann so zwischen den erzeugten Knoten unterschieden werden.

Tipp:
Falls die Tab-Vervollständigung nicht funktioniert, sind vielleicht die ROS-Umgebungsvariablen nicht gesetzt. Dies lässt sich mit dem Befehl *source ~/catkin_ws/devel/setup.bash* beheben.

10.3 ROS-Knoten auf PC und BeagleBone, die miteinander kommunizieren

In diesem Beispiel wird der ROS-Master nicht auf dem BB sondern auf dem PC gestartet, damit dort auf die Messwerte (Topic */range_val*) des Ultraschallsensors (Knoten *srf04_talker.py*) zugegriffen werden kann.

BB und PC befinden sich im selben WLAN und kommunizieren darüber. Dafür muss dem BB die IP-Adresse des Masters mitgeteilt werden. Dazu muss **auf dem BB** die Datei *~/.bashrc* entsprechend editiert werden. Der für ROS relevante Inhalt am Ende dieser Datei ist nachfolgend wiedergegeben:

```
source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
export EDITOR='nano -w'

export ROS_MASTER_URI=http://192.168.178.62:11311[10]
export ROS_HOSTNAME=192.168.178.71[11]
```

Der PC hat in diesem Fall im WLAN die IP-Adresse *192.168.178.62*, der BB *192.168.178.71*. Damit kennt das ROS auf dem BB nun den Master.

Da der ROS-Master auf dem PC läuft müssen dort an der Datei *~/.bashrc* normalerweise keine Änderungen vorgenommen werden. Jedoch sollte nachgeschaut werden, ob dort überhaupt diese *export*-Kommandos eingepflegt sind. Falls nicht muss um beim Beispiel oben zu bleiben Folgendes am Ende der Datei eingetragen werden:

```
export ROS_MASTER_URI=http://192.168.178.62:11311
export ROS_HOSTNAME=192.168.178.62
```

Dabei handelt es sich zweimal um die IP-Adresse des PC.

Auf dem PC Terminator mit vier Terminalfenster ausführen. Auf zwei Fenstern mit dem Bash-Skript *./bb-con_wlan.sh* eine SSH-Verbindung mit dem BB via WLAN aufbauen.

Beispiel ROS-Netzwerk:

Knoten *srf04_talker* (Ultraschallsensor) auf dem BB ausführen und dessen Topic */range_val* auf dem PC ausgeben.

Terminal 1 PC: *roscore #ROS-Master starten auf PC*

Terminal 1 BB: *rosrun pat_intro srf02_talker.py #Knoten starten auf BB*

10 Falls Kommunikation via USB: IP-Adresse 192.168.6.1
11 Alternativ beaglebone.local und falls Kommunikation via USB: IP-Adresse 192.168.6.2 bzw. beaglebone.local

Terminal 2 PC: `rostopic list` `#vorhandene Topics auflisten`

`rostopic echo /range_val` `#Topic /range_val ausgeben`

Zum Beenden in jedem Terminalfenster `strg + c` eingeben.

10.4 Simulink ROS-Knoten auf PC erzeugen, die mit ROS-Knoten auf Beagle-Bone kommunizieren

Dieses Beispiel beinhaltet eigentlich das gleiche Vorgehen wie im Abschnitt 10.1, wo mehrere ROS-Knoten auf dem PC gecoded und anschließend zusätzlich zum ROS-Master auf dem PC ausgeführt wurden.

Nun wird jedoch der Knoten nicht durch ein Python- oder C-Programm auf dem PC erzeugt, sondern mit einem PC-Simulinkmodell. Im Hintergrund erzeugt das Simulinkmodell C-Code und aus diesem C-Code wird eine ROS-Knoten erzeugt und ausgeführt. Insofern ist das Vorgehen fast das gleiche wie in im Abschnitt 10.1, wenn dort der Knoten nicht mit Python sondern C erzeugt worden wäre.

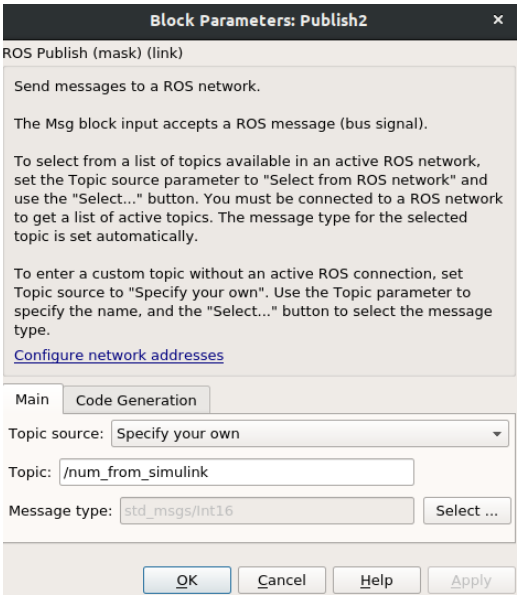
Der Knoten wird also mit Simulink erstellt und ausgeführt. Ein solcher Simulink-ROS-Knoten soll im PAT später für den Kalman-Filter, für die Sensorfusion bzw. für die Regelung verwendet werden.

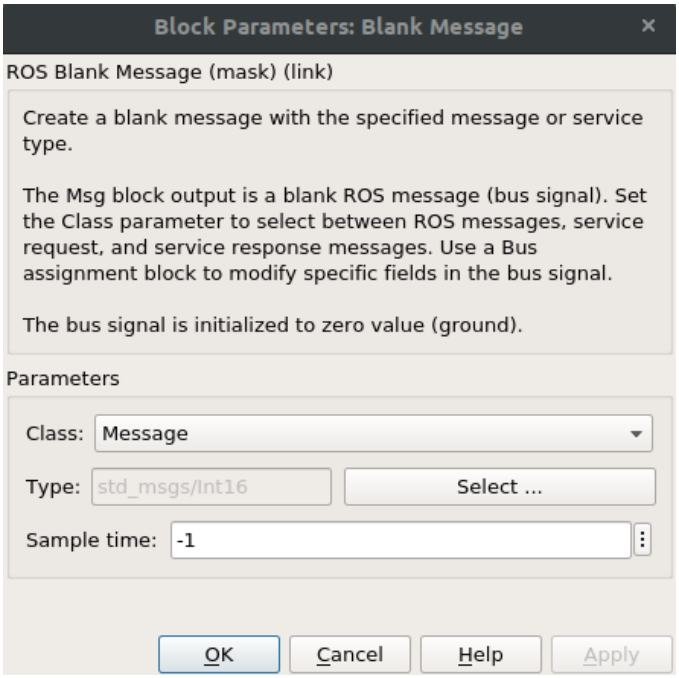
Da der ROS-Master wieder auf dem PC läuft müssen an der Datei `~/.bashrc` des BB die gleichen Änderungen wie im vorangegangenen Abschnitt vorgenommen werden. An der Datei `~/.bashrc` des PC wird wie im vorherigen Beispiel keine Änderung vorgenommen.

Achtung:
Falls Sie über die SSDToGo arbeiten, müssen Sie die MATLAB-Installation auf der SSDToGo zuerst mit Ihrem MATLAB-Konto aktivieren. Dazu folgen Sie den Anweisungen der Datei `readme` auf dem Desktop des Ubutu-Users `master`. MATLAB verlangt als letzten Schritt einen Login-Namen, für den `master` angegeben werden muss.

10.4.1 PC-Simulinkknoten als reiner ROS Publisher

Das Simulinkmodell `count_pub.slx` beinhaltet einen trivialen Publisher-Knoten, der ein Topic mit Namen `/num_from_simulink` mit Datentyp `Int16` ausgibt, welches bei jedem Sample um den Wert 1 erhöht wird bis 255 (8 Bit) erreicht sind und dann erneut von 0 weiter zählt.





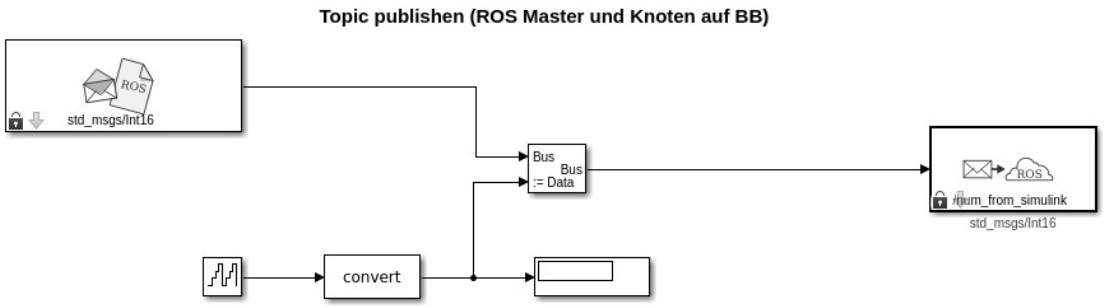
In dem Block „`ROS Blank Message`“ wird von den ROS-Standardmessages der Nachrichtentyp `std_msgs/Int16` ausgewählt.

Im Block „`ROS Publish`“ wird dem Topic der Namen `/num_from_simulink` gegeben und der Nachrichtentyp `std_msgsInt16` zugewiesen.

Unter dem Tab *Simulation* in Simulink wird das Menü über *PREPARE* aufgeklappt:
Dort befindet sich der Menüpunkt *Model Settings*. Hier wird unter *Solver* und *Fixed-Step size (fundamental sample time)* der Wert 0.1 eingegeben. D.h. dieser Knoten veröffentlicht (und inkrementiert) 10 mal pro Sekunde sein Topic.

Im Selben Menü befindet sich auch der Menüpunkt *ROS Network*. Da der ROS-Master und der ROS-Knoten beide auf dem PC laufen, muss hier keine IP-Adresse sondern *Default* eingegeben werden.

Nun wird mit *strg + alt + t* auf dem PC ein Terminal mit zwei Terminalfenstern geöffnet, darin zum Verzeichnis *catkin_ws* gewechselt und der Befehl *source devel/setup.bash* in jedem Terminalfenster ausgeführt[12].



Anschließend wird im Terminalfenster mit *roscore* der ROS-Master gestartet. Im Menüpunkt *ROS Network* von vorhin kann über den Button *Test* abgeprüft werden, ob der eben gestartete ROS-Master von Simulink aus erreichbar ist.

Im Tab *Simulation* kann nun durch Klicken des runden grünen Icons *Run* der ROS-Knoten des Simulinkmodells *count_pub* gestartet werden.

Achtung:
Nach dem Starten des Simulinkmodells erscheint am unteren Fensterrahmen die vergangene Simulationszeit. Falls diese Zeit so schnell voranschreitet, wie man sich das oft bei einer Vorlesung wünscht, dann fehlt im Modell der "Real Time Pacer" Block. Dieser Block sorgt dafür, dass das Simulinkmodell (aka "Simulation") in "Echtzeit" ausgeführt wird. D.h. das Topic mit der "fundamental sample time" von 0,1 s also 10 mal pro Sekunde veröffentlicht und nicht so schnell wie es die Rechenleistung des PCs ermöglicht.

Die Zeit im unteren Fensterrahmen muss also genauso schnell voranschreiten wie die reale Zeit.

Im noch nicht belegten Terminalfenster wird nun mit *rosnode list* nach dem eben gestarteten Simulinkknoten gesucht:

```
rosnode list
/count_pub_xxxxx
/rosout
```

Dabei ist */count_publ_xxxxx* der Name des ROS-Knotens und */count_publ* sein Typ. Die fünfstellige Zahl xxxxx ist bei jedem Starten des Knotens anders, damit bei mehreren Knoten dieses Typs die einzelnen Knoten unterschieden werden können.

Mit dem Befehl *rostopic list* werden die verfügbaren Topics aufgelistet:

```
rostopic list
/num_from_simulink
/rosout
/rosout_agg
```

Mit dem Befehl *rostopic info* wird der Nachrichtentyp und der Namen des Publisher-Knotens angezeigt

```
rostopic info /num_from_simulink
Type: std_msgs/Int16
Publishers:
```

12 Das Laden der ROS-Umgebungsvariablen kann in die Datei *~/.bashrc* in Form des Befehls *source ~/catkin_ws/devel/setup.bash* geschrieben werden. Dann wird dieser Befehl und damit das Laden automatisch beim Öffnen eines neuen Terminalfensters (= Bash Shell) ausgeführt.

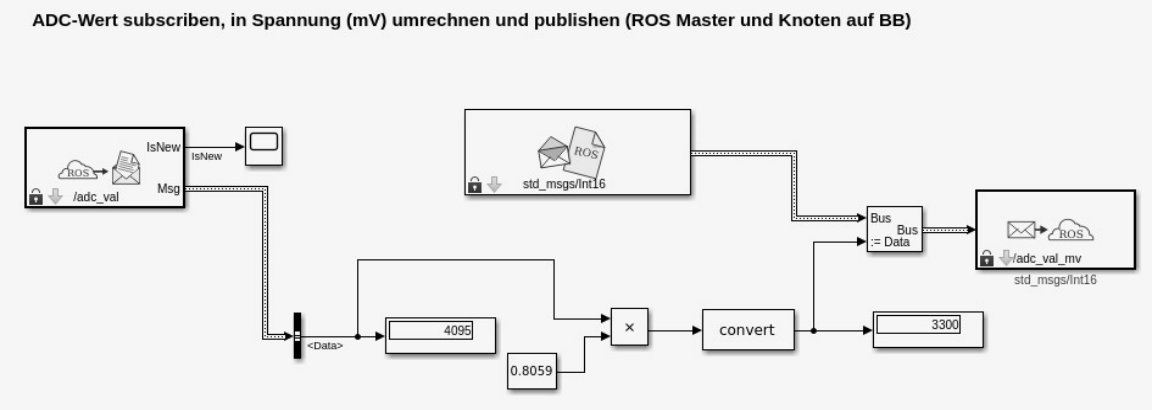
```
* /count_pub_81473 (http://192.168.178.62:37851/)
Subscribers: None
```

Und mit *rostopic echo /num_from_simulink* wird der Wert des Topics angezeigt:

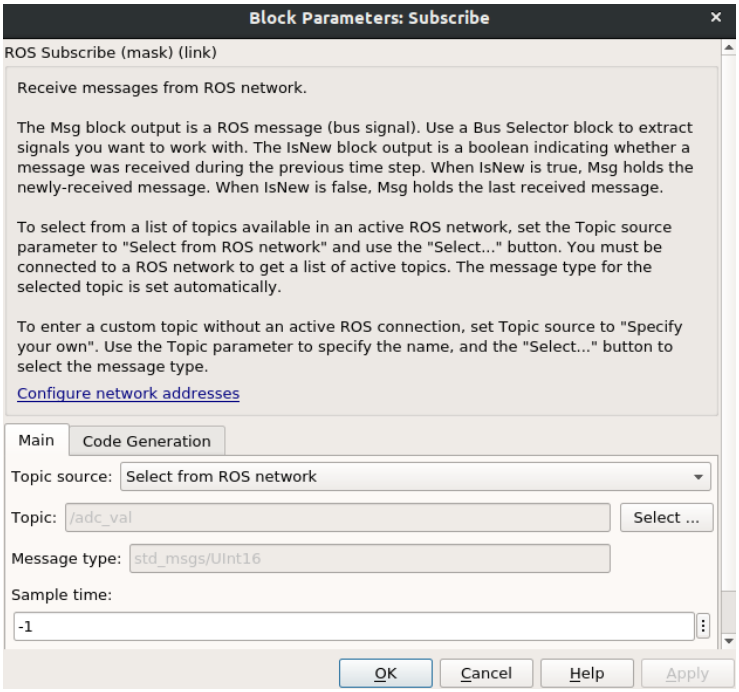
```
rostopic echo /num_from_simulink
data: 63
---
data: 64
---
data: 65
---
data: 66
---
data: 67
```

10.4.2 PC-Simulinkknoten als Publisher-/Subscriber der mit einem Publisher des BB kommuniziert

Der ROS Master läuft auf dem BB. Anders als in den vorangegangenen Beispielen kann hier auch ein Windows-PC verwendet werden, da der ROS-Master auf dem BB ausgeführt wird.



Im Beispiel-Simulinkmodell *adcconv_subpub.slx* wird vom PC-Simulinkknoten der Topic */adc_val* des BB-Knotens *adc_talker* abonniert. Der UInt16-Wert von */adc_val* (ADC-Output als 12-Bit LSB-Wert 0...4095) wird vom Simulinkknoten in Millivolt bezogen auf V_{Ref} = 3,3 V umgerechnet (also mit 3300 mV/4095 = 0,8059 multipliziert, VCC Cape auf 3,3 V) und als Topic */adc_val_mv* veröffentlicht.



Im *ROS Subscribe* Block des Simulinkmodells wird der Topic */adc_val* und dessen Nachrichtentyp *std_msgsUInt16* ausgewählt.

Weiter existiert wie im Abschnitt 10.4.1 ein *ROS Publish Block*, in dem das neue Topic mit dem ADC-Wert in Millivolt */adc_val_mv* mit Nachrichtentyp *std_msgsUInt16* angegeben ist.

Über den Menüpunkt *Simulation > PREPARE > ROS Network* wird unter *ROS Master (ROS)* die IP-Adresse (hier *192.168.178.71*) des BB eingegeben, damit der Simulinkknoten den ROS-Master auf dem BB finden kann.

Auf dem BB muss nun auch sichergestellt sein, dass die richtige IP-Adresse des Masters in der Datei *~/.bashrc* abgelegt ist. Dazu diese mit *nano ~/.bashrc* editieren und in der untersten Zeile zweimal die IP-Adresse des BB (hier *192.168.178.71*) einfügen.

```
export ROS_MASTER_URI=http://192.168.178.71:11311[13]
export ROS_HOSTNAME=192.168.178.71
```

Achtung:
Anschließend müssen mit dem Befehl *source ~/.bashrc* diese neue Umgebungsvariablen für jedes Terminalfenster aktiviert werden.

Danach werden mit *roscore* der ROS-Master und mit *rosrun pat_intro adc_talker.py* der adc_talker-Knoten auf dem BB gestartet.

Nun kann auf dem PC unter Simulink im Menüpunkt *Simulation > PREPARE > ROS Network* abgefragt werden, ob der ROS-Master auf dem BB erreichbar ist. Ist dies der Fall, dann wird schlussendlich im Tab *Simulation* von Simulink der Knoten durch Klicken des runden grünen *Run*-Buttons gestartet.

Im Simulinkmodell werden die Integerwerte und die auf Millivolt umgerechneten Werte des ADC angezeigt. Auf dem BB erscheint nach Eingabe des Befehls *rosnode list* bzw. *rostopic list* der Simulink ROS-Knoten bzw. dessen Topic.

Output *rosnode list*:

```
/ADCIntTalker
/adcconv_subpub_xxxxx
/rosout
```

Output *rostopic list*:

```
/adc_val
/adc_val_mv
/rosout
/rosout_agg
```

Achtung:
Auch hier wird der "Real Time Pacer" Block benötigt, damit das Simulinkmodell (aka "Simulation") in "Echtzeit" ausgeführt wird. Ob dem so ist sehen Sie im unteren Fensterrahmen am Verlauf der Simulationszeit.

Übertragen auf die Aufgaben des PAT wäre */adc_val* der Messwert des Ultraschallsensors (Knoten auf BB). Der Kalman-Filter und die Regelung in Form eines PC-Simulinkknotens berechnen daraus eine Solldrehzahl der Motoren, was */adc_val_mv* entspricht. Der Knoten für die Motorsteuerung (auf dem BB) abonniert diese Solldrehzahl.

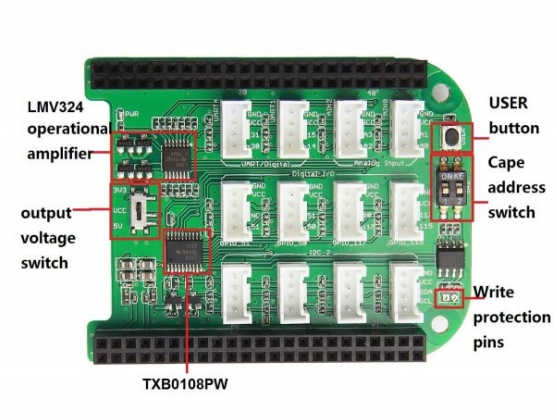
11 Simulation des RoboToGo unter Simulink

Im GitHub Repository zum PAT finden Sie Simulinkmodelle, in denen der RoboToGo simuliert wird. Es ist ratsam zuerst die Regelung und den Kalmanfilter in dieser Simulation zu optimieren. Damit erhalten Sie gute Startwerte für die anschließende Optimierung am realen RoboToGo unter realen Bedingungen.

13 Falls Kommunikation via USB: IP-Adresse 192.168.6.2

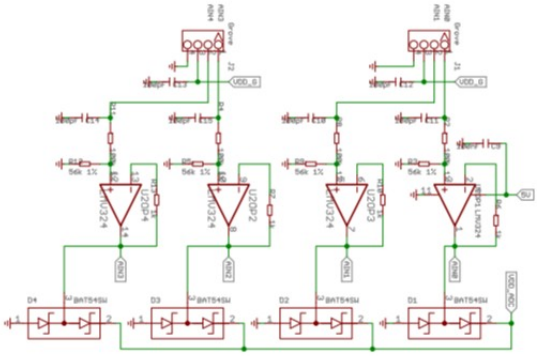
12 Anhang

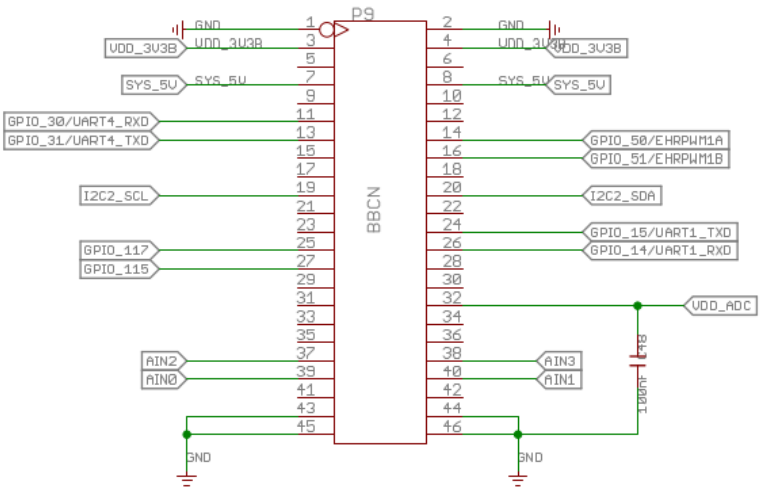
12.1 Das Grove Base Cape Version 2

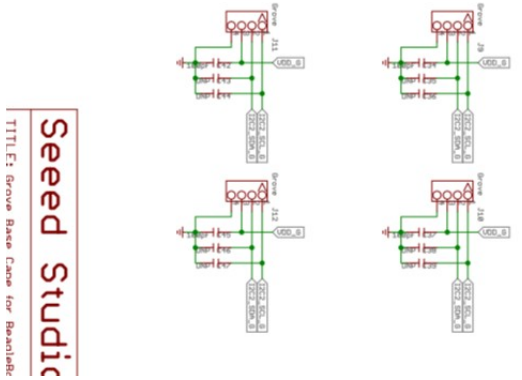


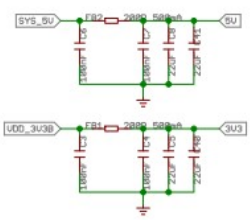
Im PAT wird vorzugsweise diese neue Version 2 verwendet. Die analogen Eingänge werden nicht wie in Version 1 nur über einen einfachen Spannungsteiler sondern zusätzlich über einen Verstärker geführt. Das Cape kann mit VCC 3,3 V und 5 V betrieben werden. Im PAT wird VCC = 5 V verwendet.

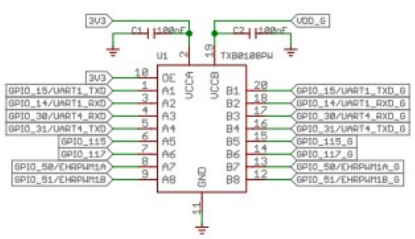
Der Spannungsteiler sowie der Verstärkungsfaktor sind unabhängig von VCC, so dass der ADC immer bei 5 V voll ausgesteuert ist. Nach dem Verstärker begrenzen Klemmdioden die ADC-Eingangsspannung auf 0...1,8 V.
Die digitalen Ein- und Ausgänge laufen anders als bei Version 1 nicht mehr über Klemmdioden sondern über Pegelwandler.
Quelle: [31].

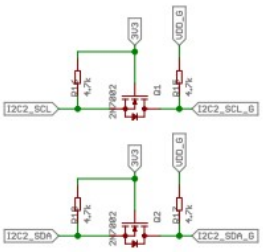


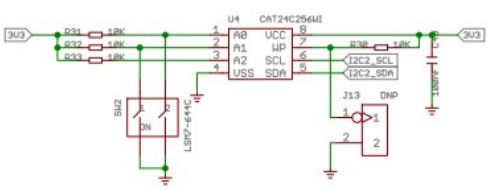












12.2 Nano Text Editor Cheat Sheet

File handling

Ctrl+S	Save current file
Ctrl+O	Offer to write file ("Save as")
Ctrl+R	Insert a file into current one
Ctrl+X	Close buffer, exit from nano

Editing

Ctrl+K	Cut current line into cutbuffer
Alt+6	Copy current line into cutbuffer
Ctrl+U	Paste contents of cutbuffer
Alt+T	Cut until end of buffer
Ctrl+]	Complete current word
Alt+3	Comment/uncomment line/region
Alt+U	Undo last action
Alt+E	Redo last undone action

Search and replace

Ctrl+Q	Start backward search
Ctrl+W	Start forward search
Alt+Q	Find next occurrence backward
Alt+W	Find next occurrence forward
Alt+R	Start a replacing session

Deletion

Ctrl+H	Delete character before cursor
Ctrl+D	Delete character under cursor
Ctrl+Shift+Del	Delete word to the left
Ctrl+Del	Delete word to the right
Alt+Del	Delete current line

Operations

Ctrl+T	Run a spell check
Ctrl+J	Justify paragraph or region
Alt+J	Justify entire buffer
Alt+B	Run a syntax check
Alt+F	Run a formatter/fixer/arranger
Alt+:	Start/stop recording of macro
Alt+;	Replay macro

Moving around

Ctrl+B	One character backward
Ctrl+F	One character forward
Ctrl+←	One word backward
Ctrl+→	One word forward
Ctrl+A	To start of line
Ctrl+E	To end of line
Ctrl+P	One line up
Ctrl+N	One line down
Ctrl+↑	To previous block
Ctrl+↓	To next block
Ctrl+Y	One page up
Ctrl+V	One page down
Alt+\	To top of buffer
Alt+/	To end of buffer

Special movement

Alt+G	Go to specified line
Alt+]	Go to complementary bracket
Alt+↑	Scroll viewport up
Alt+↓	Scroll viewport down
Alt+<	Switch to preceding buffer
Alt+>	Switch to succeeding buffer

Information

Ctrl+C	Report cursor position
Alt+D	Report word/line/char count
Ctrl+G	Display help text

Various

Alt+A	Turn the mark on/off
Tab	Indent marked region
Shift+Tab	Unindent marked region
Alt+N	Turn line numbers on/off
Alt+P	Turn visible whitespace on/off
Alt+V	Enter next keystroke verbatim
Ctrl+L	Refresh the screen
Ctrl+Z	Suspend nano

Quelle: [32]

12.3 Linux Cheat Sheet

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2*
if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

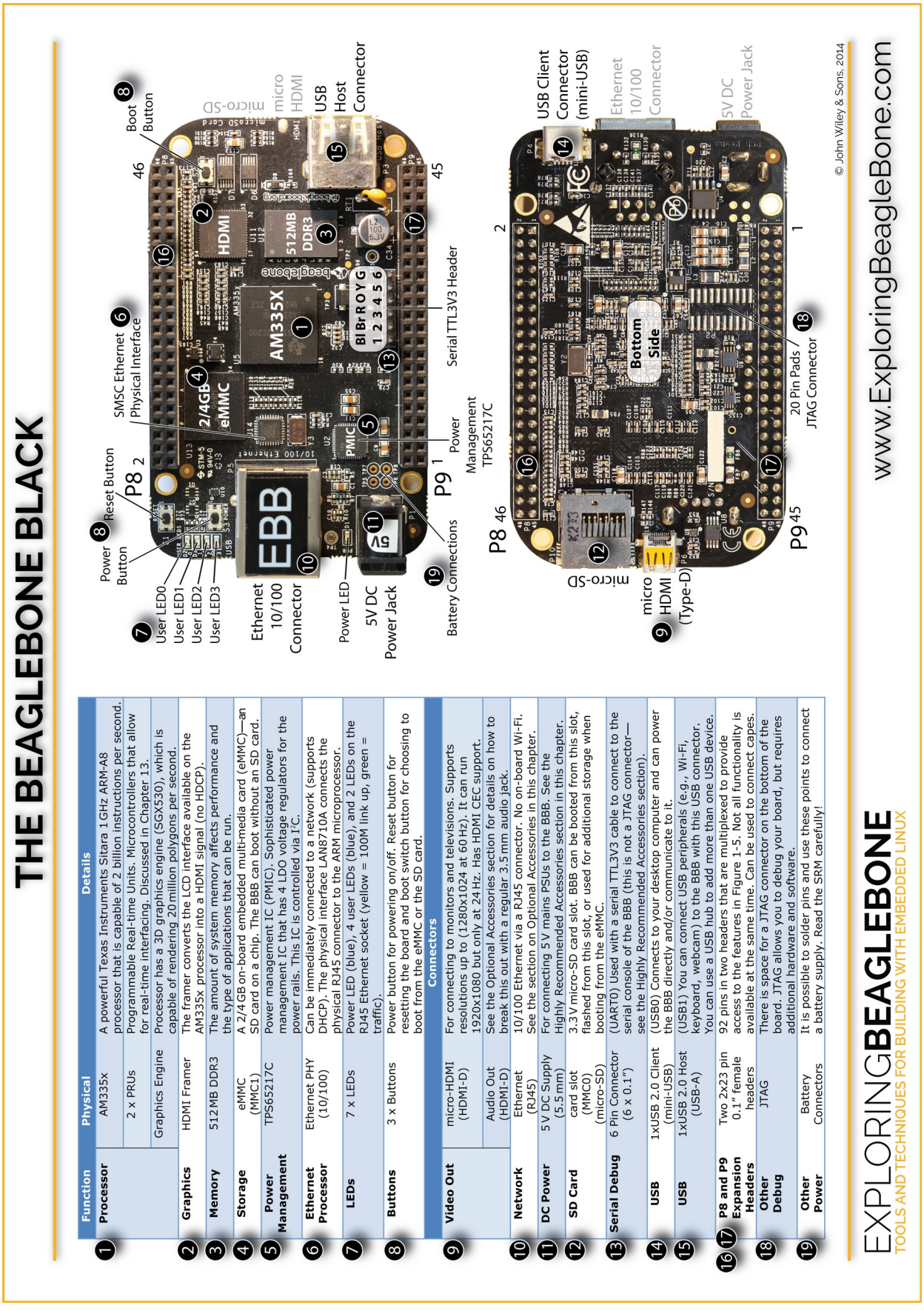
Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

Quelle: [33]

12.4 Übersicht BB Hardware (Quelle [19])



	Function	Physical	Details
1	**Processor**	AM335x	A powerful Texas Instruments Sitara 1 GHz ARM-A8 processor that is capable of 2 billion instructions per second.
		2 x PRUs	Programmable Real-time Units. Microcontrollers that allow for real-time interfacing. Discussed in Chapter 13.
		Graphics Engine	Processor has a 3D graphics engine (SGX530), which is capable of rendering 20 million polygons per second.
2	**Graphics**	HDMI Framer	The framer converts the LCD interface available on the AM335x processor into a HDMI signal (no HDCP).
3	**Memory**	512 MB DDR3	The amount of system memory affects performance and the type of applications that can be run.
4	**Storage**	eMMC (MMC1)	A 2/4 GB on-board embedded multi-media card (eMMC)—an SD card on a chip. The BBB can boot without an SD card.
5	**Power Management**	TPS65217C	Power management IC (PMIC). Sophisticated power management IC that has 4 LDO voltage regulators for the power rails. This IC is controlled via I²C.
6	**Ethernet Processor**	Ethernet PHY (10/100)	Can be immediately connected to a network (supports DHCP). The physical interface LAN8710A connects the physical RJ45 connector to the ARM microprocessor.
7	**LEDs**	7 x LEDs	Power LED (blue), 4 user LEDs (blue), and 2 LEDs on the RJ45 Ethernet socket (yellow = 100M link up, green = traffic).
8	**Buttons**	3 x Buttons	Power button for powering on/off. Reset button for resetting the board and boot switch button for choosing to boot from the eMMC or the SD card.
	Connectors		
9	**Video Out**	micro-HDMI (HDMI-D)	For connecting to monitors and televisions. Supports resolutions up to (1280x1024 at 60 Hz). It can run 1920x1080 but only at 24 Hz. Has HDMI CEC support.
		Audio Out (HDMI-D)	See the Optional Accessories section for details on how to break this out with a regular 3.5 mm audio jack.
10	**Network**	Ethernet (RJ45)	10/100 Ethernet via a RJ45 connector. No on-board Wi-Fi. See the section on Optional Accessories in this chapter.
11	**DC Power**	5V DC Supply (5.5 mm)	For connecting 5V mains PSUs to the BBB. See the Highly Recommended Accessories section in this chapter.
12	**SD Card**	card slot (MMC0) (micro-SD)	3.3V micro-SD card slot. BBB can be booted from this slot, flashed from this slot, or used for additional storage when booting from the eMMC.
13	**Serial Debug**	6 Pin Connector (6 x 0.1")	(UART0) Used with a serial TTL3V3 cable to connect to the serial console of the BBB (this is not a JTAG connector—see the Highly Recommended Accessories section).
14	**USB**	1xUSB 2.0 Client (mini-USB)	(USB0) Connects to your desktop computer and can power the BBB directly and/or communicate to it.
15	**USB**	1xUSB 2.0 Host (USB-A)	(USB1) You can connect USB peripherals (e.g., Wi-Fi, keyboard, webcam) to the BBB with this USB connector. You can use a USB hub to add more than one USB device.
16 17	**P8 and P9 Expansion Headers**	Two 2x23 pin 0.1" female headers	92 pins in two headers that are multiplexed to provide access to the features in Figure 1-5. Not all functionality is available at the same time. Can be used to connect capes.
18	**Other Debug**	JTAG	There is space for a JTAG connector on the bottom of the board. JTAG allows you to debug your board, but requires additional hardware and software.
19	**Other Power**	Battery Connectors	It is possible to solder pins and use these points to connect a battery supply. Read the SRM carefully!

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P8_01		Offset from:		DGND										Ground
P8_02		44e10800		DGND										Ground
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6	R9	Allocated emmc2
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7	T9	Allocated emmc2
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2	R8	Allocated emmc2
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3	T8	Allocated emmc2
P8_07	36	0x890/090	66	TIMER4	gpio2[2]					timer4		gpmc_advn_ale	R7	
P8_08	37	0x894/094	67	TIMER7	gpio2[3]					timer7		gpmc_oen_ren	T7	
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]					timer5		gpmc_be0n_cle	T6	
P8_10	38	0x898/098	68	TIMER6	gpio2[4]					timer6		gpmc_wen	U6	
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13	R12	
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12	T12	
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9	T10	
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]			ehrpwm2_tripzone_in	mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10	T11	
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15	U13	
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14	V13	
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11	U12	
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0	V12	
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8	U10	
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2	V9	Allocated emmc2
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1	U9	Allocated emmc2
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5	V8	Allocated emmc2
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4	U8	Allocated emmc2
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1	V7	Allocated emmc2
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0	U7	Allocated emmc2
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0	V6	
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync	U5	Allocated HDMI
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk	V5	Allocated HDMI
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync	R5	Allocated HDMI
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]	pr1_pru1_pru_r31_11	pr1_pru1_pru_r30_11				gpmc_a11	lcd_ac_bias_en	R6	Allocated HDMI
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14	V4	Allocated HDMI
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15	T5	Allocated HDMI
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13	V3	Allocated HDMI
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11	U4	Allocated HDMI
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12	V2	Allocated HDMI
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10	U3	Allocated HDMI
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8	U1	Allocated HDMI
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9	U2	Allocated HDMI
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6	T3	Allocated HDMI
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7	T4	Allocated HDMI
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4	T1	Allocated HDMI
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5	T2	Allocated HDMI
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2	R3	Allocated HDMI
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3	R4	Allocated HDMI
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0	R1	Allocated HDMI
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1	R2	Allocated HDMI
P9 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 7	Mode 6	Mode 5	Mode 4	Mode 3	Mode 2	Mode 1	Mode 0	CPU	

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P9_01		44e10000		GND										Ground
P9_02		Offset from:		GND										Ground
P9_03		44e10800		DC_3.3V										250mA Max Current
P9_04				DC_3.3V										250mA Max Current
P9_05				VDD_5V										1A Max Current
P9_06				VDD_5V										1A Max Current
P9_07				SYS_5V										250mA Max Current
P9_08				SYS_5V										250mA Max Current
P9_09				PWR_BUT										5V Level (pulled up PMIC)
P9_10				SYS_RESETn								RESET_OUT	A10	
P9_11	28	0x870/070	30	UART4_RXD	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	All GPIOs to 4-6mA output
P9_12	30	0x878/078	60	GPIO1_28	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	31	UART4_TXD	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxer	gpmc_csn5	mii2_rxer	gpmc_wpn	U17	
P9_14	18	0x848/048	50	EHRPWM1A	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	48	GPIO1_16	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	rmii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	51	EHRPWM1B	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_dtsn	D17	Allocated I2C2
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated I2C2
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	15	UART1_TXD	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated mcasp0_pins
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	Allocated mcasp0_pins
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr	C12	Allocated mcasp0_pins
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated mcasp0_pins
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated mcasp0_pins
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated mcasp0_pins
P9_32				VADC										1.8 ADC Volt. Ref.
P9_33				AIN4									C8	1.8V input
P9_34				AGND										Ground for ADC
P9_35				AIN6									A8	1.8V input
P9_36				AIN5									B8	1.8V input
P9_37				AIN2									B7	1.8V input
P9_38				AIN3									A7	1.8V input
P9_39				AIN0									B6	1.8V input
P9_40				AIN1									C7	1.8V input
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both to P21 of P11
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both to P21 of P11
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both to P22 of P11
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Allocated mcasp0_pins
P9_43				GND										- See Pg.50 of the SRM
P9_44				GND										Ground
P9_45				GND										Ground
P9_46		cat		GND										Ground
P9	$PINS	ADDR +	GPIO NO.	Name	Mode 7						Mode 1	Mode 0	CPU	Notes

EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P9 Header

www.ExploringBeagleBone.com

Quelle: [19], siehe auch [34]

12.5.1 Ermitteln der installierten Linuxdistribution

Der BB ist mit einem Ubuntu 20.04 LTS Betriebssystem ausgestattet. Ubuntu ist eine Linuxdistribution. Welche Distribution sich genau auf dem BB befindet, ermittelt der Befehl

`cat /etc/debian_version` (Output *`bullseye/sid`*)

oder *`cat /etc/dogtag`* (Output *`BeagleBoard.org ROS Image 2020-05-18`*).

oder *`lsb_release -a`* mit Output

```
No LSB modules are available.
Distributor ID:   Ubuntu
Description:      Ubuntu 20.04.2 LTS
Release:    20.04
```

```
Codename:   focal
```

Debianversion und Kernelversion sind etwas Verschiedenes. Die Kernelversion wird mit dem Befehl *uname -a* angezeigt. Im Praktikum wird die Kernelversion 4.19.xx verwendet. Hier bezeichnet die erste Zahl die eigentliche „Kernelversion“, welche sich nur alle paar Jahre ändert. Die beiden folgenden Zahlen beziehen sich auf die „Major Revision“ und „Minor Revision“ des Kernels.

12.6 Paketmanagement über apt

Nachfolgend sind die wichtigsten Befehle für das Paketmanagement via *apt-get* aufgeführt. Der Platzhalter *MeinPaket* steht für ein Softwarepaket wie z.B. *nano*.

Paket installieren	*apt-get install MeinPaket*
Index Paketmanager aktualisieren	*apt-get update*
Ist *MeinPaket* installiert?	*dpkg-query -l ‘*MeinPaket*‘*
Ist ein Paket mit *MeinPaket* verfügbar?	*apt-cache search MeinPaket*
Das Paket *MeinPaket* entfernen Wie oben aber mit Konfigurationsdateien	*apt-get remove MeinPaket* *apt-get purge MeinPaket*
Nicht mehr benötigte Pakete (Abhänigkeiten) entfernen	*apt-get autoremove*
Hilfe zu *apt* aufrufen	*apt-get help*

Statt *apt-get* kann in den meisten Fällen auf der Befehl *apt* verwendet werden. *apt-get* und *apt* sind lediglich leicht unterschiedliche Frontends der Paketverwaltung.

12.7 Verwendete Abkürzungen

PAT Sensor- und Regelungssystemepraktikum

BB BeagleBone

BBB BeagleBone Black

BBG BeagleBone Green

µC Mikrocontroller

LES Linux Embedded System

PRU Programmable Real-Time Unit

SSH Sichere Kommunikation über Ethernet in einem Terminalfenster. SSH ist ein Protokoll.

IDE Entwicklungsumgebung für das Programmieren (z.B. `Cloud9`, Eclipse, Arduino und im weiteren Sinne auch MATLAB/Simulink)

DHCP Dynamic Host Configuration Protocol

APT Advanced Packaging Tool für das Paketmanagement unter Linux, Aufruf mit *apt-get*

GPIO General Purpose Input Output, Schnittstellen zur I/O-Periferie wie Schaltein-/ausgänge, PWM-Ausgänge oder digitale Busse.

DTO Device Tree Overlay (Pinkonfiguration zur Laufzeit)

I/Os Ein-/Ausgabeschnittstellen

DT Device Tree (Pinkonfiguration beim Bootvorgang)

BASH Bourne Again Shell. Das auf Linuxsystemen am häufigsten verwendete Shellprogramm, welches Befehle in einem Terminalfenster entgegennimmt und interpretiert.

12.7.1 Installation des Tools minicom für die UART (serielle) Kommunikation im Terminalfenster

Aus einem Terminalfenster lässt sich die UART-Verbindung mit dem Tool *minicom* unkompliziert testen. Es wird installiert via *sudo apt-get install minicom*.

12.7.2 Installation des Pythonpakets pySerial für die UART (serielle) Kommunikation

Pythonpakete sollten soweit in den Ubunt-Paketquellen vorhanden mit *apt-get* installiert werden, um Konflikte mit der Python eigenen Paketverwaltung *pip3* zu vermeiden [35]. Das Paket *pySerial* für Python 3 ist in den Ubuntu-Paketquellen vorhanden. Somit wird es mit folgendem Befehl installiert:

```
sudo apt-get install python3-serial
```

12.8 Kommunikationsmöglichkeiten mit dem BB

Zum Programmieren oder für die Fehlersuche beim BB muss man mit ihm via Tastatur und Bildschirm kommunizieren. Dafür gibt es eine Reihe verschiedener Möglichkeiten, die in der nachfolgenden Tabelle zusammengefasst sind.

	Anschluss	**Linux Benutzeroberfläche**	**Terminalfenster (Shell)**	**Internetseiten auf dem BB**	**Cloud9 IDE**	**Internetzugang**	**Bemerkung**
Tastatur und Monitor direkt	USB bzw. HDMI	Direkt	Über Linux Benutzeroberfläche	Über Linux-Browser	Über Linux-Browser	Ja, wenn Verbindung über Netzwerkbuchse	Geht nur bei BBB
Ethernet über USB	Micro/Mini USB-Buchse Unterseite BBG/BBB	Auf dem PC über TCP/IP mit VNC-Software am PC und VNC-Server am BB	Auf dem PC über SSH-Protokoll und Bash Shell direkt am BB	Über den PC-Browser plattfomunabhängig	Über den PC-Browser plattfomun abhängig	Ja, wenn Verbindung über Netzwerkbuchse oder mit Trick	Stromversorgung über USB oder 5 V Buchse bzw. VDD_5V
Ethernet	Netzwerkbuchse	siehe oben	siehe oben	siehe oben	siehe oben	Ja, wenn Internetgateway im selben Netzwerk	Zusätzlich Stromversorgung nötig, PC, BB und Internetgateway im selben Netzwerk
Seriell	Serial Debug Pins auf Oberseite	Nein	Wie oben aber ohne SSH	Nein	Nein	Nein	Einzige Möglichkeit um Bootmeldungen zu empfangen

12.9 Fachbücher zum BB

Buch	**Bemerkung**
BeagleBone Cookbook [3]	Behandelt überwiegend BoneScript in Form von Projektanleitungen, jedoch gutes Kapitel „Real-Time I/Os“ in dem die verschiedenen Möglichkeiten der PRU-Nutzung sowie spezielle C-Bibliotheken vorgestellt werden (z.B. PRU Speak).
BeagleBone For Dummies [4]	Neben BoneScript gute Einführung in Python (mit BBIO Adafruit Bibliothek), ansonsten aber zum größten Teil Makerbuch mit vielen konkreten Projekten aber wenig Tiefe.
30 BeagleBone Black Projects for the Evil Genius [36]	Reines Makerbuch überwiegend basierend auf BoneScript, konkrete Bauanleitungen mit wenig Tiefe.

BeagleBone Home Automation[37]	Verwendet Python und befasst sich mit Server-Client Kommunikation, Android wird ebenfalls berücksichtigt. Jedoch nur wenige Infos bezüglich I/Os.
BeagleBone Robotic Projects [38]	Verwendet Python und Bildverarbeitung (OpenCV über Python). Wenig Tiefe da mehr Makerbuch mit Anleitung für Robotikprojekte.
BeagleBone Black Cookbook [6]	Verwendet BoneScript, C-Programmierung und Python (mit BBIO Adafruit Bibliothek), gutes Kapitel 6 zu RealTime I/Os auch mit PRU (PyPRUSS), SDR-Beispiel. Inhaltlich mit mittlerer Tiefe, insgesamt jedoch recht unstrukturiertes Makerbuch
Programming the BeagleBone [39]	Makerbuch mit BoneScript und Python (mit BBIO Adafruit Bibliothek), ein wenig IoT mit REST-API. Insgesamt aber wenig Inhalt und Tiefe.
Exploring the BeagleBone, Second Edition [1]	***Mit Abstand das umfassendste, am besten strukturierte und umfangreichste Buch zum BB. auf den zugehörigen Internetseiten [2] finden sich viele Videos, Codebeispiele und Aktualisierungen. Dieses Buch ist ein Muss, wenn man sich ernsthaft mit dem BB befasst.***

12.10 Tastenkombinationen für dem Terminal-Multiplexer Terminator

Kombination	Auswirkung
Strg + ⇧ + O	Das Terminal horizontal teilen
Strg + ⇧ + E	Das Terminal vertikal teilen
Strg + ⇧ + S	Die Bildlaufleiste verstecken
Strg + ⇧ + W	Das aktuelle Terminal schließen
Strg + ⇧ + Q	Programm beenden
Strg + ⇧ + T	Einen neuen Reiter öffnen
Strg + ⇧ + N	Zum nächsten Terminal wechseln
Strg + ⇧ + P	Zum vorherigen Terminal wechseln
Strg + ⇧ + X	Vollbildanzeige des aktiven Terminals
F11	Vollbild-Modus
"Drag & Drop"	
Strg + rechte Maustaste	Terminal zum positionieren aufnehmen
Strg loslassen	Position anzeigen lassen
rechte Maustaste loslassen	Terminal an Position positionieren

Quelle siehe [14].

12.11 Arduino-Firmware UARTComm_BB_Ardu.ino zum Testen der UART-Verbindung

Achtung:
Wenn die UART-Verbindung direkt über die Buchsenleisten und nicht über Das Grove Base Cape hergestellt wird, dann muss unbedingt ein Arduino mit 3,3 V Pegel verwendet werden, da sonst der BB durch Überspannung zerstört wird.

Diese Arduino-Firmware sendet nach einem Reset mit 115200 Baud über die UART-Schnittstelle (D0: RX, D1: TX) im Sekundentakt den String "n: Arduino bereit...warte auf Input..." wobei n jeweils um 1 inkrementiert wird.
Sobald der Arduino auf seinem RX-Pin ein Zeichen empfangen hat, leuchtet die LED13 kurz auf, eine ADC-Messung wird durchgeführt und das vorher empfangene Zeichen wird zusammen mit dem Wandlerwert in LSB (10 Bit) auf die Schnittstelle ausgegeben.

```
int adc_val= 0;  // Ausgabewert ADC
int inByte = 0;  // Auf RX empfangenes Byte
```

```
void setup()
{
  pinMode(13,OUTPUT);
  int i=0;
  Serial.begin(115200, SERIAL_8N1);
  while(Serial.available()<=0){   // Warten bis Zeichen auf RX zum Start
    Serial.print(i,DEC);
    Serial.println(": Arduino bereit...warte auf Input...");
    delay(1000);
    i+=1;
  }
}

void loop()
{
  if (Serial.available() > 0) {   // Zeichen empfangen
    inByte = Serial.read();
    Serial.flush();
    adc_val = analogRead(A0);
    Serial.write(inByte);
    Serial.print(",");
    Serial.println(adc_val,DEC);
    digitalWrite(13,HIGH);
    delay(100);
    digitalWrite(13,LOW);
  }
}
```

12.12 Bash-Skript wifi_reset.sh zum Überprüfen und Wiederherstellen der WLAN-Verbindung

```
#!/bin/bash
echo none > /sys/class/leds/beaglebone\:green\:usr3/trigger
iwgetid >/dev/null

if [ $? != 0 ]
then
    connmanctl disable wifi 2>/dev/null
    connmanctl enable wifi
else
   echo timer > /sys/class/leds/beaglebone\:green\:usr3/trigger
fi
```

Um das obige Skript via `Cron` ausführen zu lassen muss es nach `/usr/local/bin` kopiert werden. Dann wird mit dem Befehl `sudo nano etc/crontab` in der Datei `crontab` folgende Zeile eingefügt:

```
*/1 *  * * *  root    bash /usr/local/bin/wifi_reset.sh
```

12.13 Bash-Skript pin_set.sh zur Konfiguration der Pins beim Booten

```
#!/bin/bash
echo I/O-Pins werden konfiguriert...

# PWM Pin P9_14 und 16
config-pin p9.14 pwm
config-pin p9.16 pwm
echo 0 > /sys/class/pwm/pwmchip4/export
echo 1 > /sys/class/pwm/pwmchip4/export

# UART2 auf P9_11 (RX) und 13 (TX) (Eigentlich fuer SPI vorgesehen)
config-pin p9.11 uart
config-pin p9.13 uart

# UART4 auf P9_21 (TX) und 22 (RX)
#config-pin p9.21 uart
#config-pin p9.22 uart
```

```
# UART1 auf P9_24 (TX) und 26 (RX)
#config-pin p9.24 uart
#config-pin p9.26 uart
```

Dieses Bash-Skript wird durch den Service *pin-set.service* (*/etc/systemd/system*) beim Booten einmalig ausgeführt (*sudo systemctl enable pin-set.service* um diesen Service einzurichten).

Inhalt Textdatei *pin-set.service*:

```
[Unit]
Description=Setup of  BB pins for PAT

[Service]
Type=simple
ExecStart=/bin/bash /home/beagle/bash_ws/pin_set.sh

[Install]
WantedBy=multi-user.target
```

12.14 Bash-Skript zum SSH-Verbindungsaufbau WLAN

```
#!/bin/bash
wifi_ip=beaglebone.local
echo ..........................................................
echo Verbindung wird über WiFi-Adresse $wifi_ip hergestellt
echo ..........................................................
echo
ssh $wifi_ip -l beagle -p 22
```

Im Fall von USB wird statt beaglebone.local die IP-Adresse 192.168.6.2 verwendet.

12.15 Verwendung des Tools config-pin

Mit dem Befehl *config-pin -h* wird die Hilfefunktion aufgerufen und die Befehlssyntax aufgelistet.

12.15.1 Anzeigen des aktuell geladenen Pinkonfiguration

Mit dem Befehl *config-pin -i px.y* erhält man eine allgemeine Information über den Pin Nr. y auf der Buchsenleiste x. Insbesondere werden dabei die verschiedenen Modi (z.B. i2c, can, spi_cs) aufgelistet, auf die der entsprechende Pin mit *config-pin -a px.y ...* konfiguriert werden kann.

```
beagle@beaglebone:~$ config-pin -i p9.16
Pin name: P9_16
Function if no cape loaded: gpio
Function if cape loaded: default gpio gpio_pu gpio_pd gpio_input pwm
Function information: gpio1_19 default gpio1_19 gpio1_19 gpio1_19 gpio1_19 ehrpwm1b
Kernel GPIO id: 51
PRU GPIO id: 83
```

Der Befehl *config-pin -q px.y* liest die aktuelle Pin-Konfiguration aus:

```
beagle@beaglebone:~$ config-pin -q p9.16
P9_16 Mode: pwm
```

Dies funktioniert auch mit anderen als den GPIO-Pins:

```
beagle@beaglebone:~$ config-pin -i p9.35
Pin is not modifiable: P9_35 AIN6
```

12.15.2 Setzen von Pinfunktionen und -werten

Bei einfachen GPIO-Pins kann mit *config-pin* deren Funktion als Eingang *in* oder Ausgang *out* gesetzt werden. Anschließend kann für einen Eingang ein Pull-Up-Widerstand mit *in+* oder oder ein Pull-Down-Widerstand mit *hi-* gesetzt werden. Mit dem Befehlt *config-pin -q* wird dann der Eingangswert des Pins ausgegeben (hier als Beispiel einmal Pin 9_11 mit Pull-Down auf GND bzw. mit Pull-Up auf 3,3 V).

```
beagle@beaglebone:~$ config-pin p9.11 in
beagle@beaglebone:~$ config-pin p9.11 in+
beagle@beaglebone:~$ config-pin -q p9.11
P8_11 Mode: gpio_pu Direction: in Value: 0
```

```
beagle@beaglebone:~$ config-pin -q p9.11
P8_11 Mode: gpio_pu Direction: in Value: 1
```

Ein GPIO als Ausgang kann mit den Werten 0 und 1 auf High oder Low gesetzt werden.

```
beagle@beaglebone:~$ config-pin p9.26 out
beagle@beaglebone:~$ config-pin p9.26 1
beagle@beaglebone:~$ config-pin p9.26 0
```

Funktionalitäten wie PWM oder I²C können mit *config-pin* den dafür in Frage kommenden Pins zugewiesen werden. Im nachfolgenden Beispiel wird dem Pin 9_22 die PWM-Funktion zugewiesen.

```
beagle@beaglebone:~$ config-pin -q p9.22
P9_22 Mode: default Direction: in Value: 1
beagle@beaglebone:~$ config-pin -a p9.22 pwm
beagle@beaglebone:~$ config-pin -q p9.22
P9_22 Mode: pwm
```

12.16 Stückliste RoboToGo

Bezeichnung	**Stück-zahl**	**Bemerkung**
BeagleBone Green	1	Version ohne integriertes WLAN
MicroSDHC-Speicherkarte 16GB - SanDisk Ultra	1	
Beaglebone Seed Grove Cape V2	1	
Grove 4-Pin Female Jumper zu Grove 4-Pin	2	5er Pack
Grove Kabel 4-Pin Male auf Jumper	2	5er Pack
Pololu Ball Caster with 1/2" Plastic Ball #952	1	
Steckboard, Breadboard 400 (300/100) transparent	1	
Pololu TB6612FNG Dual Motor Driver Carrier	1	
Adafruit Proto Plate for Beagle Bone & Beagle Bone Black	1	Alternativlieferant mouser.de
SRF02 Ultraschallsensor	1	Alternativlieferant antratek.de
Pololu VL53L1X TOF-Sensor	1	
Getriebemotor mit Rad, 3 .. 9 V DC	2	Silikonbänder für besseren Grip
Ansmann Powerbank, Li-Ion, 6600 mAh (alternativ 8800 mAh)	1	
USB A Stecker offene Kabelenden GND / VDD_5V	1	Werden an Cape GND und VDD_5V gelötet
Edimax EW-7811UN WLAN-Adapter, USB, 150 MBit/s	1	
Kabelbinder, 280 mm	1	100er Pack, 2 Stück pro Roboter
Gummifüße, selbstklebend, 10 x 10 mm	4	120er Pack
Klingeldraht, D0,6 mm 2x20 cm verdrillt	2	10 m Pack, alternativ Conrad Schaltdraht YV D0,5 mm Blau / Gelb je 100 m, Nr. 606081
Transcend ESD240C Portable SSD 120GB, USB-C 3.1 (TS120GESD240C)	1	Inkl. zwei Anschlusskabel (USB, und USB-C), siehe unten
USB-A auf USB-C Kabel	1	
USB-C auf USB-C Kabel	1	
Steckbrücken M-M Blau	2	
Steckbrücken M-M Rot	1	
Steckbrücken M-M Gelb	1	
Steckbrücken M-M Weiß	1	
Steckbrücken M-M Lila	1	
Steckbrücken M-M Schwarz	1	

12.17 Glossar

Bash-Shell:
Wir auf dem PC ein Terminalfenster geöffnet, dann wird dazu automatisch ein Programm gestartet, welches die im Fenster eingegebenen Textbefehle entgegennimmt, interpretiert und ausführt. Dieses Programm ist üblicherweise die sogenannte "local" „Bash-Shell“ des Linux-Betriebssystems des PC. Das Terminalfenster an sich wird auch „Konsole“ genannt. Baut man in diesem Terminalfenster eine SSH-Verbindung mit dem BB auf, so erscheint darin automatisch die "remote" Bash-Shell des Linux-Betriebssystems des BB.

Toolchain:
Hierunter versteht man die „Kette aus Werkzeugen“, die benötigt wird, um ein Programm für ein eingebettetes System zu schreiben, zu kompilieren, auf das System zu übertragen und dort zu debuggen.

Cross Compiler:
Der Compiler erstellt aus einem Quellcode ein ausführbares Programm. Dies geschieht meistens auf einem PC, wobei das ausführbare Programm ebenfalls für einen PC der selben Art gedacht ist. Wird auf einem PC der Quellcode erstellt, das ausführbare Programm soll aber auf dem Linux-Betriebssystem des BB verwendet werden, dann muss der Quellcode entweder auf dem BB kompiliert werden, oder auf dem PC muss ein Cross Compiler eingesetzt werden.

Target:
Das eingebettete System (hier das LES BB), auf welches von einem PC aus zugegriffen wird. Wird auch „remote machine“ genannt im Gegensatz zum PC, der als „local machine“ bezeichnet wird.

Host und Client Computer:
Ein Client ist ein Computer, der eine Dienstleistung eines Host-Computers in Anspruch nimmt. Baut man im Terminalfenster des PC eine SSH-Verbindung mit dem BB auf, dann wird der PC zum Client des Host-Computers BB.

Pfad:
Auch PATH genannt. Eine Liste von Verzeichnissen, in denen das Betriebssystem nach einer ausführbaren Programmdatei sucht, nachdem man deren Namen (ohne „./“) in das Terminalfenster eingegeben hat.

Kernel:
Damit ist das eigentliche Linux-Betriebssystem gemeint. Der Kernel ist ein C-Programm, das für die entsprechende Rechnerplattform wie den BB kompiliert wird und dann als Image auf die Plattform übertragen wird. Diese ausführbare Datei findet sich unter `/boot`.

Image:
Hiermit ist eine 1:1 bitweise Kopie des Speichermediums des BB gemeint. Ein Image kann aus dem Internet (meist komprimiert) heruntergeladen werden und auf die SD-Karte bzw. den eMMC des BB übertragen werden. Umgekehrt kann der Ist-Zustand des Speichermediums des BB als Image auf einen PC gesichert und von da aus auf SD-Karten anderer BB geklont werden.

Zu den ROS-Begriffen findet sich in [11], Kapitel 4.1 "ROS Terminology" ein exzellenter Glossar.

Quellenverzeichnis

1: Molloy, D.: Exploring Beaglebone, Second Edition. Wiley, Indianapolis, 2019.

2: BeagleBoard.org Foundation: beagleboard.org. Letzter Aufruf 02/2020.

3: Yoder, M. A. et al.: BeagleBone Cookbook. O'Reilly, Sebastopol, 2015.

4: Santos, R. et al.: BeagleBone for Dummies. JohnWiley, Hoboken, 2015.

5: Adafruit Inc.: learn.adafruit.com/category/beaglebone. Letzter Aufruf 02/2020.

6: Hamilton, C. A.: BeagleBone Black Cookbook. Packt Publishing, Birmingham, 2015.

7: Hersteller Seeed, China: wiki.seeedstudio.com/BeagleBone_Green/. Letzter Aufruf 02/2020.

8: Distributor EXP GmbH, Saarbrücken: exp-tech.de. Letzter Aufruf 02/2020.

9: Hersteller Seeed, China: seeed.cc. Letzter Aufruf 02/2020.

10: Bartmann, Erik: Die elektronische Welt mit Raspberry Pi entdecken. O'Reilly, Sebastopol, 2016.

11: Pyo, Y. S.: ROS Robot programming. ROBOTIS Co., Ltd., 2017.

12: ROS.org, offizielle Wikiseiten: wiki.ros.org/ROS/Introduction. Letzter Aufruf 02/2020.

13: ROS.org, offizielle Installationsanleitungen: wiki.ros.org/noetic/Installation/Ubuntu. Letzter Aufruf 02/2021.

14: Terminal-Multiplexer Terminator: wiki.ubuntuusers.de/Terminator/. Letzter Aufruf 02/2020.

15: Setting up a Raspberry Pi as a WiFi access point: learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point/overview. Letzter Aufruf 11/2016.

16: 01 Intel Open Source: Connman: 01.org/connman/documentation. Letzter Aufruf 04/2020.

17: Bartmann, Erik: Die elektronische Welt mit Arduino entdecken. O'Reilly, Sebastopol, 2014.

18: beaglebone-universal-io, Tool config-pin: github.com/beagleboard/bb.org-overlays/tree/master/tools/beaglebone-universal-io. Letzter Aufruf 02/2020.

19: Exploring Beaglebone: exploringbeaglebone.com. Letzter Aufruf 04/2016.

20: BeagleBone Black Collars: www.doctormonk.com/2014/01/beaglebone-black-collars.html. Letzter Aufruf 06/2016.

21: Beschriftungsaufkleber für BB-Buchsenleisten: www.logicsupply.com/media/resources/beaglebone/BeagleBonePinGuide.pdf. Letzter Aufruf 10/2018.

22: Raspberry Pi: I2C-Konfiguration und -Programmierung: www.netzmafia.de/skripten/hardware/RasPi/RasPi_I2C. Letzter Aufruf 10/2018.

23: Raspberry Pi: Serielle Schnittstelle: netzmafia.de/skripten/hardware/RasPi/RasPi_Serial. Letzter Aufruf 10/2018.

24: Download "Digitale Signalverarbeitung mit Python": http://www.pyfda.org/. Letzter Aufruf 10/2018.

25: Python in der Schule und Hochschule: www.tec.reutlingen-university.de/prof-mack/mechatronikstudium/python-in-der-schule-und-hochschule/. Letzter Aufruf 02/2020.

26: Klein, B.: Einführung in Python 3: Für Ein- und Umsteiger. Hanser Verlag, München, 2014.

27: Adafruit BMP Python Library: learn.adafruit.com/using-the-bmp085-with-raspberry-pi/using-the-adafruit-bmp085-python-library. Letzter Aufruf 01/2016.

28: Woyand, H.-B.: Python für Ingenieure. Hanser Verlag, München, 2017.

29: ROS Answers Forum: answers.ros.org/question/354867/. Letzter Aufruf 10/2020.

30: ROS.org, offizielle Tutorials: wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem. Letzter Aufruf 02/2020.

31: Grove Base Cape for BeagleBone v2: www.seeedstudio.com/wiki/Grove_Base_Cape_for_BeagleBone_v2. Letzter Aufruf 07/2016.

32: Nano Editor Cheat Sheet: www.nano-editor.org/dist/latest/cheatsheet.html. Letzter Aufruf 02/2020.

33: Unix/Linux Command Reference: files.fosswire.com/2007/08/fwunixref.pdf. Letzter Aufruf 02/2020.

34: Beagleboard: Cape Expansion Headers: http://elinux.org/Beagleboard:Cape_Expansion_Headers. Letzter Aufruf 06/2016.

35: Verwendung von pip unter Ubuntu.: matthew-brett.github.io/pydagogue/installing_on_debian.html. Letzter Aufruf 02/2020.

36: Rush, C.: 30 BeagleBone Projects for the Evil Genius. Mv Graw Hill Education, New York, 2014.

37: Lumme, J.: BeagleBone Home Automation. Packt Publishing, Birmingham, 2013.

38: Grimmet, R.: BeagleBone Robotic Projects. Packt Publishing, Birmingham, 2013.

39: Chavan, Y.: Programming the BeagleBone. Packt Publishing, Birmingham, 2016.