KOMPETENZZENTRUM FÜR MATHEMATISCHE MODELLIERUNG IN MINT-PROJEKTEN IN DER SCHULE

KOMMS Reports Nr. 14 (2021)

Reports zur Mathematischen Modellierung
in MINT-Projekten in der Schule

KOMMS
Kompetenzzentrum für Mathematische Modellierung in MINT-Projekten in der Schule

MINT-Projekte mit Arduino & Rasperry Pi
Technische Grundlagen und Projektideen

Patrick Capraro

EUROPÄISCHER SOZIALFONDS
Mehr Chancen für Rheinland-Pfalz

EUROPÄISCHE UNION
Europäischer Sozialfonds

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Zusammenfassung:
In diesem Text werden einige wichtige Grundlagen zusammengefasst, mit denen ein schneller Einstieg in das Arbeiten mit Arduino und Raspberry Pi möglich ist. Wir diskutieren nicht die Grundfunktionen der Geräte, weil es dafür zahlreiche Hilfestellungen im Internet gibt. Stattdessen konzentrieren wir uns vor allem auf die Steuerung von Sensoren und Aktoren und diskutieren einige Projektideen, die den MINT-interdisziplinären Projektunterricht bereichern können.

Inhaltsverzeichnis

1. Die Hardware

1.1. Arduino vs. Raspberry Pi

Der Arduino und der Raspberry Pi werden oft in einem Atemzug genannt, wenn es um Projekte geht, in denen Geräte gesteuert und Sensordaten verarbeitet werden. Beide sind jedoch sehr unterschiedlich in ihren Fähigkeiten.

Der Raspberry Pi ist ein vollwertiger Computer. Man kann mit vielen Programmiersprachen auf dem Gerät arbeiten und nach Belieben Software installieren. Man kann Filme und Musik darüber abspielen oder einen Webserver betreiben. Ein besonderes Feature sind die GPIO (**g**eneral **p**urpose **i**nput and **o**utput) Pins, mit denen man externe Geräte anschließen und steuern kann. Die Pins haben jedoch einen Nachteil: Sie arbeiten alle digital (bis auf einen, der über PWM (Pulsweitenmodulation) verfügt). Will man anaolge Daten verarbeiten, muss man daher weitere Geräte (z.B. einen analog-digital-Wandler) dazwischenschalten.[1]

Der Arduino hingegen ist ein Mikrocontroller: Er verfügt über digitale und anaolge Pins und ist daher wesentlich vielseitiger, wenn es um auslesen von Sensoren und Steuern von Geräten geht. Seine Speicherkapazität ist allerdings sehr begrenzt: Er kann ein Programm mit einer beschränkten Größe (auf vielen Boards 32 kB) speichern und in Endlosschleife abarbeiten. Wird ein neues Programm auf das Board geladen, wird das alte überschrieben. Für komplexe Berechnungen eignet er sich nicht und er hat auch keine Möglichkeit, Sensordaten dauerhaft zu speichern. Er eignet sich gut, um gemessene oder berechnete Werte direkt an ein anderes Gerät weiterzugeben (z.B. ein Display; eine LED, die den aktuellen Zustand anzeigt; ein Relais; einen Motor, der gesteuert wird; …).

1.1.1. Arduino

Es gibt verschiedene Arduino-Boards, die sich in ihrer Leistung und auch im Preis deutlich unterscheiden. Daneben gibt es von Fremdfirmen auch Nachbildungen, die kompatibel und etwas billiger sind. So kann man Nachahmungen des Arduino nano bereits für unter 5 Euro das Stück bekommen.

Programmiert werden die Boards in der Programmiersprache C. Es gibt eine kostenlose Entwicklungsumgebung, die über `www.arduino.cc` erhältlich ist. Der Code kann dann über ein USB-Kabel auf den Controller geladen werden.

1.1.2. Raspberry Pi

Der Raspberry Pi ist ein vollwertiger Computer. Als Festplatte lässt sich eine SD-Karte verwenden. Über ein HDMI Kabel kann ein Bildschirm angeschlossen werden und per USB können Maus und Tastatur betrieben werden.

Viel interessanter als diese klassische Arbeitsweise ist jedoch häufig der *headless*-Betrieb: Hierbei wird der Pi ohne Tastatur, Maus und Bildschirm betrieben und muss sich lediglich in einem Netzwerk befinden. Über entfernte Rechner lässt sich der Pi via **ssh**[2] steuern, oder aber er agiert voll automatisiert.

[1] Zum Unterschied zwischen analogen und digitalen Pins siehe Anhang A.1.
[2] **s**ecure **sh**ell, siehe Anhang B.4.

Es gibt für den Raspberry Pi zahlreiche Geräte, mit denen er sich erweitern lässt: eine Kamera, ein Touch-Display und vieles andere. Außerdem können über die GPIO Pins verschiedene Geräte oder selbstgebaute Schaltungen eingebunden werden.

Es gibt mittlerweile eine vielzahl von Raspberry Pi Modellen, die sich in ihrer Funktionsweise sehr unterscheiden: Prozessorleistung und Anzahl der USB Ports sind unterschiedlich, die neueren Modelle verfügen standardmäßig über WLAN und Bluetooth, usw.

Die ersten Modelle waren für ca. 35 Euro erhältlich. Bei den neueren Modellen gibt es deutliche Unterschiede bei Preis und Leistung: Der Raspberry Pi Zero ist für ca. 25 Euro erhältlich, die leistungsstarken Modelle kosten bis zu 85 Euro (Stand Januar 2021), wobei unter Umständen noch zusätzliche Kosten anfallen: Stromkabel (ein USB-Ladekabel vom Smartphone reicht unter Umständen aus) und SD-Karte (8GB oder mehr) braucht man in jeden Fall, weitere Dinge wie ein Gehäuse sind optional.

1.2. Stromversorgung

1.2.1. Arduino

Die kleineren Arduino-Boards benötigen eine 5 V Eingangsspannung über den Mikro- bzw. Mini-USB-Anschluss. Sie lassen sich beispielsweise über den USB-Port des Computers mit Strom versorgen.

Die größeren Boards benötigen 7-12 V und werden in aller Regel über ein Netzteil betrieben.

Man kann Arduinos aber auch über die Pins mit Strom versorgen und dann beispielsweise eine Batterie anklemmen.

Der Stromverbrauch hängt stark von den angeschlossenen Geräten ab. Üblicherweise arbeitet man mit Stromstärken von 20-50 mA. Wenn man Geräte mit hohem Verbrauch steuern will (z.B. Elektromotoren), dann sollte man diese über eine externe Spannungsquelle versorgen (siehe Anhang A.2).

1.2.2. Raspberry Pi

Die Raspberry Pis werden über einen Mikro USB (bei neueren Modellen USB-C) Anschluss mit Strom versorgt. Die Spannung beträgt 5 V und der Stromverbrauch beträgt je nach Modell zwischen 700 mA und 3 A.

Bei den leistungsschwachen Geräten reicht oft ein Handyladekabel aus. Bei den leistungsstarken Modellen lohnt es sich, ein Kabel zu kaufen, das speziell für den Raspberry vertrieben wird. Man sollte davon absehen, das Gerät über den USB-Port des Computers mit Strom zu versorgen. Das war zwar bei den ersten Modellen noch möglich, aber bei zu hohem Stromverbrauch kann der Raspberry Pi abstürzen. Womöglich schädigt man mit dieser Methode auch den Computer, daher sollte man besser darauf verzichten.

1.3. Mit den Geräten arbeiten lernen

Es gibt im Internet zahlreiche Lernanleitungen, die den Einstieg erleichtern. Eine gute Anlaufstelle sind die Webseiten der Organisationen, welche die Geräte entwickeln und ver-

treiben[3].

Der Einstieg ist beim Arduino sehr einfach. Man muss lediglich lernen, mit der Arduino IDE zu arbeiten und einfachen C-Code programmieren können (man braucht wenige Befehle, die man schnell gelernt hat; Grundlagenkenntnisse über Schleifen, Verzweigungen und Variablen sind aber von Vorteil).

Der Raspberry ist wesentlich Vielseitiger, was aber auch den Einstieg langwieriger macht. Man sollte Grundlagenkenntnisse über Linux-Betriebssysteme haben (idealerweise auf der Kommandozeile[4]). Was Programmierkenntnisse angeht, ist man zunächst nicht an eine Sprache gebunden. Python ist allerdings sehr Hilfreich, da die meisten Online-Tutorials für den Pi sich daran orientieren.

1.4. Headless-Betrieb beim Raspberry Pi

1.4.1. Das Problem mit der IP-Adresse

Oft will man den Raspberry Pi ohne Peripheriegeräte (Bildschirm und Tastatur) betreiben. Dazu muss man ihn so konfigurieren, dass er sich automatisch in ein WLAN einwählt, oder man verbindet ihn über ein Kabel mit dem jeweiligen Netzwerk. Über einen ssh[5]-Zugang kann man dann auf dem Pi arbeiten.

Für diese Methode benötigt man allerdings die IP Adresse des Pi. In vielen Netzwerken wird sie standardmäßig bei jedem Anschalten des Geräts neu vergeben, so dass man erst einige Schritte unternehmen muss, bis man weiterarbeiten kann.

- **Netzwerkadresse über den Router auslesen:** Befindet man sich in seinem eigenen (Heim-)Netzwerk und hat Zugriff auf den Router, dann kann man dort die IP Adresse auslesen.
- **statische IP Adresse vergeben:** In seinem eigenen Netzwerk hat man auch die Möglichkeit, eine bestimmte IP Adresse für ein Gerät zu reservieren. Dazu muss man in aller Regel die MAC Adresse (Hardwareadresse) des Netzwerkgeräts hinterlegen. Diese kann man in Erfahrung bringen, indem man den Befehl `ifconfig` in die Konsole eingibt (siehe Abbildung 1).
- **Hostnamen verwenden:** Man kann dem Pi beim Einrichten auch einen individuellen Hostnamen geben. Dazu verwendet man den Befehl `sudo raspi-config` und kann sich durch das Menü zur Einstellung `hostname` durchklicken. Hat der Pi den Hostnamen `mypi`, dann kann man sich per ssh mit dem Befehl `ssh pi@mypi.local` verbinden.
  Achtung: Das Auflösen des Hostnamens funktioniert über unixartige Betriebssysteme (macOS, Linux) standardmäßig, über Windows sind möglicherweise weitere Maßnahmen notwendig.
- **IP Adresse beim Gerätestart ausgeben:** Man kann sich eine dynamisch zugewiesene IP Adresse auch ausgeben lassen. Eine Möglichkeit besteht darin, die Adresse über den Audioausgang vorlesen zu lassen. Dazu muss die Software `espeak`[6] installiert sein und in die Datei `<heimverzeichnis>/.rc.local` muss der Code aus Listing 1 angehängt werden.

[3] `www.arduino.cc` bzw. `www.raspberrypi.org`.
[4] Eine kleine Einführung in die Linux-Kommandozeile finden Sie im Anhang B.
[5] Siehe Anhang B.4.
[6] `sudo apt install espeak`

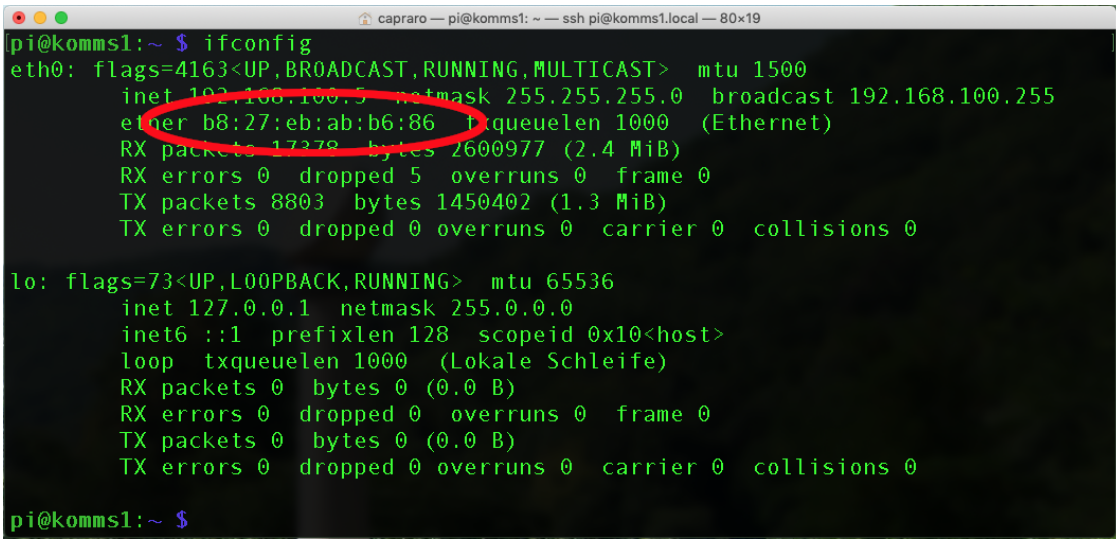


Abbildung 1: Output des Befehls `ifconfig`. Die MAC Adresse ist ein Code bestehend aus zweistelligen Hexadezimalzahlen, die durch Doppelpunkte getrennt werden.

```
hostname –I | cut –d' ' –f1 | espeak –vde
```

Listing 1: Der Befehl `hostname -I` liest die IP-Adresse (es werden ipv4 und ipv6 ausgegeben) aus, `cut -d' ' -f1` wählt den ersten Eintrag (ipv4) und löscht alles andere, `espeak` gibt den Text über den Audioausgang aus.

1.5. Der Raspberry im Feldeinsatz ohne Netzwerk

Manchmal will man mit dem Raspberry Pi mobil sein, beispielsweise, wenn er in einem Roboter verbaut ist, oder wenn man Sensormessungen im Freien machen möchte. Um trotzdem per ssh Zugriff zu haben, muss man erfinderisch sein.

Lösung 1: Eigenen WiFi Hotspot erzeugen.
Der Raspberry lässt sich so konfigurieren, dass er seinen eigenen WLAN-Hotspot erzeugt. Mit den neueren Modellen (ab Raspberry Pi 3) geht das ohne weitere Hardware, da ein WiFi Chip integriert ist. Mit älteren Modellen braucht man einen WiFi Stick, und hier ist nicht jedes Modell gleichwertig. Gute Erfahrungen habe ich mit *Edimax*-Sticks gemacht.

Für das Erzeugen dieses Hotspots gibt es diverse Online-Tutorials, z.B. unter
https://howtoraspberrypi.com/create-a-wi-fi-hotspot-in-less-than-10-minutes-with-pi-raspberry/.

Sie können dann ihre IP Adresse innerhalb eines gewissen Wertebereichs frei wählen (z.B. 10.0.0.1) und können einen Netzwerknamen und ein Passwort festlegen, damit sich andere Geräte in dieses Netzwerk einwählen können. Damit ist ein ssh Login auch fernab jeglicher Netzwerk-Infrastruktur möglich.

Lösung 2: Ein Kabelnetzwerk erzeugen
Man kann den Raspberry auch direkt per Netzwerkkabel mit einem anderen Computer verbinden. Dieser muss dann entsprechend konfiguriert werden, damit die Kommunikation mit dem Pi möglich ist.

- **macOS (getestet unter High Sierra und Mojave):** In den Systemeinstellungen zunächst

unter `Freigaben` eine Internetfreigabe über den entsprechenden Ethernet-Port erlauben. Dann unter `Netzwerk` den Port über DHCP konfigurieren. Bei meinen bisherigen Versuchen hatte der Pi stets die IP 192.168.2.2.

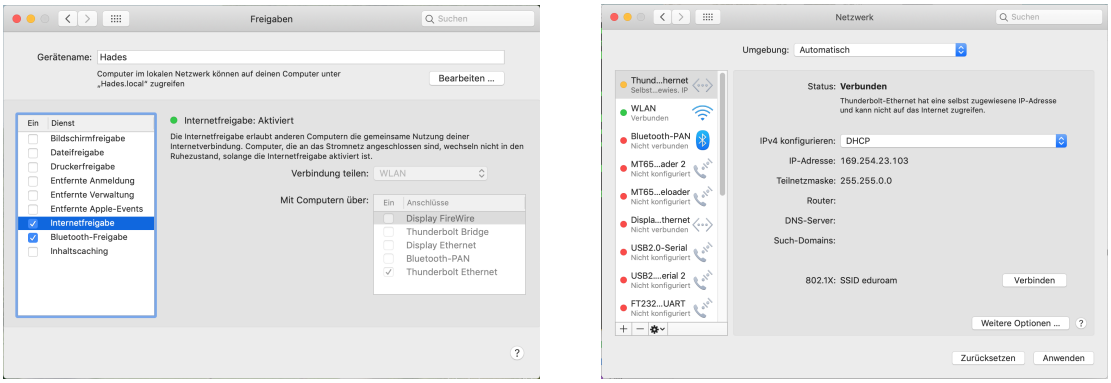


Abbildung 2: Kabelnetzwerk in macOS einrichten.

- **Ubuntu (getestet unter 16.04):** Unter LAN-Einstellungen kann ein neues Netzwerk eingerichtet (bzw. ein bestehendes Netzwerk konfiguriert) werden. In den Einstellungen wählt man dann unter IPv4 die Methode *Nur Link-Local*.

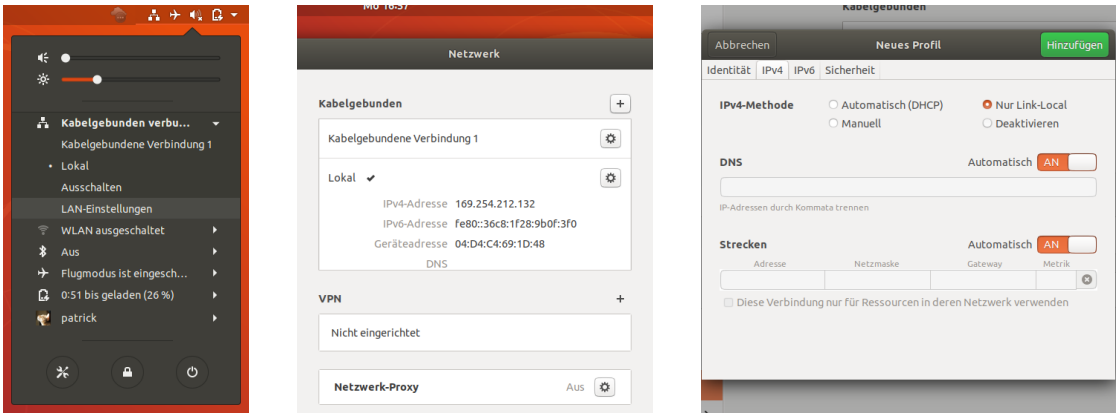


Abbildung 3: Kabelnetzwerk in Ubuntu einrichten.

- **Windows:** Bei einem Test unter Windows 10 musste lediglich das Ethernetkabel eingesteckt werden und die Verbindung war erfolgreich. Dieses Ergebnis konnten wir jedoch nicht auf älteren Windowsversionen reproduzieren.

1.6. Die GPIO-Pins am Raspberry Pi

1.6.1. Steuerung über die Kommandozeile

Über die Kommandozeile lassen sich Pins mit folgenden Befehlen ansteuern (in den Beispielen wird stets Pin Nr. 17 verwendet):

Zunächst muss der Pin aktiviert werden. Das geht über den Befehl

```
echo "17" > /sys/class/gpio/export
```

Um Geräte zu steuern, muss der Modus auf `out` gestellt werden (und um Daten zu empfangen entsprechend auf `in`):

```
echo "out" > /sys/class/gpio/gpio17/direction
```

Nun können wir dem Pin wahlweise den Wert 0 oder 1 zuweisen:

```
echo "1" > /sys/class/gpio/gpio17/value
```

Um Daten zu lesen (beim Modus `in`):

```
cat /sys/class/gpio/gpio17/value
```

Um den Pin zu deaktivieren, benutzt man den Befehl

```
echo "17" > /sys/class/gpio/unexport
```

Achtung: Bei einer Internetsuche nach den Begriffen 'GPIO' und 'Kommandozeile' wird man vermutlich auf die Software **wiringpi** verwiesen. Der Entwickler hat jedoch im August 2019 den Support eingestellt, weswegen die Software nicht mehr ohne weiteres installiert werden kann.

1.6.2. Steuerung über Python

Es gibt verschiedene Pakete, die das Steuern der Pins ermöglichen. Wir nutzen `RPi.GPIO`.

Zunächst lädt man

```
import RPi.GPIO as GPIO
```

Dann muss die Nummerierung der GPIO Pins festgelegt werden. Wir verwenden das Schema BCM, um die selbe Nummerierung wie bei der Steuerung über die Kommandozeile zu erhalten

```
GPIO.setmode(GPIO.BCM)
```

Schließlich können wir einen Pin über die Nummer auswählen und entweder für Input oder Output konfigurieren. Beim Input setzt man

```
GPIO.setup(17, GPIO.IN)
```

und kann mit dem Befehl

```
pin=GPIO.input(17)
```

abrufen, ob ein Signal anliegt (pin==1) oder nicht (pin==0).

Will man einen Output erzeugen, dann setzt man zunächst

```
GPIO.setup(17, GPIO.OUT)
```

und kann dann anschließend den Wert `HIGH` oder `LOW` setzen, z.B.

```
GPIO.output(17, GPIO.HIGH)
```

1.7. Die Pins am Arduino

1.7.1. Pin aktivieren

Wir verwenden im folgenden Beispiel den Pin mit der Nummer 7. Um den Pin für die Steuerung eines Geräts zu aktivieren, muss im Setupteil des Codes der Befehl

```
pinMode(7,OUT);
```

ausgeführt werden. Um Daten zu lesen vewenden wir

```
pinMode(7,IN);
```

1.7.2. Digitalpins

Im Hauptteil des Programms kann anschließend mit

```
digitalWrite(7,HIGH);
```

der Pin eingeschaltet werden, und mit

```
digitalWrite(7,LOW);
```

wird er wieder ausgeschaltet. Auslesen kann man den Pin mit dem Befehl

```
wert=digitalRead(7);
```

1.7.3. Analogpins

Die Analogpins am Arduino geben nicht wirklich ein analoges Signal aus, sondern eine Pulsweitenmodulation (PWM)[7], die das analoge Signal imitiert.
Nun verwenden wir als Beispiel den Pin A4. Um ihn einzuschalten verwenden wir den Befehl

```
analogWrite(A4,wert);
```

wobei die Größe `wert` eine ganze Zahl von 0 bis 255 sein kann. 255 entspricht dabei einem 5 V Signal (bzw. 3,3 V bei manchen Boards). Ausgelesen wird der Pin mit dem Befehl

```
wert=analogRead(A4);
```

Dabei werden Werte zwischen 0 und 1023 gelesen. Auch hier entspricht der Maximalwert einer Spannung von 5 V (bzw. 3,3 V).
Achtung: Evt. sind die Funktionen `analogRead()` bzw. `analogWrite()` nicht auf allen analogen Pins verfügbar. Hier helfen die offiziellen Internetseiten des Arduinoprojekts weiter.

2. Elektronische Bauteile anschließen und steuern

2.1. Der Klassiker zum Einstieg: Eine LED blinken lassen

Zur Vorbereitung wird eine LED mit dem ausgewählten Pin verbunden (Pluspol an der LED ist meist die längere Elektrode). Der Minuspol muss selbstverständlich mit einem Groundpin verbunden werden. Achtung: Bitte auch einen Widerstand dazwischenschalten, sonst kann die LED beschädigt werden.

2.1.1. Arduino

Unter `setup` wird der Pin für den Output aktiviert. Der Code unter `loop` wird in Endlosschleifen wiederholt. Auf dem ausgewählten Pin wird abwechselnd eine Spannung angelegt und wieder abgeschaltet. Dazwischen finden Verzögerungen von je 1 s statt.

```
void setup() {
  pinMode(5, OUTPUT);
}

void loop() {
```

[7] Siehe Anhang A.1.

```
  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);
}
```

Listing 2: Codebeispiel *Blinkende LED* für Arduino.

2.1.2. Raspberry Pi: shell

Um die LED automatisch blinken zu lassen, kann man ein Skript schreiben (siehe Listing 3), das die Befehle aus Abschnitt 1.6 ausführt (für eine Einführung zu Shellskripten siehe Anhang B.5). Wenn wir die Datei `blink.sh` nennen, dann können wir sie mit dem Befehl `./blink.sh` aufrufen, sofern wir uns in dem selben Pfad befinden, oder wir verwenden `<pfad>/blink.sh`.

```
#!/bin/bash
echo "17" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio17/direction
while [ 1 ]
do
  echo "1" > /sys/class/gpio/gpio17/value
  sleep 1
  echo "0" > /sys/class/gpio/gpio17/value
  sleep 1
done
```

Listing 3: Codebeispiel *Blinkende LED* mit bash. Das Programm läuft in einer Endlossschleife; Abbrechen mit STRG+C.

2.1.3. Rasbperry Pi: Python

Das Paket `RPi.GPIO` stellt Befehle zur Verfügung, welche die GPIO Pins ansteuern können. Auch hier muss zunächst der Pin für den Output aktiviert werden, bevor die Spannung ein- (`HIGH`) bzw. ausgeschaltet (`LOW`) wird.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(4,GPIO.OUT)
while (1):
   GPIO.output(4,GPIO.HIGH)
   time.sleep(1)
   GPIO.output(4,GPIO.LOW)
   time.sleep(1)
```

Listing 4: Codebeispiel *Blinkende LED* mit python.

Achtung: Wird das Pythonprogramm beendet, dann werden auch die Pins deaktiviert. Will man einen Pin über längere Zeit eingeschaltet lassen, in der sonst nichts passieren soll,

dann ist unter Umständen die Steuerung über die Kommandozeile (Abschnitt 2.1.2) die klügere Wahl.

2.2. Einen Taster auslesen

Viele Lernsets für Arduino und Raspberry Pi beinhalten vierpolige Taster. Der innere Aufbau des Tasters ist in Abbildung 4 illustriert.

Um den Taster auszulesen, muss zunächst eine Spannung angeschlossen werden. Dazu verbindet man auf der Seite 1 einen Pin mit einer Spannungsquelle (5 V) und auf der Seite 2 einen Pin mit GND. Zusätzlich sollte man einen großen Widerstand (Größenordnung 10kΩ) zwischen Taster und GND schalten, damit man beim Tastendruck keinen Kurzschluss erzeugt.

Nun kann man auf Seite 2 einen Pin anschließen, der auf input konfiguriert wurde. Beim Tastendruck liegt eine Spannung an, wenn der Taster losgelassen wird, liegt keine an. Zum Auslesen des Signals beachten wir die Anweisungen in den Abschnitten 1.6 bzw. 1.7.3.

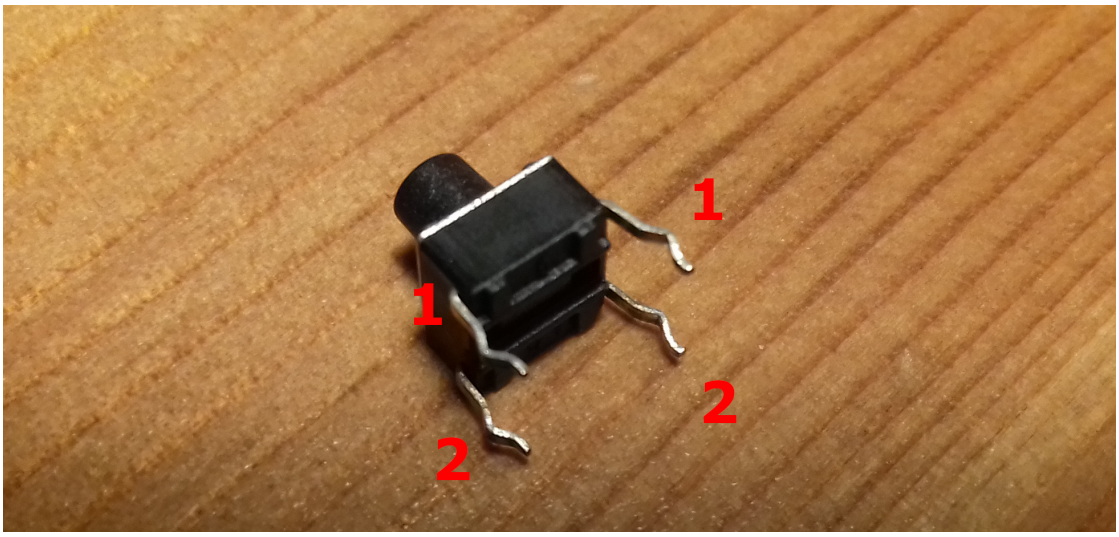


Abbildung 4: Die Pins mit gleicher Beschriftung sind miteinander verbunden. Wird der Taster gedrückt, dann sind alle Pins verbunden.

2.3. Ein einfacher Sensor: analoger Temperatursensor

Das folgende Beispiel wurde mit dem Thermistor KY-013 aus dem **Sensorkit X40** von **joy-it** umgesetzt. Der Sensor besteht aus einem temperaturabhängigen Widerstand. Die Spannung, die am Widerstand abfällt, wird gemessen und bildet das analoge Signal (zwischen 0 V und 5 V). Die Stärke des Signals kann anschließend in eine Temperatur umgerechnet werden.

2.3.1. Analoge Signale beim Arduino

Der Sensor wird auf folgende Weise mit dem Arduino-Board verbunden: Der V+ Pin mit 5 V, der GND Pin mit GND und der Sensor-Pin mit einem analogen Pin, in diesem Beispiel mit

A5. Wenn wir nun das Signal auslesen, bekommen wir mit dem Code in Listing 5 eine Zahl zwischen 0 und 1023 (den Output können wir über den seriellen Monitor beobachten).

```
int sensorPin = A5;

void setup() {
  Serial.begin(9600);
  pinMode(sensorPin,INPUT);
}

void loop() {
  int daten=analogRead(sensorPin);
  Serial.println(daten);
  delay(2000);

}
```

Listing 5: Code zum Auslesen des Signals eines analogen Sensors.

Mit diesem Ergebnis kann man noch nicht allzu viel anfangen, da es nur eine Aussage über die ausgelesene Spannung macht und nicht über die Temperatur. Die Firma joy-it liefert jedoch Codebeispiele mit, aus denen sich die verwendete Umrechnungsformel beschaffen lässt. In Listing 6 sehen wir einen verbesserten Arduino-Code.

```
#include<math.h>
int sensorPin = A5;

//Funktion rechnet Sensordaten in Temperatur um
//Entnommen aus dem joy—it—Tutorial
double Thermistor(int RawADC) {
  double Temp;
  Temp = log(10000.0 * ((1023.0 / RawADC — 1)));
  Temp = (0.000234125 + (0.0000000876741 * Temp * Temp)) * Temp;
  Temp = 1 / (0.001129148 + Temp);
  Temp = Temp — 273.15;
  return Temp;
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  int daten = analogRead(sensorPin);
  Serial.println(Thermistor(daten));
  delay(200);
}
```

Listing 6: Code zum Auslesen des Signals eines analogen Sensors.

2.3.2. Analoge Signale beim Raspberry-Pi

Wie bereits erwähnt kann der Raspberry Pi die anaolgen Sensordaten nicht direkt einlesen. Man muss daher einen analog-digital-Konverter dazwischenschalten. Je nach Funktionsweise des Konverters kann dieses digitale Signal in eine Zahl umgewandelt werden (es gibt unterschiedliche Protokolle zum Einlesen digitaler Daten, z. B. 1-Wire (siehe Abschnitt 2.4) oder i2c.). Anschließend kann man ähnlich wie in Listing 6 vorgehen und den Wert in eine Temperatur umrechnen.

2.4. Digitale Sensoren: 1-Wire Temperatursensor

Einige digitale Sensoren nutzen zur Datenübertragung das 1-Wire Protokoll. Hier müssen sowohl beim Arduino als auch beim Raspberry Pi entsprechende Vorbereitungen getroffen werden, um die Daten richtig einlesen zu können.

Der Sensor DS1820 hat 3 Pins. Davon ist einer für die Spannungsversorgung (3,3 V), einer für die Masseverbindung und einer für das Signal. Der Signalpin wird mit einem digitalen Pin am Gerät verunden. Üblicherweise verbindet man die Pins für Spannung und Signal noch mit einem 4,7 kΩ-Widerstand.

2.4.1. 1-Wire beim Arduino

Zunächst müssen die Bibliotheken `OneWire.h` und `DallasTemperature.h` installiert werden[8]. Anschließend benutzen wir den Code in Listing 7, um die Temperatur über den seriellen Monitor auszugeben. Der Signalpin des Sensors ist übrigens an den digitalen Pin 4 des Arduino angeschlossen.

```
#include <Wire.h>
#include <DallasTemperature.h>
#include <OneWire.h>
OneWire oneWire(4);
DallasTemperature sensors(&oneWire);

void setup(){
  Serial.begin(9600);
  sensors.begin();
}

void loop(){
  sensors.requestTemperatures();
  Serial.println(sensors.getTempCByIndex(0));
  delay(1000);
}
```

Listing 7: Temperaturausgabe auf dem seriellen Monitor des Arduino.

[8] Siehe Anhang C.

2.4.2. 1-Wire beim Raspberry

Zunächst muss der Raspberry entsprechend konfiguriert werden, so dass er die Daten empfangen und verstehen kann. Dazu schreibt man in die Datei `/boot/config.txt`

```
dtoverlay=w1-gpio,gpiopin=4,pullup=on
```

um den Pin mit der Nummer 4 zu aktivieren. In die Datei `/etc/modules` schreiben wir zusätzlich

```
w1-gpio pullup=1
w1-therm
```

Nach einem Neustart des RPI finden wir im Ordner `/sys/bus/w1/devices/` einen Ordner, dessen Name der Seriennummer des Sensors entspricht. Evt. sind es mehrere solcher Ordner, falls mehrere Sensoren angeschlossen sind (es können mehrere Sensoren am selben Pin angeschlossen sein).

Wir können nun die Datei `/sys/bus/w1/devices/<seriennummer>/w1_slave` auslesen. Die Temperatur, angegeben in Milligrad Celsius, steht hinter einem `t=`.

2.4.3. 1-Wire-Daten archivieren

Da der Raspberry Daten speichern kann, können wir Temperaturmessungen automatisieren und die Daten in einer Textdatei hinterlegen, um sie zu einem späteren Zeitpunkt elektronisch oder manuell auszuwerten. Dabei hilft das Skript in Listing 8.

`grep` sucht die Zeile mit der Zeichenfolge `t=`, dann extrahiert `tail -c 6` die letzten 6 Zeichen des strings (den Zahlenwert); anschließend wird die Temperaturangabe so zerlegt, dass man die Milligrad-Werte und die ganzen Grad zu einer Kommazahl zusammensetzen kann; in der letzten Zeile werden die Daten in die Datei file.dat geschrieben.

Um den Code automatisch auszuführen, kann man einen sogenannten Cronjob[9] erzeugen, der das Skript aufruft.

```
#!/bin/bash
DATE=$(date '+%Y-%m-%d_%H:%M:%S')
temp=$(cat /sys/bus/w1/devices/<seriennr>/w1_slave | grep t=)
t=$(echo $temp | tail -c 6)
t1=$(echo $t | head -c 2)
t2=$(echo $t | tail -c 4)
echo "$DATE;$t1.$t2" >> file.dat
```

Listing 8: Code zum Archivieren der Temperaturdaten und der zugehörigen Datumangaben.

2.4.4. 1-Wire-Daten in einer HTML-Datei ausgeben

Wenn auf dem Raspberry Pi ein Webserver läuft, kann die Temperatur auf einer HTML-Seite ausgegeben werden. Den Zugriff auf die Daten kann beispielsweise ein php-Programm ausführen. Anschließend muss (ähnlich wie oben) der ausgelesene Text entsprechend formatiert werden. Der Code in Listing 9 zeigt wie es geht (hier ist wieder `<snr>` die Seriennummer des Sensors).

[9] Siehe Anhang B.6.

```php
<?php
$temp=exec('cat /sys/bus/w1/devices/<snr>/w1_slave | grep t=');
$temp=explode('t=',$temp);
$temp=$temp[1]/1000;
$temp=round($temp,2);
echo "Die Innentemperatur betraegt $temp &#x00B0;C";
?>
```

Listing 9: Code zum Wiedergeben der Temperaturanzeige in einer HTML-Datei

2.4.5. 1-Wire mit Python auslesen

Um die Temperatur mit Python zu erhalten, gehen wir einen ähnlichen Weg wie in Listing 8 und lesen die entsprechende Datei aus. Wir sehen ein entsprechendes Codebeispiel in Listing 10.

```python
f = open("/sys/bus/w1/devices/28-0416a0f316ff/w1_slave","r")
x=f.read()
temp=x[-6:]
temp=float(temp)/1000
print(temp)
```

Listing 10: Auslesen der Temperaturdaten in Python.

2.5. Motoren steuern

2.5.1. Mögliche Schaltungen beim Gleichstrommotor

Es gibt verschiedene Möglichkeiten, eine Motorsteuerung zu realisieren. Welche Steuerung für das vorliegende Projekt geeignet ist, hängt dabei stark von den verwendeten Motoren ab, sowie von den Aufgaben, die bewältigt werden müssen.

Üblicherweise muss man für einen Motor gleich zwei getrennte Schaltungen aufbauen, da jede Drehrichtung ihren eigenen Stromkreis erfordert.

Für die Steuerung sind gegebenenfalls analoge Signale (bzw. PWM) sinnvoll, so dass es besser ist, den Arduino zu verwenden. Allerdings lässt sich am Raspberry Pi auch eine PWM am Pin 18 realisieren. Ein echtes analoges Signal lässt sich z.B. mit Hilfe eines Potentiometers erzeugen, muss dann allerdings manuell geregelt werden.

- **Motor direkt über das Gerät (Arduino bzw. Raspberry Pi) steuern**
  Es ist prinzipiell denkbar, den Motor als Verbraucher zwischen einen anaolgen oder digitalen Pin und den GND Pin zu schalten. Der Motor ist dann aktiv, wenn der Pin eingeschaltet wird, und er ist inaktiv, wenn der Pin ausgeschaltet wird. Bei einem anaolgen Pin kann man auch die Drehgeschwindigkeit des Motors regulieren. Bei den meisten Motoren kommt diese Lösung jedoch nicht infrage, da die benötigte Leistung zu groß ist.
  Achtung: Da ein Elektromotor Induktionsspannungen erzeugt, kann das steuernde Gerät beschädigt werden. Daher sollte man Induktionsspannungen durch eine Diode unterdrücken.

- **Motor durch einen Transistor steuern**
  Wenn wir den Motor durch einen Transistor steuern, können wir einen zweiten Stromkreis mit einer externen Spannungsquelle (z.B. Batterie) aufbauen. Dann können wir höhere Spannungen und Stromstärken zulassen, ohne Arduino und Raspberry Pi zu überfordern. Indem wir eine 5 V Spannung auf die Basis des Transistors legen, kann der Motorstromkreis geschlossen werden (bzw. geöffnet, sobald die Spannung an der Basis weg ist). Die Drehgeschwindigkeit kann reguliert werden, indem ein anaolger Pin an die Basis angeschlossen wird.
  Achtung: Auch der Transistor sollte durch Dioden vor Induktinosspannungen geschützt werden.
- **Motor durch eine Brückenschaltung steuern**
  Es gibt integrierte Schaltkreise, die alle wichtigen Elemente einer Motorschaltung bereits beinhalten. Wir stellen eine solche Schaltung im folgenden Abschnitt vor.

2.5.2. Ein Beispiel: Gleichstrommotor mit H-Brücke L293D

Wir verwenden im Beispiel den integrierten Schaltkreis L293D. Der Chip hat 4 Ausgänge und kann daher 2 Motoren steuern (2 Ausgänge je Motor, um beide Drehrichtungen steuern zu können). Die Pins des Schaltkreises haben folgende Funktionen (Nummerierung der Pins siehe Abbildung 5):

- enable (1+9): aktiviert jeweils einen Motor (und damit 2 output-Pins).
- output (3+6,11+14): je zwei output-Pins werden mit den beiden Polen eines Motors verbunden.
- input (2+7,10+15): je zwei input-Pins steuern einen Motor.
- GND (4,5,12,13): die vier inneren Pins dienen zur Masseverbindung.
- VCC1 (16): Spannungsversorgung des Schaltkreises (5 V).
- VCC2 (8): Spannungsversorgung des Motors (bis zu 36 V).

Das Prinzip lautet wie folgt: Der enable-Pin 1 aktiviert die output-Pins 3 und 6, wenn an ihm das Signal HIGH liegt. Mit den output-Pins ist der Elektromotor verbunden. Dieser dreht sich in die eine Richtung, wenn input-Pin 2 auf HIGH und input-Pin 7 auf LOW gestellt ist. Um die Drehrichtung zu ändern, muss 2 auf LOW und 7 auf HIGH gestellt werden.

Ganz ähnlich kann man mit dem enable-Pin 9 die output-Pins 11 und 14 aktivieren. Mit den input-Pins 10 und 15 kann man dann die Drehrichtung des zweiten Motors festlegen.

Es genügt, wenn einer der GND-Pins mit dem Arduino verbunden wird. Man kann den VCC1 sowie beide enable-Pins mit einer konstanten 5 V Spannung am Arduino verbinden. Die input-Pins können dann mit Digitalpins verbunden werden, um die Motoren anzusteuern. Wahlweise kann man auch Analogpins (PWM) verwenden, um die input-Pins zu steuern, dann sollte sich auch die Drehgeschwindigkeit regulieren lassen. Echte Analogsignale (ohne PWM) sind hier allerdings nicht sinnvoll, da der Schaltkreis logische Inputs und keine anaolgen Inputs erwartet.

2.6. Schrittmotoren

Im Gegensatz zu Gleichstrommotoren lassen sich Schrittmotoren viel gezielter steuern. Hier dreht sich der Motor nicht kontinuierlich, sondern in exakt vorgegebenen Schritten,

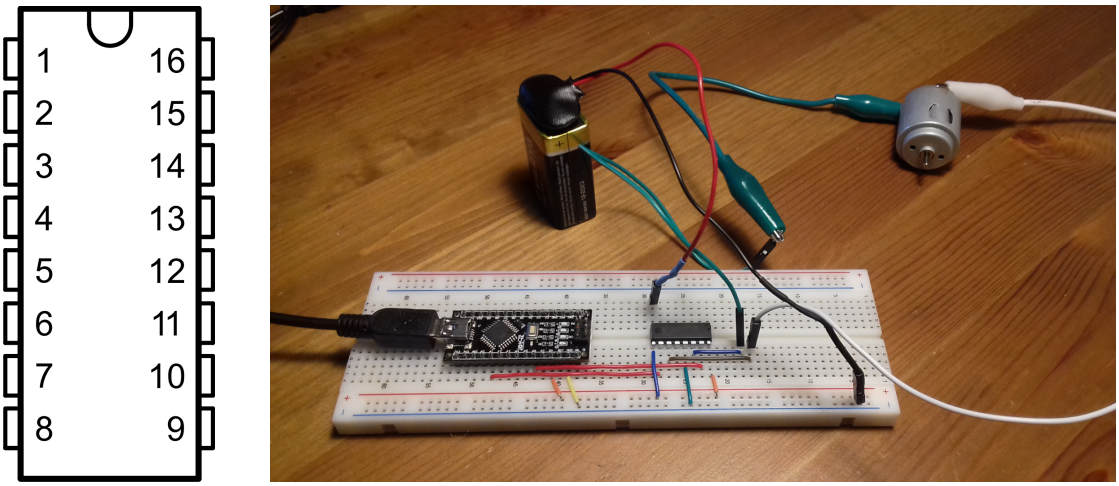


Abbildung 5: Links: Schematischer Aufbau des Schaltkreises L293D; rechts: Schaltung zum Steuern eines Motors.

so dass sich der Drehwinkel (und damit auch die Drehgeschwindigkeit) exakt einstellen lassen. Für den Schrittmotor benötigt man nur digitale Signale, somit ist eine Steuerung über den Raspberry Pi problemlos möglich.

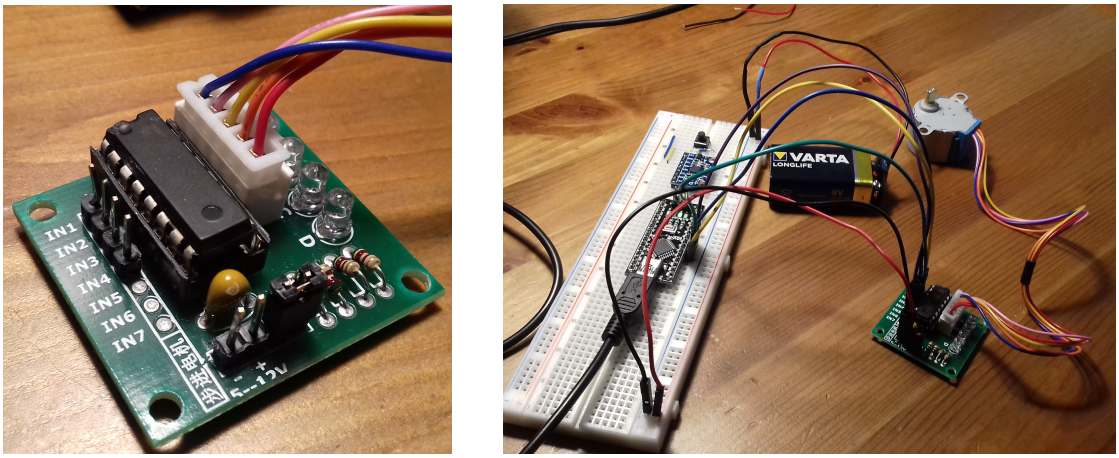


Abbildung 6: Links: Motortreiber mit 4 Inputpins und zwei Pins für die Stromversorgung des Motors; rechts: Schaltung zum Steuern des Motors.

Für die Steuerung eines Schrittmotors kann ebenfalls eine Brückenschaltung verwendet werden. Hierzu gibt es spezielle Motortreiber, die sehr einfach zu bedienen sind (siehe Abbildung 6). Dort sind zwei Pins für die externe Spannungsversorgung vorgesehen. Die vier Inputpins (IN1 bis IN4) werden mit digitalen Pins verbunden. Mit diesen wird der Motor gesteuert. Dabei müssen die vier Pins nach einem gewissen Schema aktiviert und deaktiviert werden, das in Abbildung 7 dargestellt ist.

Den Code zum Steuern des Schrittmotors über den Arduino finden wir in Listing 11. Die Funktion `step` übernimmt eine ganze Zahl als Argument. Diese Zahl gibt zum einen die Pause zwischen zwei Phasen des Drehvorgangs in Millisekunden an (je größer der Betrag der Zahl, um so langsamer dreht der Motor). Das Vorzeichen der Zahl gibt die Drehrichtung an.

	IN1	IN2	IN3	IN4
Phase 1	HIGH	LOW	LOW	LOW
Phase 2	HIGH	HIGH	LOW	LOW
Phase 3	LOW	HIGH	LOW	LOW
Phase 4	LOW	HIGH	HIGH	LOW
Phase 5	LOW	LOW	HIGH	LOW
Phase 6	LOW	LOW	HIGH	HIGH
Phase 7	LOW	LOW	LOW	HIGH
Phase 8	HIGH	LOW	LOW	HIGH

Abbildung 7: Ein Drehschritt ist aufgeteilt in 8 Phasen, die nacheinander abgearbeitet werden.

```
int Pin0 = A2;
int Pin1 = A3;
int Pin2 = A4;
int Pin3 = A5;

void setup()
{
 pinMode(Pin0, OUTPUT);
 pinMode(Pin1, OUTPUT);
 pinMode(Pin2, OUTPUT);
 pinMode(Pin3, OUTPUT);
}

int step(int d)
{
  int p=abs(d);
  if(d>0){
   digitalWrite(Pin0,HIGH);
   delay(p);
   digitalWrite(Pin3,LOW);
   delay(p);
   digitalWrite(Pin1,HIGH);
   delay(p);
   digitalWrite(Pin0,LOW);
   delay(p);
   digitalWrite(Pin2,HIGH);
   delay(p);
   digitalWrite(Pin1,LOW);
   delay(p);
   digitalWrite(Pin3,HIGH);
   delay(p);
   digitalWrite(Pin2,LOW);
   delay(p);
  }
  if(d<0){
```

```
    digitalWrite(Pin3,HIGH);
    delay(p);
    digitalWrite(Pin0,LOW);
    delay(p);
    digitalWrite(Pin2,HIGH);
    delay(p);
    digitalWrite(Pin3,LOW);
    delay(p);
    digitalWrite(Pin1,HIGH);
    delay(p);
    digitalWrite(Pin2,LOW);
    delay(p);
    digitalWrite(Pin0,HIGH);
    delay(p);
    digitalWrite(Pin1,LOW);
    delay(p);
   }
}
void loop()
{
  int n=200;
  for(int i=1;i<=n;i++){
    step(5);
  }
 delay(500);
}
```

Listing 11: Schrittmotorsteuerung für den Arduino.

2.7. Analoges Audiosignal über den Arduino auslesen

Wird Musik über ein Audiokabel ausgelesen, müssen wir folgendes beachten: Das Signal besteht aus einer Wechselspannung, die zwischen zwei Werten $-U_{\max}$ und $+U_{\max}$ schwankt, wobei $U_{\max}$ von dem Gerät abhängt, das die Musik abspielt. Üblicherweise ist $U_{\max} < 2\,\mathrm{V}$ (meine Experimente haben z.B. ergeben, dass am Klinke-Ausgang des Raspberry Pi bei maximaler Lautstärke $U_{\max} = 1.8\,\mathrm{V}$ ist).

Der Arduino kann jedoch nur Signale auslesen, die sich zwischen 0 und 5 V bewegen. Daher müssen wir die Spannung mit einer Offsetschaltung entsprechend verschieben. Zunächst kann es sinnvoll sein, mit einem Spannungsmessgerät die Spannungsamplitude des Audioausgangs herauszufinden. Ist die Amplitude kleiner als 2.5 V, dann reicht es, auf das Signal ein Offset von 2.5 V zu addieren. Da der Arduino selbst einen 5V Spannungsausgang hat, kann man diese Spannung teilen und für das Offset nutzen. Einen Schaltplan dafür finden wir in Abbildung 8.

Jetzt müssen wir das Audiosignal in festen Zeitschritten abgreifen, um es weiter zu verarbeiten. Dazu müssen wir den Zeitschritt `dt` (bzw. die Abtastrate `Ar`, welche der Kehrwert von `dt` ist) festlegen. Wir geben die Größe in Mikrosekunden an und können mit der Funktion `micros()` die aktuelle Zeit in Mikrosekunden abrufen.

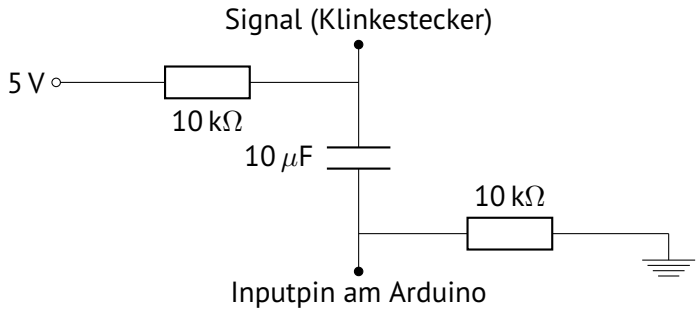


Abbildung 8: Schaltung für ein 2.5 V Offset.

In Listing 12 finden wir ein Codebeispiel. `Serial.begin` und `Serial.println` ermöglichen uns, die Daten auf dem seriellen Monitor auszugeben oder über den seriellen Plotter graphisch darzustellen. Die `while`-Schleife ist dazu da, solange zu pausieren, bis der Zeitschritt `dt` verstrichen ist.

Die Werte, die wir erhalten, liegen zwischen 0 und 1023, wobei der höchste Wert einer Spannung von 5 V entspricht. Will man die Daten weiter verarbeiten, kann man das Offset wieder abziehen, muss also den Wert 512 subtrahieren.

```
unsigned long t;
unsigned int dt;
int signal;
int Ar;
void setup() {
  pinMode(A3,INPUT);
  Ar=8000;
  dt=round(1000000*1.0/Ar);
  Serial.begin(9600);
}

void loop() {
  t=micros();
  signal=analogRead(A3)
  Serial.println(signal);
  while(micros()<t+dt){}
}
```

Listing 12: Ein Audiosignal wird über den Pin A3 eingelesen und auf dem Seriellen Monitor ausgegeben.

3. Projektideen

3.1. Temperatur- und Luftfeuchtigkeit überwachen

Ähnlich wie beim Vorgehen in Abschnitt 2.4.3 kann man selbstverständlich auch mit anderen Sensordaten verfahren. Man könnte damit zum Beispiel eine Wetterstation betreiben.

Eine andere Idee ist es, die Daten in einem geschlossenen Raum zu erheben, um das Raumklima zu beobachten. Möglicherweise ist es sinnvoll, neben Raumtemperatur und Luftfeuchtigkeit auch andere Daten zu bekommen, etwa:

- Wann ist der Heizkörper an oder aus (z.B. mit einem weiteren Temperatursensor direkt am Heizkörper)
- Wann wird der Raum gelüftet (hier gibt es z.B. magnetische Sensoren oder Lichtschranken, die sich an Türen und Fenstern anbringen lassen)
- Gibt es weitere Faktoren, die das Raumklima beeinflussen, z.B. einen Wäschetrockner? Lassen sich Daten dieser Geräte auch mit Sensoren erfassen?

Nun kann man Rückschlüsse ziehen, unter welchen Bedingungen das Raumklima besonders gut oder eher schlecht ist.

3.2. Zeitrafferaufnahmen machen

Der Raspberry Pi kann Kameras über die Kommandozeile steuern. Es gibt eine offizielle Raspberry Pi Kamera, man kann jedoch auch eine übliche USB Webcam anschließen (deren Qualität jedoch meist deutlich schlechter ist). Über einen Cronjob (siehe Abschnitt B.6) kann man die Aufnahmen nach einem vorgegebenen Zeitschema steuern, beispielsweise

- alle 10 Minuten
- jeden Tag um 12 Uhr
- zur vollen Stunde, aber nur am Wochenende
- …

3.2.1. Zeitraffervideos erstellen

Mit einem speziellen Kommandozeilenbefehl kann man dann ein Zeitraffervideo erstellen. Wir gehen davon aus, dass die Bildateien durchnummeriert und nach dem Muster `foto_<nummer>.jpg` benannt sind, wobei `nummer` eine fortlaufende, vierstellige Zahl ist. Der Befehl lautet

```
avconv -y -r 24 -i foto_%04d.jpg -r 24 -vcodec libx264 -q:v 20 video.mp4
```

wobei die Zahl nach der Option -r die Abtastrate (Framerate) angibt, d.h. wie viele Bilder verwendet werden, um eine Sekunde des Videos zu erzeugen.

Um die Fotos automatisch nach dem vorgegebenen Muster zu erzeugen, kann man folgendermaßen vorgehen (im Beispiel wird in jeder Minute ein Foto gemacht):

- Man erstellt einen Cronjob wie in Abschnitt B.6 beschrieben. Dort wird ein shellscript aufgerufen, z.B. über den cronjob
  `* * * * * /home/pi/timelapse/takefoto.sh`
- Im Ordner `timelapse` im Heimverzeichnis des Benutzers `pi` wird die Datei `takefoto.sh` mit dem Code aus Listing 13 erzeugt
- Ebenso wird im selben Ordner die Datei `n.txt` erzeugt, die als einzigen Eintrag eine `0` enthält (hier wird die Nummer gespeichert, die das nächste Foto erhalten soll).
- Schließlich werden die Dateien zum Ausführen freigegeben:
  `chmod +x n.txt takefoto.sh`

Bemerkung: Die Software der Raspberry Pi Kamera hat ebenfalls eine timelapse-Funktion. Der Vorteil der hier vorgestellten Methode ist, dass sie auch über sehr lange Zeiträume

funktioniert. Selbst wenn der Pi zwischendurch ausgeschaltet wird, fährt er mit den Aufnahmen fort, sobald er wieder hochgefahren hat. Für die korrekte Einstellung des Datums und der Uhrzeit (für manche Cronjobs ist das unerheblich) benötigt man jedoch eine Internetverbindung.

```
#!/bin/bash
n=$(cat /home/pi/timelapse/n.txt) #Zahl lesen
n=$(printf "%04d" $n) #fuehrende Nullen ergaenzen
raspistill --nopreview -o /home/pi/timelapse/foto_$n.jpg
n=${n##+(0)} #fuehrende Nullen streichen
n=$(($n+1)) #1 addieren
echo $n > /home/pi/timelapse/n.txt #neuen Wert speichern
```

Listing 13: Der Code erzeugt ein Foto mit der Raspberry Pi Kamera und vergibt Dateinamen mit fortlaufender Nummerierung. Wichtig sind die absoluten Dateipfade, damit das Skript auch als Cronjob ausgeführt werden kann.

3.2.2. Projektideen

- Aufnahme des Sonnenstandes zu einer bestimmten Uhrzeit, um jahreszeitbedingte Unterschiede zu untersuchen (Aufnahme des Sonnenaufgangs ist schwieriger, da der Zeitpunkt variiert).
- Zeitrafferaufnahmen von Pflanzen über mehrere Stunden oder Tage.
- Geschwindigkeitsmessung von Wolken und anderen langsamen Objekten.
- Gut geeignet, um die Fächer Informatik und Kunst zu verbinden.

3.3. Einen Schrittzähler entwickeln

Mit einem Beschleunigungssensor (z.B. dem Beschleunigungs- und Gyrosensor GY-521) kann man Bewegungsdaten messen, die man zur Konstruktion eines Schrittzählers nutzen kann. Eine ausführliche Beschreibung des Projekts findet man im KOMMS-Report Nr. 12 *Produktorientierte mathematische Modellierung am Beispiel eines Schrittzählers*.

3.4. Steuerung einer Lichtorgel / eines Musikbrunnens

Durch die Auswertung von akustischen Signalen oder Daten lassen sich Steuerungen für Lichtorgeln und Musikbrunnen entwickeln. Im Folgenden gibt es einige Anregungen zur Umsetzung dieser Projektidee.

3.4.1. Das akustische Signal einlesen

Um an das akustische Signal zu kommen, sind mehrere Herangehensweisen denkbar:

- **Mikrofon am Arduino**: Ein Mikrofon erzeugt ein analoges Signal, das am Arduino eingelesen werden kann.
- **Mikrofon am Raspberry Pi**: Mit einer externen USB-Soundkarte lassen sich Mikrofone mit Klinkestecker direkt an den Raspberry Pi anschließen. Ein solches Signal könnte z.B. über Python oder eine andere Programmierumgebung ausgelesen werden.

- **Signal am Klinkestecker abgreifen**: Wir können die Methode verwenden, die in Abschnitt 2.7 beschrieben ist.
- **Musikdateien verwenden**: Wir können eine Musikdatei (z.B. im `wav`-Format) in Python einlesen (siehe Listing 14).

```
from scipy.io import wavfile
[Ar,x]=wavfile.read('beispiel.wav')
```

Listing 14: Der Code liest eine `wav`-Datei ein. Die Audiodaten werden in der Variable `x` gespeichert (evt. zwei Kanäle), die Abtastrate in `Ar`.

3.4.2. Das Signal verarbeiten

Will man mit den Daten im Amplitudenbild arbeiten, so kann man direkt loslegen. Um die Frequenzspektren zu analysieren, kann eine Fouriertransformation hilfreich sein. In Python lässt sich das mit dem Code aus Listing 15 realisieren.

```
from scipy.fftpack import fft
#kanal=Audiodaten aus einem der beiden gelesenen Audiokanaele
frequenzen=fft(kanal)
```

Listing 15: Berechnung des Frequenzspektrums in Python.

Auch für den Arduino gibt es Bibliotheken, die eine Fouriertransformation ermöglichen. Beispielsweise kann die Bibliothek `arduinoFFT.h` verwendet werden.

Darüberhinaus ist es natürlich auch denkbar, einen Frequenzfilter zu konstruieren, der das eingelesene Signal direkt weiterverarbeitet, bevor es vom Gerät eingelesen wird.

3.4.3. Das Projekt realisieren

Mit den eingelesenen Daten wird nun das mathematische Modell gefüttert, das die Grundlage für die Steuerung bildet. Schließlich kann man verschiedene Projekte realisieren:

- **Modell einer Lichtorgel**: Die Lichtorgel lässt sich modellhaft mit kleinen LED-Lämpchen konstruieren. Die Spannungsversorgung der Arduinopins würde z.B. ausreichen, um die Lampen zu steuern, und per PWM kann auch die Helligkeit reguliert werden.
- **Eine richtige Lichtorgel konstruieren**: Hier sind größere Spannungen für die Lampen notwendig, so dass eine komplexe Steuerung mit Transistorschaltungen zum Einsatz kommen kann.
- **Einen Musikbrunnen konstruieren**: Dies ist ein größeres Projekt: Mit Wasserpumpe und steuerbaren Ventilen (z.B. Magnetventilen) kann ein Wasserspiel gesteuert werden, das zur Musik passt.
- **Steuerung einer Simulation**: Selbstverständlich lässt sich am Raspberry Pi (evt. auch am Arduino mit einem kleinen TFT-Display) eine grafische Simulation erzeugen.

3.5. Einen 3D Scanner konstruieren

Wir stellen zwei Herangehensweisen vor, wie ein solcher 3D Scanner funktionieren kann. Bei der ersten ist die technische Komponente etwas stärker im Fokus, da das Messinstru-

ment (oder wahlweise die Probe) wesentlich komplizierter bewegt werden muss. Im zweiten Beispiel kommen Methoden der Bildverarbeitung zum Einsatz und es ist ein gewisses Grundverständnis projektiver Geometrie nötig.

3.5.1. Abstandssensoren

Hier wird die Probe mit einem Abstandssensor abgetastet. Die Sensordaten müssen dann mit den relativen Positionen der Probe und des Sensors zueinander in Verbindung gebracht werden. Dann können die Daten in ein 3D Objekt umgerechnet werden. Im folgenden stellen wir stichpunktartig einige Erfahrungen vor.

- Ultraschallsensoren sind in den Arduino- und Raspberry Pi Lernsets sehr geläufig und zu deren Steuerung findet man viele Informationen im Internet. Die Genauigkeit ist jedoch nur für sehr große Objekte geeignet.
- Alternativ lassen sich auch Infrarotsensoren einsetzen, bei denen eine größere Messgenauigkeit denkbar ist.
- Da man mit den Messgeräten eine Oberfläche scannen will, hat man zwei Freiheitsgrade und benötigt zwei Motoren. Beispielsweise kann man die Probe auf einem rotierenden Teller platzieren und den Sensor entlang der Höhenachse bewegen.
- Schrittmotoren sind hier sinnvoll, da wir hier den Drehwinkel exakt einstellen können.

In Abbildung 9 sehen wir den Scanner, der bei der Modellierungswoche im Februar 2018 gebaut wurde.

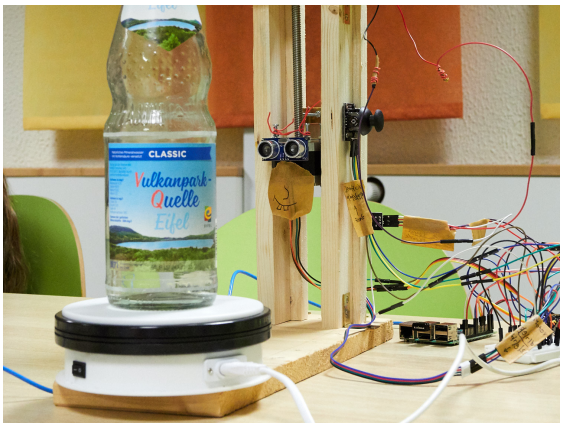


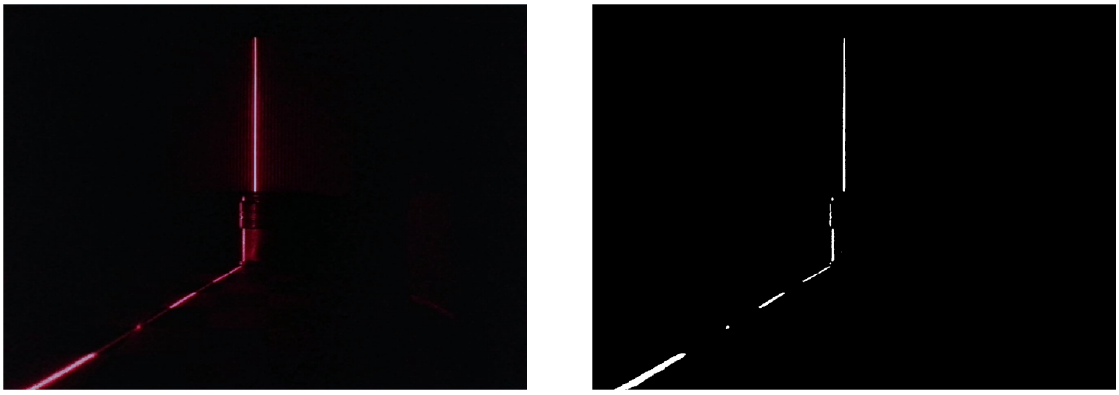
Abbildung 9: Ein 3D Scanner bestehend aus einem Ultraschallsensor, einem Drehteller für die Probe und einem Schrittmotor mit Gewindestange für den Sensor (Foto: Martin Bracke).

3.5.2. Streifenprojektion

Beim Prinzip der Streifenprojektion wird ein Lichtstreifen auf die Probe projiziert. Wird die Probe aus einem anderen Blickwinkel fotografiert, dann lässt sich die Krümmung an der Oberfläche der Probe visualisieren.

Dazu wird mit Methoden aus der Bildverarbeitung das Streifenmuster lokalisiert (siehe Abbildung 10 und Listing 16). Die Koordinaten der berechneten Pixel müssen nun in die

Koordinaten des 3D Objekts umgerechnet werden.



(a) Foto der Streifenprojektion …

(b) …und bearbeitetes Bild.

Abbildung 10: Foto und Verarbeitetes Bild bei der Streifenprojektion.

```
import numpy as np
import cv2
img = cv2.imread("streifenprojektion.png")
hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

# Range for laser
lower = np.array([0,0,120])
upper = np.array([255,255,255])
mask = cv2.inRange(hsv, lower, upper)

cv2.imshow('mask',mask)
```

Listing 16: Pythoncode zur Aufteilung in helle und dunkle Pixel.

3.6. Ein Segwaymodell steuern

Mit zwei Elektromotoren und einem Gyrosensor kann man ein Modell konstruieren, an welchem sich die Steuerung eines Segway erproben lässt. Das Prinzip ist simpel: neigt sich das Segway in eine Richtung, müssen die Elektromotoren in die andere Richtung gegensteuern. Wenn diese Steuerung gut eingestellt ist, kann sich das Segwaymodell selbst stabilisieren. Durch Verlagerung des Schwerpunkts fährt das Modell nach vorne bzw. nach hinten.

Das Arbeiten mit dem Gyrosensor GY-521 ist im KOMMS-Report Nr. 12 *Produktorientierte mathematische Modellierung am Beispiel eines Schrittzählers* ausführlich erläutert.

A. Grundbegriffe Elektronik

A.1. Analog und Digital

Bei der Datenübertragung zwischen Raspberry Pi und Arduino mit Sensoren und anderen Geräten ist immer wieder die Rede von **analogen** und **digitalen** Signalen.

Physikalisch gesehen ist der Unterschied, dass analoge Signale Spannungswerten von 0 bis üblicherweise 5 V entsprechen[10], während digitale Signale nur die Werte **keine Spannung liegt an** (LOW, 0) und **Spannung liegt an** (HIGH, 1) kennen. Ein Informationsaustausch über digitale Signale erfolgt daher über eine Serie aus Nullen und Einsen, die gemäß eines vereinbarten Protokolls in eine verwertbare Information umgerechnet werden können.

Achtung: Bei beiden Geräten ist das analoge Ausgangssignal nicht wirklich ein Spannungswert zwischen 0 und 5 V, sondern es entsteht aus einer sogenannten **Pulsweitenmodulation (PWM)**. Dabei wird ein digitales Signal mit einer hohen Frequenz ein- und wieder ausgeschaltet. Man erhält also ein periodisches Signal aus Rechteckpulsen, bei der die Zeitdauer der Phase *Eingeschaltet* variiert werden kann. Der Mittelwert der Spannung während einer Periode kann als Stärke eines analogen Signals interpretiert werden.

A.2. Wenn 5 V zu wenig sind

Manchmal will man Geräte steuern, die deutlich höhere Spannungen benötigen, als die üblichen 5-12 V, die Arduino und Raspberry Pi liefern können. In diesem Fall hat man zwei Möglichkeiten: Man nutzt eine externe Spannungsquelle (Batterie, Netzteil, etc.) für den Betrieb des Geräts und schaltet diesen Stromkreis über ein Relais oder über Transistoren ein und aus.

A.2.1. Relais

Ein Relais ist ein Schalter, der über eine angelegte Spannung geöffnet oder geschlossen werden kann. In diesem Fall benötigt man ein Relais mit einer Schaltspannung von 5 V, damit auch die kleineren Arduinos und der Pi das Relais schalten können.

A.2.2. Transistor

Transistoren sind vielseitiger als Relais. Im Prinzip kann man durch das anlegen einer relativ kleinen Spannung den Widerstand des Transistors regulieren. Wird der Widerstand hoch reguliert, dann sperrt der Transistor (der gesteuerte Stromkreis ist außer Betrieb). Wird der Widerstand herunterreguliert, dann kann ein Strom durch den Transistor fließen und das gesteuerte Grät wird aktiv.

Durch Variation der angelegten Spannung kann auch die Spannung, die am Transistor abfällt, reguliert werden. Damit ist der Transistor nicht einfach ein *Ein- und Ausschalter*. Man kann ihn auch verwenden, um beispielsweise die Drehgeschwindigkeit eines Elektromotors zu ändern.

[10] Diese können dann in eine Zahl umgewandelt werden, z.B. entspricht beim Arduino ein 5 V Eingangssignal einem Wert von 1023, ein 2,5 V Eingangssignal entspricht dem Wert 511 usw.

Transistoren schalten schneller als Relais, sind aber auch empfindlicher und möglicherweise bei sehr hohen Strömen und Spannungen nicht geeignet.

B. Crashkurs Linux

Unter dem Begriff *Linux* fasst man eine Reihe von UNIX-artigen Betriebssystemen zusammen. Bekannte Linux-Distributionen sind beispielsweise OpenSuse, Ubuntu, Debian oder Linux Mint. Das Standardbetriebssystem für den Raspberry Pi ist Raspberry Pi OS (ehemals Raspbian), das, ähnlich wie Ubuntu, an Debian angelehnt ist.

Linux hat viele Gemeinsamkeiten mit macOS, das auch auf UNIX basiert. Viele der hier erwähnten Besonderheiten finden daher auch auf Apple-Rechnern Anwendung.

Das vielleicht wichtigste Feature von Linux ist die Steuerbarkeit über die Kommandozeile. Dort kann man durch die Eingabe von Textbefehlen arbeiten, benutzt also keine Maus und keine grafische Oberfläche.

B.1. Die Kommandozeile

Die Kommandozeile (auch shell, terminal, command line) ist eine Arbeitsumgebung, in der das Betriebssystem lediglich durch die Eingabe von Textbefehlen gesteuert wird. Alte Hasen erinnern sich vielleicht noch an MS Dos, das zur Zeit von Windows 3.11 unverzichtbar war. Die Linux-Kommandozeile sieht zunächst sehr ähnlich aus.

Zu jedem Zeitpunkt befindet sich die Kommandozeile in einem Ordner des Dateibaums. In aller Regel startet man in seinem eigenen Home-Verzeichnis

`/home/<benutzername>`

Das Wurzelverzeichnis (vergleichbar mit `C:` unter Windows) wird einfach mit einem / bezeichnet. Unterordner werden als Text angehängt und mit weiteren / voneinander separiert, z.B.

`/home/pi/meinordner/unterordner`

Um in einen Unterordner zu wechseln, verwendet man den Befehl

`cd <ordnername>`

Um in den übergeordneten Ordner zu gelangen, verwendet man

`cd ..`

Man kann Dateien verschieben mit

`mv dateiname ziel`

Man kann Dateien kopieren mit

`cp dateiname ziel`

Und man kann Dateien löschen mit

`rm dateiname`

Neue Ordner erstellt man mit

`mkdir ordnername`

Da die Linuxsysteme über eine sehr aktive Onlinecommunity verfügen, findet man auch zahlreiche Hilfestellungen und Tutorials im Internet.

B.2. Der Texteditor nano

Häufig muss man Textdateien bearbeiten, sei es, um Konfigurationsdateien zu ändern, oder wenn man einen Programmcode schreiben will. Dazu kann man den Befehl

```
nano <dateiname>
```

verwenden und öffnet dann die ausgewählte Datei in einem Schreibmodus. Wenn die Datei noch nicht existiert, wird sie neu angelegt.

Die wichtigsten Befehle, die man beim Bedienen von Nano braucht, sind die folgenden:

STRG + O	Datei speichern
STRG + X	nano verlassen
STRG + W	nach Text suchen

B.3. Rechteverwaltung

Linux besitzt ein sehr striktes Rechtesystem, das den Zugriff auf Dateien für verschiedene Benutzergruppen steuert. Eine Datei hat immer einen Besitzer, das kann ein registrierter Benutzer sein, eine Gruppe von Benutzern oder eine Systemanwendung, z.B. ein Server). Außerdem wird die Datei einer Gruppe zugewiesen (im einfachsten Fall ist das auch eine einzelne Person, z.B. der Benutzer, der die Datei erstellt hat).

Diese Unterscheidung ist wichtig, weil man die Zugriffsrechte für die Datei in drei Stufen strukturieren kann: Es gibt Rechte für den `user` (das ist der Besitzer der Datei), Rechte für die `group` und Rechte für `others` (d.h. alle anderen Benutzer oder Gruppen, die der Datei NICHT zugewiesen wurden).

Die Rechte, die ein Nutzer haben kann, sind Lesen (`r`), Schreiben (`w`) und Ausführen (`x`).

B.3.1. Dateien vs. Ordner

Für Dateien sind die Rechte selbsterklärend: `r` bedeutet, dass ich mir den Inhalt einer Datei ansehen kann, `w` bedeutet, dass ich die Datei verändern kann, `x` bedeutet, dass ich die Datei ausführen kann (z.B. einen Programmcode).

Für Ordner bedeuten die Buchstaben etwas anderes: `r` bedeutet, dass ich mir den Inhalt des Ordners ansehen kann (z.B. über den Befehl `ls`). `w` bedeutet, dass ich Dateien speichern, kopieren oder löschen kann. `x` bedeutet, dass ich in den Ordner wechseln und mit den Dateien und Unterordnern arbeiten kann.

B.3.2. Rechte von Dateien auslesen

Mit dem Befehl

```
ls -al
```

kann man sich die Rechte seiner Dateien anschauen. Dabei erhält eine Liste mit Einträgen der folgenden Art:

```
drwxr-xr-x 5 capraro staff 160 10 Jul 2019 skripte
```

Der erste Block (`drwxr-xr-x`) bezeichnet dabei die Zugriffsrechte. Das vorangestellte `d` bedeutet, dass es sich um einen Ordner (directory) handelt. Dann kommen drei Dreierblöcke: Der Erste steht für den Besitzer der Datei, der Zweite für die Gruppe, und der Dritte für alle anderen. Dabei bedeuten in diesem Beispiel:

- `rwx`: Der Besitzer hat alle Rechte (Lesen, Schreiben, Ausführen)
- `r-x`: Die Gruppe darf Lesen und Ausführen
- `r-x`: Auch alle anderen dürfen Lesen und Ausführen

Außerdem sehe ich, dass die Datei dem Benutzer mit Namen `capraro` gehört und der Gruppe `staff` zugewiesen ist.

B.3.3. Rechte verändern

Mit dem Befehl `chown` kann man den Besitzer ändern. Beispielsweise kann ich mit

```
sudo chown max bild.png
```

die Datei `bild.png` dem Benutzer `max` zuordnen. Wichtig ist hier der Befehl `sudo`, der dafür sorgt, dass `chown` mit Administratorrechten ausgeführt wird. Nur der Administrator darf den Besitzer einer Datei ändern.

Mit `chgrp` lässt sich ganz ähnlich auch die Gruppe ändern. Hier ist kein `sudo` notwendig.

Mit `chmod` schließlich kann man die Rechte an einer Datei ändern. Es sind mehrere Varianten möglich. Hier einige Beispiele:

- `chmod u=rw bild.png` gibt dem Besitzer das Rechteschema `rw-`
- `chmod g=rwx,o=-- bild.png` gibt der Gruppe alle Rechte, den Anderen keine
- `chmod u+x bild.png` ermöglicht dem Benutzer, die Datei auszuführen
- `chmod +x bild.png` ermöglicht Allen, die Datei auszuführen
- `chmod 755 bild.png` setzt die Rechte mit Oktalzahlen (siehe unten)

Wenn man die Rechte mit Oktalzahlen kodieren will, hat man folgende Bedeutungen:

0 = keine Rechte	`1=--x`	`2=-w-`	`3=-wx`
`4=r--`	`5=r-x`	`6=rw-`	`7=rwx`

Die rechte werden nun durch drei Oktalzahlen beschrieben: `755` bedeutet, dass der Besitzer die Rechte `rwx` hat, Gruppe und Andere haben `r-x`.

Achtung: Ausführbare Programme (z.B. Pythonprogramme oder Shellskripte) sollten immer (zumindest für den Besitzer) die `x`-Berechtigung haben.

B.4. ssh

Der Befehl `ssh` steht für **s**ecure **sh**ell und ermöglicht dem Benutzer, sich auf einem entfernten Rechner einzuloggen und dort Kommandozeilenbefehle auszuführen. Dazu sollte man sich im selben Netzwerk befinden und man benötigt eine Adresse des Hosts (z.B. die IP Adresse oder den Hostnamen, siehe Abschnitt 1.4.1). Der Befehl lautet

```
ssh <benutzername>@<adresse>
```

Bei bekannter IP Adresse könnte das so aussehen

```
ssh pi@10.0.0.1
```

Oder man verwendet den Hostnamen

```
ssh capraro@komms1.local
```

B.4.1. ssh unter Windows

Will man sich von einem Windowssystem auf dem Pi einloggen, dann benötigt man die Software PuTTY, die kostenlos im Internet verfügbar ist. Man kann das fertig kompilierte `exe` file herunterladen, so dass man ohne Installation loslegen kann. In Abbildung 11 sehen wir das Eingabefenster.

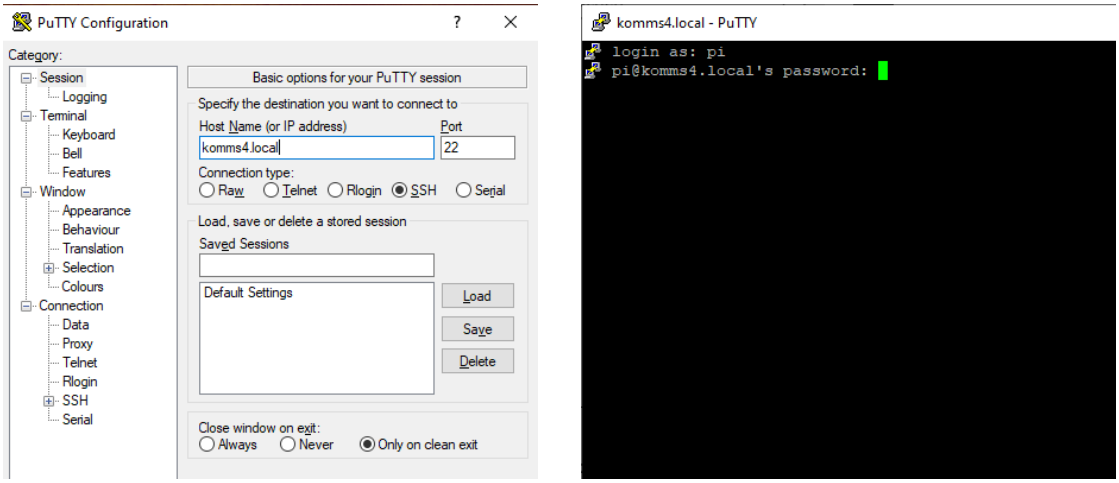


Abbildung 11: Einloggen auf Windows über Putty.

B.5. Programme von der Kommandozeile ausführen

Programme (Shellskripte, Python, C, etc.) können direkt von der Kommandozeile ausgeführt werden. Dazu ruft man einen Befehl auf, der zur jeweiligen Programmiersprache passt, und gibt anschließend die Datei an, die ausgeführt werden soll.

Beispielsweise kann man ein Pythonprogramm, das in der Datei `meincode.py` gespeichert ist, mit dem Befehl

```
python pfad/meincode.py
```

aufrufen. Ebenso kann man Kommandozeilenbefehle aus einem Skript heraus ausgeführen. Liegt der Code in der Textdatei `dateiname.sh`, dann muss man den Befehl

```
sh pfad/dateiname.sh
```

ausführen.

Der Dateipfad kann dabei absolut angegeben werden, z.B.

```
/home/pi/pythonskripte/meincode.py
```

oder man gibt relative Pfade an, z.B.

```
meincode.py
```

wenn man sich im selben Ordner befindet,

```
pythonskripte/meincode.py
```

wenn man sich im Ordner `/home/pi` befindet, oder

```
../meincode.py
```

wenn man sich in einem Unterordner von `/home/pi` befindet.

Achtung: Es ist möglich, dass man die Datei zunächst über die Rechteverwaltung (siehe

Abschnitt B.3) ausführbar machen muss.

B.5.1. Kompilierte Sprachen

Bei kompilierten Sprachen (z.B. C oder Java) muss zunächst eine ausführbare Datei erzeugt werden, die dann aufgerufen werden kann. Für die Sprache C führt man den Befehl

```
gcc pfad/meincode.c -o pfad/meinprogramm
```

aus, um die Datei `meincode.c` zu kompilieren. Dabei wird die ausführbare Datei `meinprogramm` erzeugt.

Um einen Javacode aus der Datei `meincode.java` zu kompilieren, wird

```
javac pfad/meincode.java
```

ausgeführt. Dabei wird ebenfalls eine ausführbare Datei erzeugt.

B.6. Cronjobs

Wir können einen Kalender erstellen, der Programme an bestimmten Tagen und zu bestimmten Uhrzeiten aufruft – sogenannte Cronjobs. Dazu ruft man auf der Kommandozeile den Befehl

```
crontab -e
```

auf. Man wird in eine Textdatei weitergeleitet, wo man am Ende den Code ablegen kann, der ausgeführt werden soll. Das Schema sieht wie folgt aus:

```
<Minute> <Stunde> <Tag> <Monat> <Wochentag> <befehl>
```

(beim Wochentag wird Sonntag mit einer 0 angegeben). Der folgende Befehl wird immer am ersten Tag des Monats um 00:00 ausgeführt, allerdings nur in der ersten Jahreshälfte und wenn der Tag ein Montag ist:

```
0 0 1 1-6 1 /home/pi/meinskript.sh
```

Bei der Angabe gibt es einige Besonderheiten:

- Will man mehrere Optionen für einen Eintrag angeben (z.B. Montag bis Freitag), dann kann man mit `-` einen Bereich angeben (`1-5`) oder man macht eine Aufzählung mit Kommata (`1,2,3,4,5`). Man kann auch beides kombinieren (`1-2,4-5`).
- Will man für einen Eintrag alle Optionen zulassen (z.B. zu jeder Stunde), dann kann man einen Asterisk (*) verwenden.
- Man kann Schrittweiten angeben. *Jeden dritten Tag* wäre z.B. `*/3`. Oder *alle geraden Tage in der ersten Monatshälfte* wäre `1-15/2`.

B.6.1. Cronjob beim Booten ausführen

Dazu muss `@reboot` vorangestellt werden, statt der zeitlichen Angabe, z.B.

```
@reboot /home/pi/meinskript.sh
```

C. Bibliotheken in der Arduino IDE laden

Um eine Bibliothek zu laden, wählt man *Sketch – Bibliothek einbinden – Bibliotheken verwalten* (siehe Abbildung 12). Anschließend kann man eine Schlagwortsuche nach der ge-

wünschten Bibliothek durchführen.

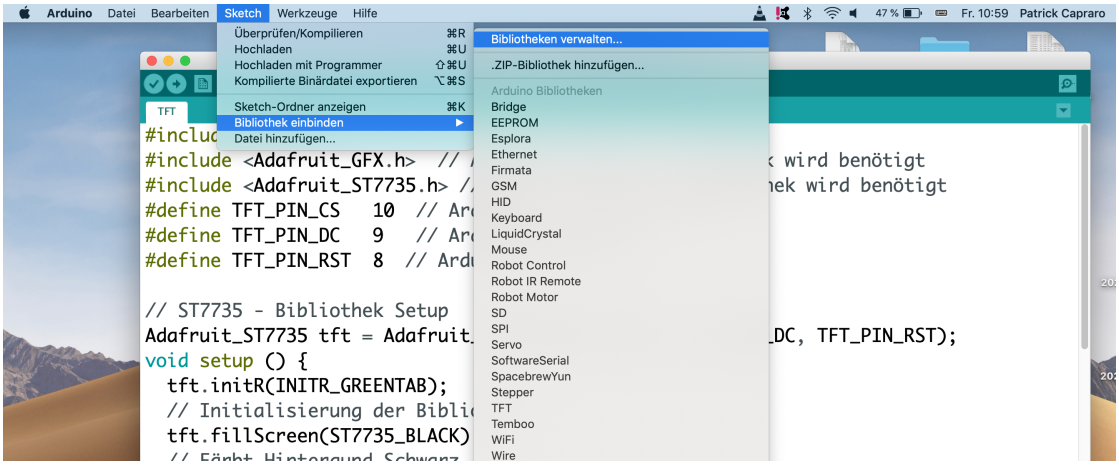


Abbildung 12: Eine Bibliothek in der Arduino IDE laden.