Raspberry Pi Basis-Workshop

Teil 1

- Was ist ein Raspberry und wozu ist er gut?
- Wie installiere ich ein System?
- Einrichtung und Zugriff via SSH
- Python, diese Schlange!
- Grundlegende GPIO-Funktionen mit Python

Teil 2

- Git(-Hub)
- WiringPi & Programmieren in C
- Makefiles
- Service & Bash
- Desktop-Rechner

Was ist ein Raspberry?

- Einplatinen-Computer
- "Arduino in Linux" - Achtung: 3,3V Logik, nicht 5V tolerant!
- Ursprünglicher Gedanke: günstige und spielerische Entwicklungsumgebung für Schüler
- "Missbrauch" von Server, Mediencenter bis Spielekonsole

Systeminstallation

- Verschiedene Systeme zum Download auf http://www.raspberrypi.org/downloads/

- Unter Windows Kopie mit Win32DiskImager http://sourceforge.net/projects/win32diskimager/

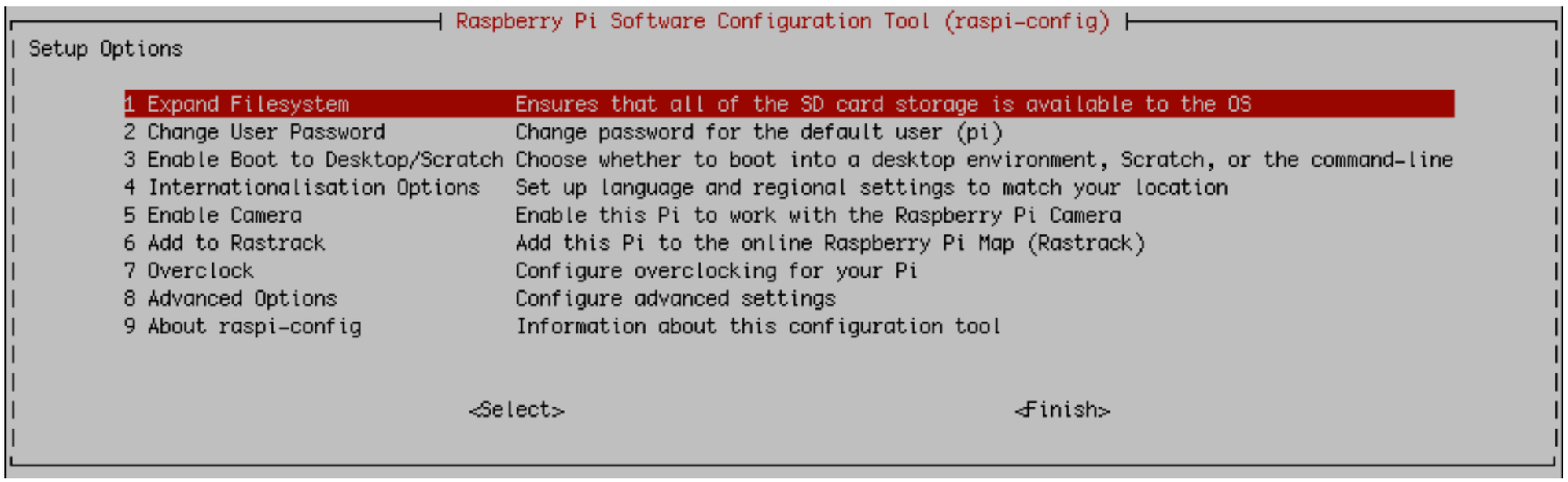
- Unter Mac und Linux
  `$dd if=`*`Image.img`*` of=/dev/`*`SD-Karte`*` bs=4M`
  Mac: **`bs=4m`**

- Alternativ werden fertige SD-Karten angeboten

Einrichten des Raspberry

- Tool raspi-config jederzeit aufrufbar
- SSH von Haus aus aktiviert
- Jetzt die ersten Schritte:
  - Verbindung mittels Putty
    IP-Adresse des Pi, Benutzer pi, Passwort raspberry
  - Verbindung mittels SSH (Linux/Mac)
    `$ssh pi@`*`IP`*

Raspi-Config



- Immer zuerst die Partition auf die gesamte SD-Karte mittels Expand Filesystem ausdehnen!

- Tastatur auf Deutsch stellen
  `4 -> I3 -> Enter -> Other -> German`

- Optional System auf Deutsch `4 -> I1 -> de_DE.UTF-8`

- `<Finish> -> yes` - REBOOT!

Python

- Name eigentlich von Monty Python

- Gut zum Lernen von sauberem Programmieren, da es keine Klammern gibt, sondern alles über Einrückungen gemacht wird.
  Achtung! Nicht Leerzeichen und Tabs vermischen!

- Variablen haben keine Datentypen

- Anderer Syntax bzw. Datentypen, z.B.

```
for(int i=0;i<10;i++){
for i in range(10):
```

range(10) = range(0,10) = [0,1,…,9],
also Iteration über eine Liste

Beispielscript
LED und Taster

- Wieder zurück zum Raspberry via SSH

- Installation der GPIO-Funktionen:

```
$sudo apt-get update  #Update der Paket-Datenbank
$sudo apt-get install python-dev python-rpi.gpio
```

- Anlegen eines neuen Scripts test.py:

```
$nano test.py
```

test.py

```
from time import sleep      #Nur sleep importieren
import RPi.GPIO as GPIO     #Paket importieren und
Namensraum "GPIO" statt RPi.GPIO nutzen
GPIO.setmode(GPIO.BOARD)    #Pin-Nummern
#Alternative wäre GPIO.BCM für GPIO-Namen
GPIO.setup(15,GPIO.OUT)     #LED am Pin 15/GPIO 22
GPIO.setup(16,GPIO.IN, pull_up_down=GPIO.PUD_UP)
#Taster am Pin 16/GPIO 23 mit internem Pull Up
while True:                 #Endlosschleife
»if (GPIO.input)==GPIO.LOW:#Taster gedrückt
»»GPIO.output(15,GPIO.HIGH)#LED an
»»sleep(1)                  #1s warten
»»GPIO.output(15,GPIO.LOW) #LED aus
#Speichern und Beenden mit CTRL+X (^C)
» = entweder Tab oder Leerzeichen, könnte jede
Ebene aber auch geändert werden, also:
[Tab]Command:
[Tab][Space]Command
```

Verkabeln
& Programm testen

- Taster an GPIO 23 (BCM)/Pin 16 und GND

- LED mit Widerstand (je nach LED) an GPIO 22/
  Pin 15 mit langem Bein (+), kurzes Bein (-) an GND

- Testen mit:
  ```
  $python test.py
  ```

- Schlägt fehl: GPIOs können nur mit Administrator-Rechten (root) verändert werden

- Den gleichen Befehl noch mal als root:
  ```
  $sudo !! #!! wird durch vorigen Befehl ersetzt, also
  $sudo python test.py
  ```

- Beenden mit CTRL+C (^C)

Pause!

Teil 2

Die Annäherung an die Arduino-Schreibweise

Git

- Git wird hauptsächlich assoziiert mit GitHub
- Datencontainer
- Versionskontrollsystem und kostenlose Open-Source-Projekt-Cloud
- Vorinstalliert, ansonsten Installation mittels
  `$sudo apt-get install git-core`

Installation von WiringPi

- Auf meinen Raspberrys teilweise vorinstalliert:
  Überprüfen mittels `ls ~/wiringPi`

- "Klonen" der aktuellen Version der Git
  `$git clone git://git.drogon.net/wiringPi`

- In das heruntergeladene Verzeichnis wiringPi wechseln
  `$cd wiringPi #Change Directory`

- Installieren
  `$./build #führt das Skript build im aktuellen Verzeichnis >./< aus`

Sonstige Git-Funktionen

- git init - erstellt neue Repository im Verzeichnis
- git pull - "zieht" sich die neueste Version von externer Git
- git fetch - synchronisiert lokale und externe Git
- git status [-s] - Status der Dateien [in Kurzfassung]
- git add x y - fügt x und y zur Repository
- git diff - Zeigt alle Änderungen seit letztem Commit
- git commit - vermerkt Änderungen zur Vorversion
- git push - "drückt" Änderungen in die externe Git

Schreiben eines C-Programms

- Zuerst wieder ins Benutzerverzeichnis wechseln:

```
$cd ../ #Ein Verzeichnis hoch, da wir ja nur eins bisher tiefer gegangen sind
$cd ~/ #Wechselt immer ins Benutzerverzeichnis
$nano test.c #Datei test.c bearbeiten
```

test.c

```
int main(int argc, char *argv[]) {
 wiringPiSetupPhys(); #gleiche Pin Nummern wie vorhin
 #ich bevorzuge wiringPiSetup();
 pinMode(15,OUTPUT);
 pinMode(16,INPUT);
 pullUpDnControl(16,PUD_UP);
 while (1) {
  if (!digitalRead(16)) {
   digitalWrite(15,HIGH);
   delay(1000);
   digitalWrite(15,LOW);
}}}
```

Kompilieren und ausführen

- Kompilieren mit gcc:
  `$gcc test.c -lwiringPi -o ledTest`

- Wird `-o ledTest` weggelassen, so wäre das resultierende Programm `a.out` benannt.

- Ausführen mit
  `$sudo ./ledTest`

- Beenden mit `CTRL+C`

Makefiles

- Anstatt jedes Mal die GCC-Befehle einzugeben einfach nur "machen"!

- `make` guckt, ob und was sich geändert hat und kompiliert nur das nötigste.

- Makefile anlegen mit
  `$nano Makefile`

Makefile

```
ledTest:test.c #ledTest hängt von Änderungen bei test.c
ab
 gcc test.c -lwiringPi -o ledTest
#wird ausgeführt, wenn sich test.c ändert

install: #"Installiert" ledTest
 mv ledTest /usr/local/bin #verschiebt die Datei

clean: #Räumt auf
 rm -f ledTest

#Möglich wären noch weitere Abhängigkeiten, wenn das
#Programm aus mehreren Dateien zusammengesetzt ist,
#in diesem Falle wäre /usr/lib/wiringPi.h eine halbwegs
#dynamische Abhängigkeit
#Einrücken muss durch TAB erfolgen!
```

Machen

- Um das Programm zu kompilieren einfach nur
  `$make`
  eingeben

- Zum "Installieren"
  `$sudo make install`

- Ab dann kann es von überall einfach ausgeführt werden:
  `$sudo ledTest`

Service einrichten

- Services sollten nur für Dämonen eingerichtet werden
  - Dämon = im Hintergrund laufendes Programm
    Dazu mehr gleich.
  - Kurzweilige Programme sind auch akzeptabel (z.B. "led an, fertig.")
  - Services liegen in `/etc/init.d/`
  - Neuen Service erstellen:
    `$nano /etc/init.d/ledTest`

ledTest

```
#! /bin/bash
### BEGIN INIT INFO
# Provides: ledTest
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
### END INIT INFO

case "$1" in
start)
 /home/pi/ledTest
esac
exit 0
```

Service starten und beim Hochfahren automatisch ausführen

- Ein Service kann mittels
  `$service ledTest start`
  ausgeführt werden

- Um den Service beim Hochfahren zu starten
  `$update-rc.d ledTest defaults`

Dämon

```
#include<unistd.h>
#include<stdlib.h>
/*Includes, Variablen, Funktionen…*/
main() {
 int pid;
 /*Setupkram, sollte nicht blocken*/
 pid = fork(); /*Prozess klonen*/
 if (pid < 0) {/*Klonen nicht erfolgreich*/
  exit(EXIT_FAILURE);
 } else if (pid > 0) {/*Klonen erfolgreich
  pid > 0 = Vaterprozess, also den,
  den wir aufgerufen haben*/
  exit(EXIT_SUCCESS);
 }
 /*pid == 0, Kindprozess, also unser Dämon*/
 /*sonstiger Programmcode*/
}
```

Desktop Rechner

- Um die normale Desktop Oberfläche zu bekommen, einfach
  $startx
  eingeben

- Kann auch bei jedem Boot automatisch zum Desktop wechseln, dazu in der **`raspi-config`**
  `3 Enable Boot to Desktop/Scratch`
  `Desktop Log in as user 'pi'…`
  aktivieren

Danke für eure Aufmerksamkeit!

Der Raspberry lässt sich jetzt mittels

`$sudo halt`

herunterfahren