Vereinfachen von Workshop-Szenarien mit Raspberry Pi durch Vorbereitung von SD-Karten-Abbildern

Malte Heidenreich

13. Januar 2017

Zusammenfassung

Dieser Bericht beschreibt, wie die technische Infrastruktur für Workshops mit Raspberry Pis vereinfacht werden. Es werden Abbilder für SD-Karten auf einem PC gebaut, die alle nötigen Programme für den Workshop beinhalten.

1 Einleitung und Motivation

Für die Vorbereitung eines Workshops benötigt man viel Zeit. In diesem Fall sollen Raspberry Pis und Arduinos verwendet werden. Um einen Raspberry Pi in Betrieb nehmen zu können, muss neben der erforderlichen Hardware auch eine SD-Karte mit dem Betriebssystem vorhanden sein. Dieses wird von der Raspberry Pi Foundation oder anderen Quellen bereitgestellt. Bei Workshops mit vielen Teilnehmern dauert es sehr lange, um alle benötigten SD-Karten mit dem Betriebssystem zu beschreiben und die Vorbereitungszeit erhöht sich demnach. Sollten noch Anpassungen an der Installation vorgenommen werden, müssen diese direkt auf einem Raspberry Pi stattfinden. Anpassungen können in diesem Fall Softwareinstallationen sein, die auf dem Raspberry Pi sehr lange dauern.

So bietet es sich an, das Abbild der SD-Karte inklusive aller Anpassungen auf einem schnelleren Rechner vorzubereiten und anschließend auf die Karten zu schreiben. Aus diesem Vorgehen ergibt sich ein weiterer Vorteil für viele Szenarien. So können die Raspberry Pis für das Starten von einem Netzwerkspeicher vorbereitet werden.

2 Beispielszenario

Dieser Abschnitt zeigt, welches Ziel der Workshop verfolgt, wie eine Kollaboration der Teilnehmer stattfinden kann und wie der technische Aufbau aussehen soll.

2.1 Inhalt des Workshops

In diesem Workshop sollen acht Teilnehmer mit Hilfe von Arduinos und Raspberry Pis Smart Objects konstruieren. Der Leiter des Workshops stellt die, zur Lösung der Aufgabe benötigten Materialien, zur Verfügung.

2.2 Zusammenarbeit

Um den Workshopteilnehmern die Zusammenarbeit zu erleichtern, soll ein geeignetes Werkzeug zum Teilen und Verwalten von Quellcode verwendet werden. Beispielcode und Dokumentation zu Programmen und Geräten kann über ein Netzlaufwerk bereitgestellt werden. Sollte gemeinsam an Code gearbeitet werden, bietet sich eine Quellcodeverwaltung an.

2.2.1 Netzlaufwerk

Konkret wird in diesem Fall ein NFS-Server verwendet. Dazu werden auf dem Raspberry Pi die folgenden Kommandos ausgeführt, um eine Verknüpfung auf dem Desktop zu erzeugen:

```
$ mkdir /home/pi/Desktop/docs
# mount mount nfs-server:/workshop/docs /home/pi/↩
    Desktop/docs
```

2.2.2 Quellcodeverwaltung

Als Quellcodeverwaltung kann Gogs[1] verwendet werden. Es wird von den Entwicklern als *painless self-hosted Git service* beschrieben. Der Vorteil dieser Methode ist, dass die Logins für die Workshop-Teilnehmer vorbereitet werden können.

3 Technischer Aufbau

In diesem Abschnitt werden die konkreten, technischen Voraussetzungen geklärt, die für den Workshop notwendig sind. Pro Teilnehmer und Leiter wird ein Raspberry Pi und ein Arduino benötigt. Auf dem Server (auch PC oder Laptop) kann der Leiter das Netzlaufwerk und die Quellcodeverwaltung bereitstellen. Router und Switch bilden die Netzwerkinfrastruktur.

Die Abbildung 1 zeigt den Aufbau der Komponenten:

[1] `https://gogs.io/`

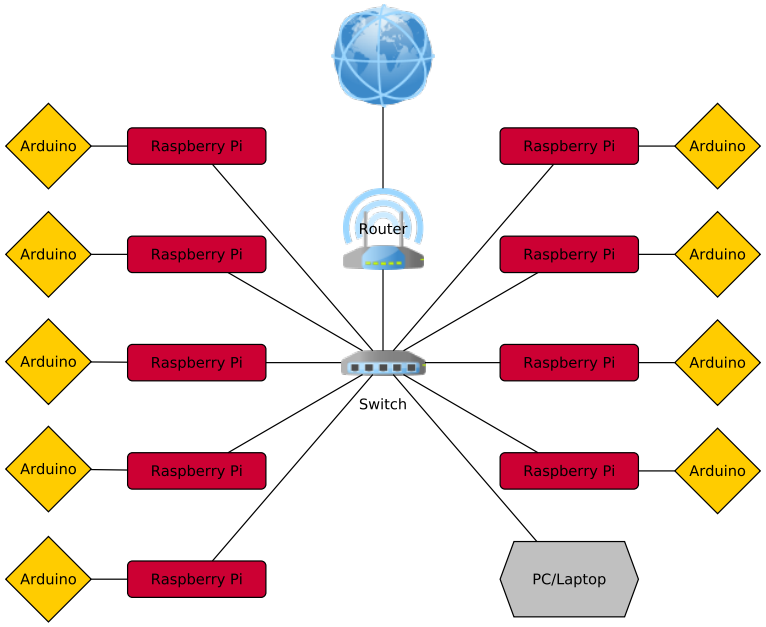

Abbildung 1: Aufbau

3.1 Hardware

Für den Workshop mit acht Teilnehmern und einem Leiter werden folgende Hardware-Komponenten benötigt:

- 9 Raspberry Pis
- 9 Arduinos
- 1 Server (auch PC oder Laptop) für das NFS
- 1 Router
- 1 Switch
- div. Kabel zum Verbinden

4 Raspberry Pi SD-Karten-Abbild

Der Raspberry Pi wird üblicherweise mit Hilfe einer SD-Karte[2] gestartet. Auf dieser befindet sich das Betriebssystem.

[2] `https://de.wikipedia.org/wiki/SD-Karte`

4.1 Alternativen

Dieser Abschnitt zeigt Quellen für fertige Abbilder und stellt Werkzeuge vor, die SD-Karten-Abbilder für den Raspberry Pi erzeugen können.

4.1.1 Fertige Abbilder

Die Raspberry Pi Foundation stellt auf der Downloadseite[3] SD-Karten-Abbilder bereit und verweist auf alternative Systeme.
Systeme mit grafischer Oberfläche sind beispielsweise Raspbian[4], NOOBS[5] und Ubuntu MATE[6]. Für den Serverbetrieb oder das IoT bieten sich Ubuntu Core[7] oder Windows 10 IoT Core[8] an.

4.1.2 PiBakery

PiBakery ist ein Werkzeug zum Konfigurieren von SD-Karten-Abbildern. Es besitzt eine grafische Benutzeroberfläche, die auf Blocks basiert. [1] Dieses Werkzeug bietet einige Vorteile: Es wird sehr gut gepflegt, es lässt sich einfach bedienen und ist Open-Source. Dem gegenüber steht, dass es schwer anpassbar ist, da eigene Scripts über Blocks-Logik hinzugefügt werden. Der folgende Screenshot zeigt, wie ein SD-Karten-Abbild über PiBakery konfiguriert werden kann.

[3] `https://www.raspberrypi.org/downloads/`
[4] `https://www.raspberrypi.org/downloads/raspbian/`
[5] `https://www.raspberrypi.org/downloads/noobs/`
[6] `https://ubuntu-mate.org/raspberry-pi/`
[7] `https://developer.ubuntu.com/en/snappy/start/`
[8] `http://ms-iot.github.io/content/en-US/Downloads.htm`

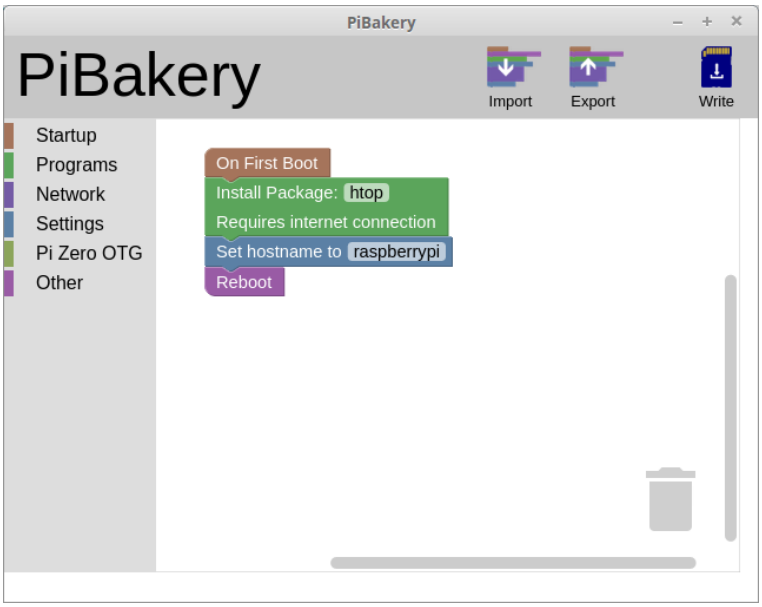

Abbildung 2: PiBakery

4.1.3 HypriotOS

HypriotOS wurde entwickelt, um Docker auf dem Raspberry Pi oder anderen ARM-basierten Geräten laufen zu lassen.[2] Es hat den Vorteil, dass es sehr gut gepflegt wird. Jedoch bietet es nur eine Basisinstallation des Betriebssystems und eigene Pakete und Anpassungen sind nur schwer zu integrieren. Zudem enthält es nicht benötigte Pakete.

5 Eigene SD-Karten-Abbilder erzeugen

Dieser Abschnitt zeigt, wie mit einer eigenen Lösung SD-Karten-Abbilder mit dem Betriebssystem für den Raspberry Pi erzeugt werden.

5.1 Voraussetzungen

Für das Ausführen des Werkzeugs sind nur wenige Voraussetzungen zu treffen. So muss auf dem Rechner Docker installiert sein, da es für das Bauen des SD-Karten-Abbilds verwendet wird. Docker erlaubt es in diesem Fall, das Abbild plattformunabhängig zu erstellen und ist für Linux, Windows und Mac OS verfügbar[4]. Weiterhin wird Docker Compose benötigt, welches es ermöglicht Anwendungen zu starten, die aus mehreren Containern bestehen[3]. Eine Internetverbindung ist erforderlich, um das Betriebssystem und zusätzliche Pakete für den Raspberry Pi herunterzuladen.

5.2 Aufbau

Die folgende Grafik zeigt, in welcher Reihenfolge die Scripts ausgeführt werden. Das Bauen wird durch *build_image* gestartet. Alle nachfolgenden Scripts werden in einem Docker-Container ausgeführt. Die Namen der Schritte entsprechen dem Dateinamen aus dem Verzeichnis *build_scripts* ohne das *.sh* am Ende.

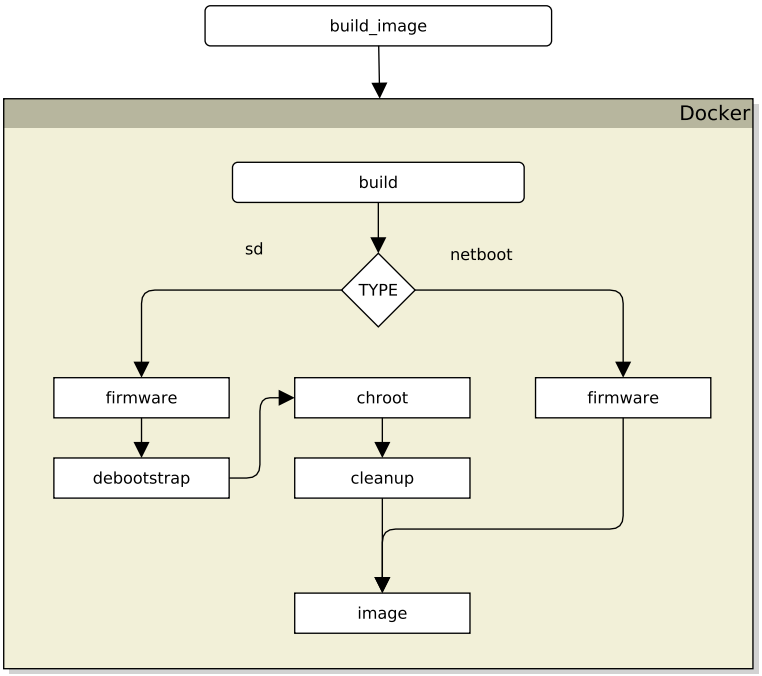

Abbildung 3: Ausführungsreihenfolge der Scripts

5.2.1 build

In diesem Schritt wird die Konfiguration eingelesen und alle definierten Variablen werden als Umgebungsvariablen zur Verfügung gestellt. Sollte die Konfigurationsdatei nicht vorhanden sein, wird der Vorgang mit einer entsprechenden Meldung abgebrochen. Anschließend werden abhängig von der *TYPE*-Variable (*sd* oder *netboot*), die nachfolgenden Schritte ausgewählt, da beispielsweise für das *netboot*-Abbild weniger Scripts ausgeführt werden müssen.

5.2.2 debootstrap

Debootstrap wird in diesem Fall zusammen mit QEMU[9] genutzt, um das Debian Betriebssystem für die Architektur des Raspberry Pi in ein Verzeichnis zu

[9] `http://wiki.qemu.org/Main_Page`

installieren.[6] Da dieser Vorgang sehr lange dauert, kann ein Archiv mit allen heruntergeladenen Dateien erzeugt werden.

5.2.3 firmware

In diesem Schritt wird die Firmware des Raspberry Pis heruntergeladen. Sollte die benötigte Version bereits im Dateisystem liegen, wird dieser Vorgang übersprungen. Anschließend wird die Firmware entpackt.

5.2.4 chroot

Dieses Script prüft zunächst, ob alle Dateien für die Anpassungen aus dem Parameter *CHROOT_SCRIPTS* der Konfigurationsdatei vorhanden sind. Anschließend werden sie ausgeführt.

5.2.5 cleanup

Um das spätere Abbild möglichst klein zu halten, werden nicht mehr benötigte Dateien entfernt.

5.2.6 image

Im letzten Schritt wird das Abbild für die SD-Karte erzeugt. Es wird der Platzbedarf für das Betriebssystem berechnet und anschließend eine Datei mit entsprechender Größe angelegt. In dieser werden zwei Partitionen erzeugt und formatiert. In die erste Partition wird die Firmware und in die zweite das Betriebssystem kopiert. Abschließend wird ein Archiv erstellt, das für den Netzwerkstart benötigt wird.

5.3 Konfiguration und Erweiterung

Zur Konfiguration wird eine Datei im JSON-Format[10] verwendet. Diese Datei enthält genau ein JSON-Objekt. Die Eigenschaften des Objekts werden beim Schritt *build* (siehe 3 und 5.2.1) in Umgebungsvariablen umgewandelt, sodass diese auch Kommandos und Verweise auf vorher definierte Variablen enthalten können.
Nachfolgend wird eine Beispielkonfiguration gezeigt:

```
1 {
2   "RASPBIAN_DIST": "jessie",
3   "RASPBIAN_VARIANT": "minbase",
4   "USE_DEBOOTSTRAP_ARCHIVE": "true",
5   "RASPBIAN_MIRROR": "http://mirror.netcologne.de/raspbian/↩
        raspbian",
6   "FIRMWARE_VERSION": "next",
7   "FIRMWARE_ZIP_URL": "https://github.com/raspberrypi/↩
        firmware/archive/${FIRMWARE_VERSION}.zip",
```

[10] https://de.wikipedia.org/wiki/JavaScript_Object_Notation

```
 8    "IMAGENAME": "rpi_sd_gui.img",
 9    "ROOTFS_FILENAME": "rpi_sd_gui.tar.gz",
10    "BUILD_DATE": "`date +%Y-%m-%d_%H:%M:%S`",
11    "LOG_FOLDER": "/log/${BUILD_DATE}",
12    "BOOT_PART_SIZE": "32",
13    "TYPE": "sd",
14    "TAG": "Raspberry Pi ${TYPE} image built on ${BUILD_DATE↩
          }",
15    "NFS_IP": "192.168.1.148",
16    "NFS_ROOTFS_DIR": "/exports/rootfs/%s",
17    "CHROOT_SCRIPTS": [
18      "base_jessie",
19      "network",
20      "pi_user",
21      "gui",
22      "extra",
23      "arduino",
24      "firmwareupdate",
25      "chromium",
26      "clean"
27    ],
28    "PI_HOSTNAME": "testpi"
29  }
```

Die wichtigsten Parameter sind zunächst *NFS_IP* und *NFS_ROOTFS_DIR*. Dort wird festgelegt, woher der Raspberry Pi das Betriebssystem laden soll. Mit dieser Konfiguration (Zeile 17-27) werden Basispakete für Debian Jessie, eine grafische Oberfläche, zusätzliche Pakete, die Arduino IDE und Chromium installiert. Zudem wird das Netzwerk konfiguriert, ein Benutzer *pi* angelegt, ein automatisches Firmwareupdate eingerichtet und abschließend aufgeräumt. Dies erfolgt im Schritt *chroot* (siehe 5.2.4).

Der Parameter *TYPE* gibt an, ob der Raspberry Pi von der SD-Karte oder das Netzwerk gestartet werden soll. Die möglichen Werte sind *sd* und *netboot*. Das Starten mit einem USB-Stick ist nur mit den Raspberry Pi 3 möglich und ist in Planung.

Für den Netzwerkstart ist weiterhin eine SD-Karte erforderlich. Auf dieser befindet sich eine kleine Partition, die den Kernel und eine Datei mit den dazugehörigen Parametern enthält. Die Größe dieser Partition kann über *BOOT_PART_SIZE* angegeben werden. Der Platzbedarf dieser Dateien liegt momentan bei 21 MB.

5.3.1 Pakete installieren

Soll das System um weitere Pakete oder Einstellungen erweitert werden, muss ein Shell-Skript im Verzeichnis *customization* erstellt werden. Ein Script (*customization/arduino.sh*) für die Beispielinstallation der Arduino IDE sieht wie folgt aus:

```
#!/bin/bash -e
```

```
DEBIAN_FRONTEND=noninteractive apt-get -qq -y install ↩
    --no-install-recommends \
   arduino

usermod -a -G dialout pi
```

Dieses wird in der Konfigurationsdatei (s.o.) in der Zeile 23 referenziert.

5.3.2 Systemstart ändern

Um ein Script beim Systemstart des Raspberry Pis auszuführen, kann das folgende Beispiel zur Aktualisierung der Firmware als Referenz herangezogen werden:

```
cat <<'EOF' > /usr/local/bin/firmwareupdate
#!/bin/bash
sleep 10

mount -o remount,rw /boot
touch /boot/lastupdate

test "x$(date +%Y-%m-%d)" == "x$(cat /tmp/lastupdate)" ↩
    && exit 0

cd /tmp/
wget https://github.com/raspberrypi/firmware/archive/↩
    next.zip
unzip next.zip 'firmware-next/boot/*' -d .
cp -R firmware-next/boot/* /boot/
date +%Y-%m-%d > /tmp/lastupdate
reboot

EOF
chmod +x /usr/local/bin/firmwareupdate

sed -i '/^exit/ d' /etc/rc.local
echo /usr/local/bin/firmwareupdate >> /etc/rc.local
echo "exit 0" >> /etc/rc.local
```

Dieses Script wird ebenfalls im Arbeitsschritt *chroot* ausgeführt und legt ein Script */usr/local/bin/firmwareupdate*, welches in */etc/rc.local* eingetragen wird. Dieses wird in der Konfigurationsdatei (s.o.) in der Zeile 24 referenziert.

5.4 Bauen

Das Bauen wird über das folgende Kommando gestartet:

```
$ ./build_image.sh ./config/config.json
```

Der Parameter *./config/config.json* stellt in diesem Fall den Pfad zur Konfigurationsdatei aus dem vorherigen Abschnitt dar.

Die folgende Tabelle zeigt die benötigten Zeiten für das Bauen der SD-Karten-Abbilder, die je nach Menge der zu installierenden Pakete variiert.

Typ	**Größe**	**Buildzeit**	**Schreiben auf SD**[11]
sd	866 MB	11m14s	87s
sd + gui	2,3 GB	36m45s	3m50s
netboot	42 MB	4s	4s

Tabelle 1: Zeiten für das Bauen der SD-Karten Abbilder

Das Bauen des *netboot*-Abbilds dauert nur wenige Sekunden, da nur die Firmware und Konfigurationsdateien auf die SD-Karte geschrieben werden müssen.

5.5 Artefakte

Das Ergebnis des Bauens liegt im Verzeichnis *output*. Dort findet man ein Abbild der SD-Karte (Dateiname *IMAGENAME*, wie im *config.json* konfiguriert) und ein *rootfs.tar.gz*. Das *rootfs.tar.gz* beinhaltet alle Dateien, die in die Hauptpartition des SD-Karten-Abbilds kopiert wurden. Wie hierüber das Starten über das Netzwerk realisiert werden kann, wird im Abschnitt 5.7 beschrieben.

5.6 Schreiben auf SD

Unter Linux kann das Abbild mit dem Kommandozeilenwerkzeug *ddrescue* auf die SD-Karte geschrieben werden:

```
# sudo ddrescue -D --force output/rpi.img /dev/sdX
```

Für andere Plattformen kann die Anleitung unter `http://elinux.org/RPi_Easy_SD_Card_Setup` verwendet werden.

5.7 Starten über das Netzwerk

In diesem Abschnitt wird erklärt, welche Schritte ausgeführt werden müssen, um einen Raspberry Pi über das Netzwerk starten zu können.

5.7.1 Vorbereitungen

Für das Starten über das Netzwerk, müssen die IP-Adressen der Raspberry Pis bekannt sein. Diese können über den DHCP-Server fest vergeben werden. Dazu müssen jedoch die MAC-Adressen aus den Raspberry Pis ausgelesen werden, da diese nicht auf der Platine oder dem Karton abgedruckt sind. Das Auslesen kann leicht über das SD-Karten-Abbild für das Starten über das Netzwerk erledigt werden. Der Raspberry Pi baut nach dem Start eine Netzwerkverbindung auf und bekommt eine IP-Adresse. mit dem Kommando

[11] bei 10 MB/s

```
$ sudo arp-scan --interface eth0 --localnet | grep b8:27:eb
192.168.1.228    b8:27:eb:2f:29:aa          (Unknown)
```

können Geräte, die von der Raspberry Pi Foundation produziert wurden im Netzwerk gesucht werden. [12]

5.7.2 SD-Karten-Abbild

Das SD-Karten-Abbild für den Netzwerkstart ist mit etwa 42 MB sehr klein. Es wird, wie im Abschnitt 5.3 gezeigt, konfiguriert. Der Parameter *TYPE* muss hierbei auf *netboot* gesetzt werden. Beim Bauen des Abbilds wird eine Partition erzeugt, die den Kernel und verschiedene Konfigurationsdateien enthalten, die den Start ermöglichen. Die Datei *cmdline.txt* enthält die Kernel-Parameter. Über die folgenden Einträge wird ein Start vom Server mit der IP 192.168.1.148 ermöglicht. Das Root-Filesystem wird hierbei aus dem Verzeichnis */exports/%s* geladen. das %s ist ein Platzhalter für die IP des Raspberry Pis.

```
[...] root=/dev/nfs rootfstype=nfs nfsroot=192.168.1.148:/exports/rootfs/%s,udp,vers=3 ip=dhcp [...]
```

5.7.3 NFS einrichten

Um das NFS für einen Raspberry Pi einzurichten, müssen die folgenden Kommandos ausgeführt werden. Die IP 192.168.1.228 wurde über das Kommando *arp-scan* herausgefunden. Das Verzeichnis */exports* ist das Hauptverzeichnis des NFS-Servers, in dem die Dateien zum Starten des Betriebssystems liegen.

```
IP=192.168.1.228
sudo mkdir -p /exports/rootfs
sudo tar xzf output/rpi_sd_gui.tar.gz --directory /tmp
sudo mv /tmp/rootfs /exports/rootfs/${IP}
sudo cp -R home/. /exports/rootfs/${IP}/home/pi/
sudo chown -R 1000:1000 /exports/rootfs/${IP}/home/pi/
```

Für den NFS-Server bietet sich in diesem Fall an, ein Dateisystem mit Deduplizierung zu verwenden. Da das Root-Filesystem für jeden Raspberry Pi separat gespeichert ist, liegen auf dem NFS zunächst viele identische Dateien. Dies ist jedoch nötig, wenn dem Benutzer die Möglichkeit geboten werden soll, benötigte Pakete nachzuinstallieren.

5.7.4 Pakete nachträglich installieren

Sollte ein Paket fehlen, muss das Dateisystem als beschreibbar eingehängt werden. Dann kann das Paket installiert werden:

[12] `http://hwaddress.com/?q=B827EB000000`

```
sudo mount -o remount,rw /
sudo apt-get update
sudo apt-get install PAKET
```

5.7.5 Betriebssystem auf SD-Karte

Sollte es nötig sein, das Betriebssystem von der SD-Karte zu laden, liegt im Homeverzeichnis des Benutzers pi ein Script *sdboot.sh*, das es ermöglicht alle Dateien auf die SD-Karte zu kopieren. Nach einem Neustart wird die SD-Karte zum Starten verwendet.

5.7.6 Firmware aktualisieren

Nach dem erstmaligen Beschreiben der SD-Karte kann diese im Raspberry Pi verbleiben. Die Aktualisierung kann auch im laufenden Betrieb erfolgen. Es bietet sich an, vor jedem Workshop alle Geräte zu starten und sich über SSH anzumelden. Zunächst muss die Bootpartition als beschreibbar eingehängt werden:

```
sudo mount -o remount,rw /boot
```

Anschließend kann die Firmware aus den offiziellen Quellen[13] heruntergeladen werden und alle benötigten Dateien auf die SD-Karte kopiert werden:

```
cd /tmp/
wget https://github.com/raspberrypi/firmware/archive/↩
    next.zip
unzip next.zip 'firmware-next/boot/*' -d .
sudo cp -R firmware-next/boot/* /boot/
sudo reboot
```

Zum Schluss erfolgt ein Neustart, um die Firmware zu aktivieren.

5.8 Designentscheidungen

Um das Starten des Raspberry Pis über das Netzwerk möglichst generisch zu halten, wird in der *cmdline.txt* in der Boot-Partition ein Platzhalter verwendet.

```
[...] root=/dev/nfs rootfstype=nfs nfsroot↩
    =192.168.1.148:/exports/%s,udp,vers=3 ip=dhcp [...]
```

Der Kernel-Parameter *nfsroot* mit Platzhalter *%s* steht für die IP-Adresse des Clients[5]. Der DHCP-Server im Router vergibt die IP-Adresse und der Raspberry Pi holt vom den NFS-Server die Dateien zum Starten des Betriebssystems.

[13] `https://github.com/raspberrypi/firmware`

5.9 Zeitersparnis

Für die SD-Karten wird eine Schreibgeschwindigkeit von 10 MB/s angenommen. Das wechseln der Speicherkarte im Kartenleser dauert etwa 30 Sekunden. Daraus ergeben sich folgende Zeiten:

	Raspbian	**sd**	**netboot**
Größe in MB	4100	2300	42
Schreibdauer in Sek. pro Karte	410	230	4,2
Wechseln der Karte in Sek.	30	30	30
Vorbereitung auf Server in Sek.	-	-	60
Summe in Sek. pro Karte	440	260	94,2
Summe in Min. für 9 Karten	**66**	**39**	**14,13**

Tabelle 2: Vergleich der Vorbereitungszeiten für Systeme mit grafischer Benutzeroberfläche

Im Falle der selbstgebauten Abbilder müssen die Zeiten aus der Tabelle 5.4 berücksichtigt werden. Diese können jedoch nicht addiert werden, da dies ein passiver Vorgang ist, der in weniger als einer Minute angestoßen werden kann. Sollte der Netzwerkstart verwendet worden sein, entfällt beim nächsten Workshop das manuelle Beschreiben der SD-Karte.

Die Vorbereitungszeit beim Raspbian-Abbild erhöht sich, sobald Modifikationen an der Installation vorgenommen werden müssen. Um diese Zeit möglichst gering zu halten, kann ein Raspberry Pi vorbereitet und angepasst werden und der Inhalt der Speicherkarte auf alle anderen kopiert werden. Eine weitere Möglichkeit der nachträglichen Anpassung ist, alle Raspberry Pis mit dem nicht konfigurierten Abbild zu starten und ein Script über SSH auszuführen.

6 Weiterentwicklung

Um das Netzlaufwerk portabel zu machen, kann der NFS-Server in einem Docker-Container gestartet werden.

Die Konfiguration wird momentan in der JavaScript Object Notation gehalten. Um dies zu vereinfachen und eine grafische Konfigurationsoberfläche zu erhalten, kann eine Integration in PiBakery angestrebt werden.

7 Fazit

Der Raspberry Pi ist eine gute Plattform, um kleine Projekte zu realisieren. Für einen Workshop mit Arduinos eignet er sich ebenfalls gut, da er ausreichend Leistung mitbringt, um die Entwicklungsumgebung laufen zu lassen. Der Vorteil dieser Lösung liegt darin, dass alle Teilnehmer eine einheitliche Umgebung zur Verfügung gestellt bekommen und sofort loslegen können. Dem Leiter

des Workshops wird zudem ein großer Teil an Verwaltungsaufwand abgenommen, da durch die Möglichkeit des Netzwerkstarts nur einmalig SD-Karten mit einem sehr kleinen Abbild beschrieben werden müssen.

Literatur

[1] Ferguson, D.: Pibakery, https://github.com/davidferguson/pibakery

[2] Hypriot: About us, https://blog.hypriot.com/about/

[3] Inc., D.: Docker compose, https://docs.docker.com/compose/

[4] Inc., D.: Welcome to the docs, https://docs.docker.com/#components

[5] Kuhlmann, G., Mares, M., Schottelius, N., Horms: Mounting the root filesystem via NFS (nfsroot) (1996), https://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt

[6] O'Connor, M.: Debootstrap, https://wiki.debian.org/Debootstrap

Alle Links wurden zuletzt am 13. Januar 2017 besucht.